Q 19 , 20

```
import numpy as np
import math
import pandas as pd
```

Default title text

```
#@title Default title text
data=[12,4,5,21,7,14,15,18,20]
print(data)
x=np.sort(data)
print(x)
```

```
[12, 4, 5, 21, 7, 14, 15, 18, 20]
[ 4  5  7 12 14 15 18 20 21]
```

```
bin1=np.zeros((3,3))
bin2=np.zeros((3,3))
bin3=np.zeros((3,3))
```

MEAN

```
#MEAN MEAN MEAN
for i in range (0,9,3):
  k=int(i/3)
  mean=(x[i] + x[i+1] + x[i+2] )/3
  for j in range(3):
    bin1[k,j]=mean
print("-----------------Mean Bin:----------------- \n",bin1)
```

```
-----------------Mean Bin:-----------------
 [[ 5.33333333  5.33333333  5.33333333]
 [13.66666667 13.66666667 13.66666667]
 [19.66666667 19.66666667 19.66666667]]
```

```
#MEDIAN MEDIAN MEDIAN
for i in range (0,9,3):
  k=int(i/3)
  for j in range (3):
    bin2[k,j]=x[i+1]
print("-----------------Median Bin :----------------- \n",bin2)
```

```
-----------------Median Bin :-----------------
 [[ 5.  5.  5.]
 [14. 14. 14.]
 [20. 20. 20.]]
```

Q

```
import pandas as pd
import numpy as np
```

```
x=[27,23,34,45,49,69,62,55,21,18]
print(x)
```

```
[27, 23, 34, 45, 49, 69, 62, 55, 21, 18]
```

```
np.digitize(x,bins=[15,30,45,60,75])
```

```
array([1, 1, 2, 3, 3, 4, 4, 3, 1, 1])
```

```
df = pd.DataFrame({"age":x})
df.head()
```

	age
0	27
1	23
2	34
3	45
4	49

```
df['binned']=pd.cut(x=df['age'], bins=[0,15,30,45,60,75],labels=['child','young','adult','
```

```
df.head()
```

	age	binned
0	27	young
1	23	young
2	34	adult
3	45	adult
4	49	senior

```
import pandas as pd
import numpy as np
```

```
x=[12,23,34,45,50,110,80,60,150,125]
print(x)
```

```
[12, 23, 34, 45, 50, 110, 80, 60, 150, 125]
```

```
np.digitize(x,bins=[30,60,90,120,150,180])
```

```
array([0, 0, 1, 1, 1, 3, 2, 2, 5, 4])
```

```
df=pd.DataFrame({'salary':x})
df.head()
```

	salary
0	12
1	23
2	34
3	45
4	50

```
df['discritized']=pd.cut(x=df['salary'],bins=[0,30,60,90,120,150,180],labels=['boom','shak
df.head(15)
```

	salary	discritized
0	12	boom
1	23	boom
2	34	shake
3	45	shake
4	50	shake
5	110	room
6	80	the
7	60	shake
8	150	da
9	125	da

```
df['discritized']=pd.cut(x=df['salary'],bins=[0,30,60,90,120,150,180])
df.head(15)
```

	salary	discritized
0	12	(0, 30]
1	23	(0, 30]
2	34	(30, 60]
3	45	(30, 60]
4	50	(30, 60]
5	110	(90, 120]
6	80	(60, 90]
7	60	(30, 60]

```
points = [ [2, 3], [2, 4], [4, 1], [3, 2] ]

dis_matrix =[]
for p in points:
  list = []
  for j in points:
    list.append((abs(p[0]-j[0])+abs(p[1]-j[1])))
  dis_matrix.append(list)

for i in range(len(points)):
  for j in range(len(points)):
    if i<j: dis_matrix[i][j] = 0
# (dis_matrix)
print(dis_matrix)
```

```
[[0, 0, 0, 0], [1, 0, 0, 0], [4, 5, 0, 0], [2, 3, 2, 0]]
```

Double-click (or enter) to edit

```
points=[[2,3],[2,4],[4,1],[3,2]]

dis_matrix=[]
for p in points:
  list=[]
  for j in points:
    list.append((abs(p[0]-j[0])+abs(p[1]-j[1])))
  dis_matrix.append(list)

for i in range(len(points)):
  for j in range(len(points)):
    if i<j: dis_matrix[i][j]=0
(dis_matrix)
```

```
[[0, 0, 0, 0], [1, 0, 0, 0], [4, 5, 0, 0], [2, 3, 2, 0]]
```

min maxxmin max

```
import pandas as pd

df=pd.DataFrame([12000,145000,320000,45000,90000,134000,200000])


display(df)
min_max=df.copy()
# print(df.copy())

for i in min_max:
  min_max[i]=(min_max[i]-min_max[i].min())/(min_max[i].max()-min_max[i].min())

print(min_max)
```

```
        0
0   12000
1  145000
2  320000
3   45000
4   90000
5  134000
6  200000
          0
0  0.000000
1  0.431818
2  1.000000
3  0.107143
4  0.253247
5  0.396104
6  0.610390
```

```
import pandas as pd
import numpy as np

# dictionary of lists
dict = {'Stud Id':[1,2,3,4,5,6,7],
        'Stud Braanch': ['AI','COMP','COMP','COMP','AI','AI','AI'],
        'Score':[17,np.nan,15,16,18,15,np.nan]}

# creating a dataframe from dictionary
df = pd.DataFrame(dict)
print(df)

# display tuples with missing values
print(df.Score.isnull())

# filling missing value using fillna()
print(df.fillna(0))
```

```
   Stud Id Stud Braanch  Score
0        1           AI   17.0
1        2         COMP    NaN
2        3         COMP   15.0
3        4         COMP   16.0
4        5           AI   18.0
5        6           AI   15.0
6        7           AI    NaN
0    False
1     True
2    False
3    False
4    False
5    False
6     True
Name: Score, dtype: bool
   Stud Id Stud Braanch  Score
0        1           AI   17.0
1        2         COMP    0.0
2        3         COMP   15.0
3        4         COMP   16.0
4        5           AI   18.0
5        6           AI   15.0
6        7           AI    0.0
```

```
import pandas as pd
import numpy as np

df=pd.DataFrame([[101,120000],
                 [102,145000],
                 [103,320000],
                 [104,45000],
                 [105,90000],
                 [106,134000],
                 [107,200000]],
                columns=['EMP ID','Salary'])


display(df)

zscore=df.copy()

for i in zscore.columns:
  zscore[i]=(zscore[i]-zscore[i].mean())/zscore[i].std()
display(zscore)
```

	EMP ID	Salary
0	101	120000
1	102	145000
2	103	320000
3	104	45000
4	105	90000
5	106	134000
6	107	200000

	EMP ID	Salary
0	-1.38873	-0.344670
1	-0.92582	-0.062814

```
pip install efficient-apriori
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: efficient-apriori in /usr/local/lib/python3.7/dist-pac
```

```
6  -1.38873  -0.557270
```

```
from efficient_apriori import apriori

transactions=[(1,3,4,2),
              (2,3,5,1),
              (1,2,3,4),
              (2,5,3),
              (1,2,3,4)]

min_support=0.7
min_confidence = 0
itemsets, rules = apriori(transactions, min_support=min_support, min_confidence=min_confid
print(itemsets)
for rule in rules:
  print(rule)
```

```
{1: {(1,): 4, (3,): 5, (2,): 5}, 2: {(1, 2): 4, (1, 3): 4, (2, 3): 5}, 3: {(1, 2, 3)
{2} -> {1} (conf: 0.800, supp: 0.800, lift: 1.000, conv: 1.000)
{1} -> {2} (conf: 1.000, supp: 0.800, lift: 1.000, conv: 0.000)
{3} -> {1} (conf: 0.800, supp: 0.800, lift: 1.000, conv: 1.000)
{1} -> {3} (conf: 1.000, supp: 0.800, lift: 1.000, conv: 0.000)
{3} -> {2} (conf: 1.000, supp: 1.000, lift: 1.000, conv: 0.000)
{2} -> {3} (conf: 1.000, supp: 1.000, lift: 1.000, conv: 0.000)
{2, 3} -> {1} (conf: 0.800, supp: 0.800, lift: 1.000, conv: 1.000)
{1, 3} -> {2} (conf: 1.000, supp: 0.800, lift: 1.000, conv: 0.000)
{1, 2} -> {3} (conf: 1.000, supp: 0.800, lift: 1.000, conv: 0.000)
{3} -> {1, 2} (conf: 0.800, supp: 0.800, lift: 1.000, conv: 1.000)
{2} -> {1, 3} (conf: 0.800, supp: 0.800, lift: 1.000, conv: 1.000)
{1} -> {2, 3} (conf: 1.000, supp: 0.800, lift: 1.000, conv: 0.000)
```

```
import pandas as pd
import numpy as np

def mean(arr):
  m=0
  for i in arr:
    m+=i
  res=m//len(arr)
  return res

def k_mean(arr,m1,m2):
  cluster1=[]
  cluster2=[]

  for i in arr:
    if(abs(m1-i)<abs(m2-i)):
      cluster1.append(i)
    elif(abs(m1-i)==abs(m2-i)):
      if(len(cluster1)>len(cluster2)):
        cluster2.append(i)
      else:
        cluster1.append(i)
    else:
        cluster2.append(i)

  c1_mean=mean(cluster1)
  c2_mean=mean(cluster2)

  print('C1 :',c1_mean)
  print('C2 :',c2_mean)

  print('Cluster1',cluster1)
  print('Cluster2',cluster2)

  if(c1_mean==m1 and c2_mean==m2):
    print('------------------------------')
    print('final cluster are')
    print('Cluster1',cluster1)
    print('Cluster2',cluster2)
  else:
    k_mean(arr,c1_mean,c2_mean)

arr=[2,5,8,12,14,16,22,24,30,32,35]
k=2
k_mean(arr,5,14)
```

```
C1 : 5
C2 : 23
Cluster1 [2, 5, 8]
Cluster2 [12, 14, 16, 22, 24, 30, 32, 35]
C1 : 6
C2 : 24
Cluster1 [2, 5, 8, 12]
```

```
Cluster2 [14, 16, 22, 24, 30, 32, 35]
C1 : 8
C2 : 26
Cluster1 [2, 5, 8, 12, 14]
Cluster2 [16, 22, 24, 30, 32, 35]
C1 : 9
C2 : 28
Cluster1 [2, 5, 8, 12, 14, 16]
Cluster2 [22, 24, 30, 32, 35]
C1 : 9
C2 : 28
Cluster1 [2, 5, 8, 12, 14, 16]
Cluster2 [22, 24, 30, 32, 35]
------------------------------
final cluster are
Cluster1 [2, 5, 8, 12, 14, 16]
Cluster2 [22, 24, 30, 32, 35]
```

```
#binning
data = [[1,120000],
        [2,23000],
        [3,34000],
        [4,45000],
        [5,50000],
        [6,110000],
        [7,80000],
        [9,150000],
        [10,125000]]


def binning(data):

  data1=  data.copy()
  bin1 = []
  bin2 = []
  bin3 = []

  #create bins
  for i in range (len(data)):
    if ((data1[i][1] >10000) and data1[i][1] < 55001):
      bin1.append(data[i][1])
      data1[i][1] = 1

    if ((data1[i][1] >55000) and data1[i][1] < 90001):
      bin2.append(data[i][1])
      data1[i][1] = 2

    if ((data1[i][1] >90000) and data1[i][1] < 1250001):
      bin3.append(data[i][1])
      data1[i][1] = 3



  print(data1)
  print("bin1 = ", bin1)
  print("bin2 = ", bin2)
```

```
  print("bin3 = ", bin3)


binning(data)
```

```
[[1, 3], [2, 1], [3, 1], [4, 1], [5, 1], [6, 3], [7, 2], [9, 3], [10, 3]]
bin1 =  [23000, 34000, 45000, 50000]
bin2 =  [80000]
bin3 =  [120000, 110000, 150000, 125000]
```