```
import random

#Function to calculate list of list for all the items

init=[]
initx=[]
def func1(rows,str):
  for l in range(0,len(rows)):
    row=rows[l].split(' ')
    init.append(row[1:])
    row[0]=str
    initx.append(row)



#fetching data of all .txt files

f = open("countries", 'r');
rows = f.read().splitlines();
f.close();
func1(rows,"countries")

f = open("animals", 'r');
rows = f.read().splitlines();
f.close();
func1(rows,"animals")

f = open("fruits", 'r');
rows = f.read().splitlines();
f.close();
func1(rows,"fruits")

f = open("veggies", 'r');
rows = f.read().splitlines();
f.close();
func1(rows,"veggies")


random.shuffle(init)
random.shuffle(initx)

for i in range(0,len(initx)):
  for j in range(1,len(initx[0])):
    initx[i][j]=float(initx[i][j])


for i in range(0,len(init)):
  for j in range(0,len(init[0])):
    init[i][j]=float(init[i][j])
```

```
category=[]

for item in initx:
  temp=item[0]
  flag=False
  for i in range(0,len(category)):
    if category[i]==temp :
      flag=True
  if flag == False:
    category.append(temp)

CATEGORY={}

for i in range(0,len(category)):
  CATEGORY.update({category[i]:0})

print(CATEGORY)
```

```
{'fruits': 0, 'animals': 0, 'veggies': 0, 'countries': 0}
```

```
#K means algorithm

import random
import math
from sklearn import *



#EuclideanDistance

def EuclideanDistance(x,y):
  sum=0
  for i in range(0,len(x)):
    sum+= math.pow(float(x[i])-float(y[i]),2)
  return math.sqrt(sum)



#Manhattan distance

def ManhattanDistance(x,y):
  sum=0
```

```
  for i in range(0,len(x)):
    sum+=(abs(x[i]-y[i]))
  return sum



#Normalization

def normalize(init):
  init1= preprocessing.normalize(init, norm='l2', axis=1, copy=True, return_norm=False)
  return init1

#Cosine similarity

def cosine_similarity(x,y):
  Mod_X=0
  Mod_Y=0
  Num=0
  for i in range(0,len(x)):
    Num+=x[i]*y[i]
    Mod_X+=math.pow(x[i],2)
    Mod_Y+=math.pow(y[i],2)
  Mod_X=math.sqrt(Mod_X)
  Mod_Y=math.sqrt(Mod_Y)
  similarity=Num/(Mod_X*Mod_Y)
  return similarity


def k_means_init(k):
  initial_mean=random.sample(init, k)
  list_of_clusters = [[] for i in range(k)]
  for x in range(0,k):
    list_of_clusters[x].append(initial_mean[x])
  return(k_means_manhattan(k,initial_mean,list_of_clusters))
```

```
def k_means(k,initial_mean,list_of_clusters):
  while(True):

    for item in init:
      index=0
      distance=999999
      for i in range(0,len(initial_mean)):
        temp=EuclideanDistance(item,initial_mean[i])
        if temp < distance:
          distance=temp
```

```
          distance=temp
          index=i
      if distance !=0:
        list_of_clusters[index].append(item)


    updated_mean=[]



    for List in list_of_clusters:
      lof=len(List[0])

      length=len(List)
      l=[]
      for i in range(0,lof):
        sum=0
        for j in range(0,length):
          sum+=List[j][i]
        sum/=length
        l.append(sum)
      updated_mean.append(l)

   # print(updated_mean)
   # print(initial_mean)

    flag=False
    for i in range(0,len(initial_mean)):
      for j in range(0,len(initial_mean[0])):
        X=round(initial_mean[i][j],6)
        Y=round(updated_mean[i][j],6)
        if X != Y:
          flag= True

    if flag== True:
      print("Hi")
      initial_mean=updated_mean
      list_of_clusters = [[] for i in range(k)]
      continue
    else:
      print("bye")

      break

  print(list_of_clusters)
  return list_of_clusters
```

```
#Manhattan Distance
```

```
def k_means_manhattan(k,initial_mean,list_of_clusters):
  while(True):

    for item in init:
      index=0
      distance=999999
      for i in range(0,len(initial_mean)):
        temp=ManhattanDistance(item,initial_mean[i])
        if temp < distance:
          distance=temp
          index=i
      if distance !=0:
        list_of_clusters[index].append(item)


    updated_mean=[]



    for List in list_of_clusters:
      lof=len(List[0])

      length=len(List)
      l=[]
      for i in range(0,lof):
        sum=0
        for j in range(0,length):
          sum+=List[j][i]
        sum/=length
        l.append(sum)
      updated_mean.append(l)

   # print(updated_mean)
   # print(initial_mean)

    flag=False
    for i in range(0,len(initial_mean)):
      for j in range(0,len(initial_mean[0])):
        X=round(initial_mean[i][j],6)
        Y=round(updated_mean[i][j],6)
        if X != Y:
          flag= True

    if flag== True:
      print("Hi")
      initial_mean=updated_mean
      list_of_clusters = [[] for i in range(k)]
      continue
```

```
    else:
      print("bye")

      break

  print(list_of_clusters)
  return list_of_clusters
```

```
#Cosine Similarity

import statistics as st


def k_means_cosine(k,initial_mean,list_of_clusters):
  while(True):

    for item in init:
      index=0
      similarity=-9999
      for i in range(0,len(initial_mean)):
        temp=cosine_similarity(item,initial_mean[i])
        if temp > similarity:
          similarity=temp
          index=i
      #if distance !=0:
      list_of_clusters[index].append(item)


    updated_mean=[]



    for List in list_of_clusters:
      lof=len(List[0])

      length=len(List)
      l=[]
      for i in range(0,lof):
        med=0
        for j in range(0,length):
          p=[]
          p.append(List[j][i])
        med=st.median(p)
        l.append(med)
      updated_mean.append(l)

   # print(updated_mean)
   # print(initial_mean)
```

```
    flag=False
    for i in range(0,len(initial_mean)):
      for j in range(0,len(initial_mean[0])):
        X=round(initial_mean[i][j],6)
        Y=round(updated_mean[i][j],6)
        if X != Y:
          flag= True

    if flag== True:
      print("Hi")
      initial_mean=updated_mean
      list_of_clusters = [[] for i in range(k)]
      continue
    else:
      print("bye")

      break

  print(list_of_clusters)
  return list_of_clusters
```

```
import collections
def evaluate1(k,list_of_clusters,initx):
  count=0
  items_in_clusters = [[] for i in range(k)]
  i=0
  for items in list_of_clusters:
    for item in items:
      for ite in initx:
        item1=ite[1:]
        flag=False
        for j in range(0,len(item1)):
          if item[j]!= item1[j]:
            flag = True
        if flag==False :
          #print("Hi")
          items_in_clusters[i].append(ite[0])
    i+=1
  print(items_in_clusters)
  return items_in_clusters
```

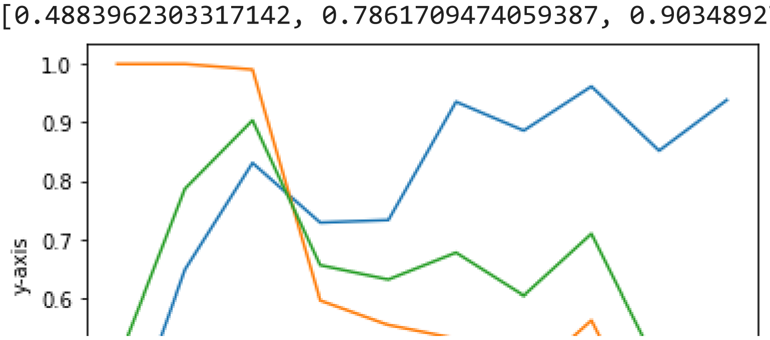
```
###Precision & Recall
import numpy as np

def graph(D) :
  items_in_clusters=D


  Total= (len(init)*(len(init)-1))/2
  TP_FP=0   #Total Positives
  TP=0
  FP=0
  FN=0
  precision=0
  recall=0
  F_score=0
  Total_Negatives=0

  for i in range (0,len(items_in_clusters)):
    x=len(items_in_clusters[i])
    TP_FP+=(x*(x-1))/2



  for items in items_in_clusters:
    for key in CATEGORY.keys():
      CATEGORY[key]=0

    for item in items:
      CATEGORY[item]+=1
    for key in CATEGORY.keys():
      TP+=(CATEGORY[key]*(CATEGORY[key]-1))/2


  print(TP)
  FP=TP_FP-TP
  Total_Negatives=Total-TP_FP
#print(len(init))


  LIST = []
  i=0
  j=0
  for items in items_in_clusters:
    l=[]
    for key in CATEGORY.keys():
      x=0
      for item in items:
        if key==item:
          x+=1
      l.append(x)
    LIST.append(l)
```

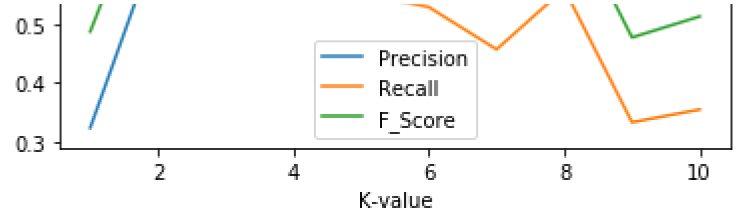
```
    LIST.append(l)

#print(LIST)
#print(items_in_clusters[3])
#print(CATEGORY)
  print(LIST)

  if len(LIST)==1:
    FN=0
  else:
    i=0
    for items in LIST:
      j=0
      for item in items:
        seq=np.array(LIST[i+1:])
        s=np.sum(seq,axis=0)
        FN=FN+ item*s[j]
        #print(s)
        j+=1

      if i==(len(LIST)-2):
        break
      else:
        #print(i)
        i+=1


  precision=TP/(TP_FP)
  recall=TP/(TP+FN)
  F_score=2*((precision*recall)/(precision+recall))

  return precision,recall,F_score
```

```
#Eucl Distance

import matplotlib.pyplot as plt

Precision=[]
Recall=[]
F_Score=[]

for i in range(1,11):
  list_of_clusters=k_means_init(i)
  items_in_clusters=evaluate1(i,list_of_clusters,initx)
  precision,recall,F_score=graph(items_in_clusters)
  Precision.append(precision)
  Recall.append(recall)
  F_Score.append(F_score)

print ("Precision:")
```

```
print("Precision:")
print(Precision)
print("Recall")
print(Recall)
print("F_Score")
print(F_Score)


k=[1,2,3,4,5,6,7,8,9,10]
plt.plot(k,Precision, label = "Precision")

plt.plot(k, Recall, label = "Recall")
plt.plot(k, F_Score, label = "F_Score")

# naming the x axis
plt.xlabel('K-value')
plt.ylabel('y-axis')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```

```
[['veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'fruits',
8079.0
[[0, 0, 5, 46], [0, 0, 54, 15], [50, 0, 0, 0], [0, 47, 0, 0], [0, 75, 0, 0], [0, 12, 0,
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
bye
[[[-0.11547, -0.0068501, -0.052155, 0.19664, -0.29308, 0.44315, -2.2152, -0.053042, -0.6
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
9916.0
[[0, 56, 0, 0], [0, 104, 0, 0], [0, 0, 0, 22], [0, 0, 3, 0], [0, 0, 2, 17], [50, 0, 0, 0
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
bye
[[[0.61345, 0.25027, 0.2146, 0.35803, -0.67823, 0.53957, -1.3509, 0.76372, 0.098648, 0.6
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
5869.0
[[0, 38, 0, 0], [0, 0, 5, 47], [18, 0, 0, 0], [32, 1, 1, 1], [0, 46, 0, 0], [0, 0, 53, 1
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
bye
[[[-0.11547, -0.0068501, -0.052155, 0.19664, -0.29308, 0.44315, -2.2152, -0.053042, -0.6
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
6255.0
[[0, 47, 0, 0], [0, 0, 50, 4], [0, 64, 0, 0], [0, 0, 3, 18], [16, 1, 2, 0], [0, 15, 0, 0
Precision:
[0.3230980499862675, 0.6476784731143329, 0.8305455236647026, 0.7284566361873095, 0.73276
Recall
[1.0, 1.0, 0.9904794287657259, 0.5959424232120594, 0.5541199138614984, 0.530601836110166
F_Score
[0.4883962303317142, 0.7861709474059387, 0.903489273714138, 0.6555701016146125, 0.631042
```

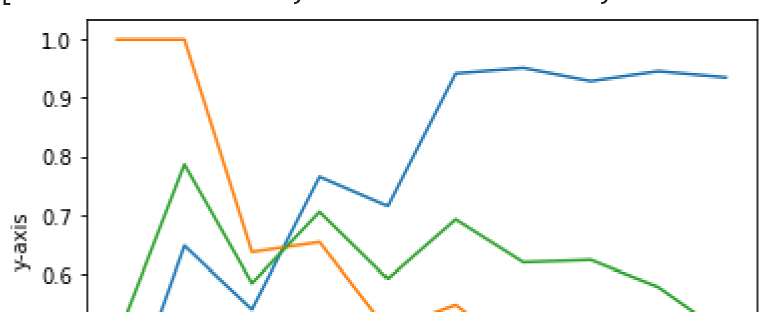


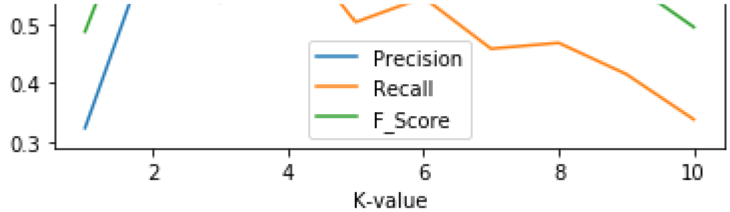

0.5
0.4
0.3
Precision
Recall
F_Score
2
4
6
8
10
K-value

```
#ManhattanDistance

import matplotlib.pyplot as plt

Precision=[]
Recall=[]
F_Score=[]

for i in range(1,11):
  list_of_clusters=k_means_init(i)
  items_in_clusters=evaluate1(i,list_of_clusters,initx)
  precision,recall,F_score=graph(items_in_clusters)
  Precision.append(precision)
  Recall.append(recall)
  F_Score.append(F_score)
print ("Precision:")
print(Precision)
print("Recall")
print(Recall)
print("F_Score")
print(F_Score)


k=[1,2,3,4,5,6,7,8,9,10]
```

```
plt.plot(k,Precision, label = "Precision")

plt.plot(k, Recall, label = "Recall")
plt.plot(k, F_Score, label = "F_Score")

# naming the x axis
plt.xlabel('K-value')
plt.ylabel('y-axis')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```

```
Hi
Hi
Hi
Hi
bye
[[[-0.10551, -0.19392, -0.3471, -0.085897, 0.33718, -1.1205, -0.86859, -0.3877, -0.71544
[['veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies'
8103.0
[[3, 0, 38, 0], [0, 0, 0, 41], [0, 0, 0, 76], [0, 0, 0, 44], [53, 0, 4, 0], [2, 0, 19, 0
Hi
Hi
Hi
Hi
Hi
bye
[[[-0.10551, -0.19392, -0.3471, -0.085897, 0.33718, -1.1205, -0.86859, -0.3877, -0.71544
[['veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies', 'veggies'
8281.0
[[1, 0, 30, 0], [0, 0, 2, 0], [53, 0, 9, 0], [1, 50, 0, 0], [0, 0, 0, 24], [0, 0, 0, 58]
Hi
Hi
Hi
Hi
Hi
bye
[[[0.90777, 0.24116, -0.12844, -0.20177, -0.24996, 0.22736, -0.61722, -0.24726, 0.21445,
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
7334.0
[[0, 0, 0, 32], [3, 0, 35, 0], [35, 0, 8, 0], [0, 0, 0, 81], [0, 0, 0, 48], [2, 0, 18, 0
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
bye
[[[0.38323, 0.066557, 0.28081, 0.12675, 0.0053225, 0.27666, 0.40096, -0.34148, 0.22719,
[['fruits', 'fruits', 'fruits', 'fruits', 'fruits', 'animals', 'fruits', 'veggies', 'fru
5961.0
[[8, 2, 6, 0], [29, 0, 1, 0], [0, 45, 0, 0], [4, 0, 47, 0], [17, 0, 7, 0], [0, 0, 0, 38]
Precision:
[0.3230980499862675, 0.6476784731143329, 0.5393290780822575, 0.7649671815951734, 0.71537
Recall
[1.0, 1.0, 0.6368582114926895, 0.6538592315538932, 0.5049302958177491, 0.54726283577014
F_Score
[0.4883962303317142, 0.7861709474059387, 0.5840501000441753, 0.7050627883528369, 0.59200
```

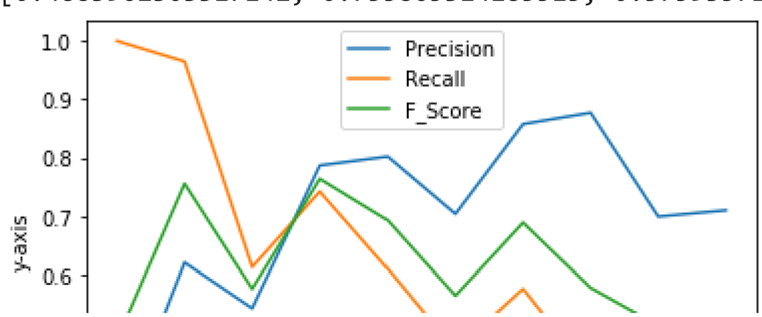


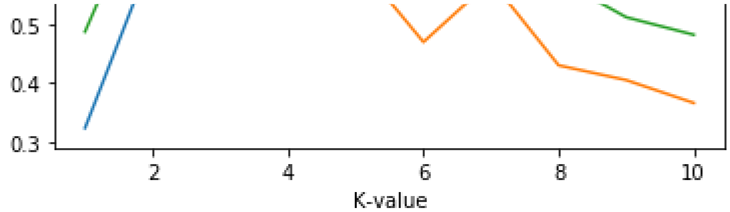

0.5
0.4
0.3
Precision
Recall
F_Score
2
4
6
8
10
K-value

```
#Cosine Graph


import matplotlib.pyplot as plt

Precision=[]
Recall=[]
F_Score=[]

for i in range(1,11):
  list_of_clusters=k_means_init(i)
  items_in_clusters=evaluate1(i,list_of_clusters,initx)
  precision,recall,F_score=graph(items_in_clusters)
  Precision.append(precision)
  Recall.append(recall)
  F_Score.append(F_score)

print ("Precision:")
print(Precision)
print("Recall")
print(Recall)
print("F_Score")
print(F_Score)


k=[1,2,3,4,5,6,7,8,9,10]
plt.plot(k,Precision, label = "Precision")

plt.plot(k, Recall, label = "Recall")
```

```
plt.plot(k, F_Score, label = "F_Score")

# naming the x axis
plt.xlabel('K-value')
plt.ylabel('y-axis')

# show a legend on the plot
plt.legend()

# function to show the plot
plt.show()
```

```
Hi
Hi
bye
[[[-0.28903, 0.46618, -0.3533, 0.35975, 0.33082, 0.0066478, -0.72074, -0.26066, 0.11912,
[['animals', 'animals', 'animals', 'animals', 'animals', 'animals', 'animals', 'animals'
10774.0
[[0, 1, 0, 49], [0, 43, 60, 0], [101, 0, 0, 0], [60, 0, 0, 0], [0, 15, 1, 1]]
Hi
Hi
bye
[[[0.70537, 0.36108, 0.048266, 0.76742, -0.34548, 0.44613, -1.0639, -0.057864, -0.38816,
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
8306.0
[[24, 0, 0, 0], [0, 2, 1, 50], [0, 54, 60, 0], [62, 1, 0, 0], [56, 0, 0, 0], [19, 2, 0,
Hi
bye
[[[-0.38017, -0.4503, -0.0047645, -0.97705, 0.2525, -0.49131, 0.13571, -0.1649, -0.12585
[['veggies', 'fruits', 'fruits', 'fruits', 'fruits', 'fruits', 'fruits', 'fruits', 'frui
10164.0
[[0, 11, 1, 0], [0, 13, 1, 1], [101, 0, 0, 0], [0, 1, 0, 48], [60, 0, 0, 0], [0, 7, 0, 1
Hi
Hi
Hi
bye
[[[-0.28903, 0.46618, -0.3533, 0.35975, 0.33082, 0.0066478, -0.72074, -0.26066, 0.11912,
[['animals', 'animals', 'animals', 'animals', 'animals', 'animals', 'animals', 'animals'
7600.0
[[0, 0, 0, 49], [0, 6, 1, 1], [57, 0, 0, 0], [0, 21, 17, 0], [23, 0, 0, 0], [81, 1, 0, 0
Hi
Hi
bye
[[[-0.057016, -0.24898, -0.20067, -0.16856, 0.050528, -0.086416, 0.70294, -0.28398, -0.1
[['fruits', 'fruits', 'countries'], ['countries', 'countries', 'countries', 'countries',
7153.0
[[1, 2, 0, 0], [54, 0, 0, 0], [0, 0, 1, 35], [56, 1, 0, 0], [29, 0, 0, 0], [21, 2, 0, 0]
Hi
Hi
Hi
bye
[[[-0.056452, 0.60837, 0.16597, -0.51307, -0.14357, 0.23802, -0.76605, 0.1929, 0.4151, 0
[['countries', 'countries', 'countries', 'countries', 'countries', 'countries', 'countri
6460.0
[[43, 0, 0, 0], [36, 0, 0, 0], [39, 1, 0, 0], [0, 13, 1, 1], [0, 1, 0, 49], [20, 0, 0, 0
Precision:
[0.3230980499862675, 0.6212646404203306, 0.5427425419904738, 0.7869502523431867, 0.80199
Recall
[1.0, 0.9649212286070498, 0.6134534738750992, 0.7422645358721524, 0.6105633004646945, 0.
F_Score
[0.4883962303317142, 0.755865314185515, 0.5759357292969062, 0.7639545056867891, 0.693307
```






```
for row in initx:
  if row[0]=='veggies':
    row[0]=0
  if row[0]=='countries':
    row[0]=1
  if row[0]=='fruits':
    row[0]=2
  if row[0]=='animals':
    row[0]=3
```