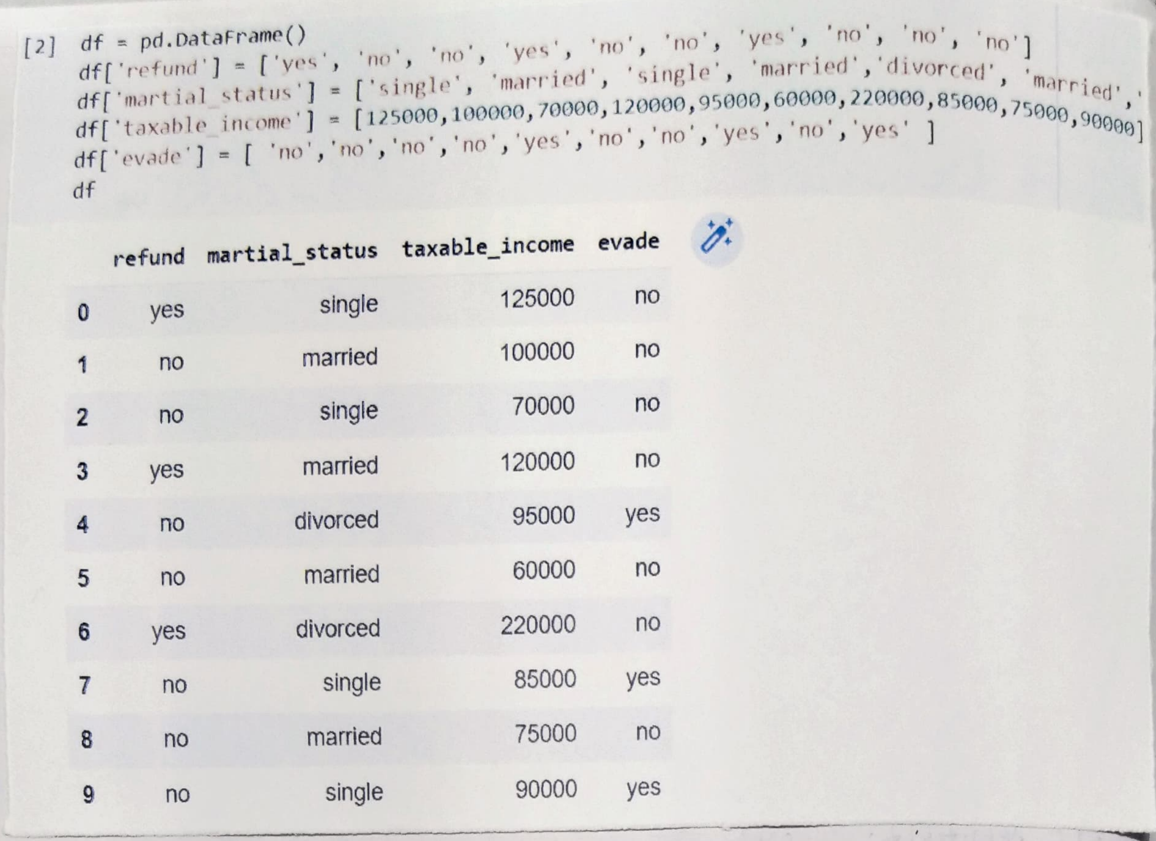
```
        elif (i=='0' and y-pred [j] =='1'):
              FN+=1

      j+=1
  Confusion_matrix = [TP, TN, FP, FN]
  print ('Confusion_matrix', Confusion_matrix)
  ACC = (TP + TN)/TP+TN+FP+FN
  print ("Accuracy: ", ACC)
  PREC = (TP) / (TP+FP)
  print ("Precision: ", PREC)
  REC = TP / (TP + FN)
  print ("Recall : ", REC)
  SN = TP / (TP+FN)
  print ("Sensitivity: ", SN)
  SP = TN/ (TN + FP)
  print ("Specificity : ", SP)
  print ("Misclassification Error: ", MCE)
  MCE = 1- ACC
```

Result:

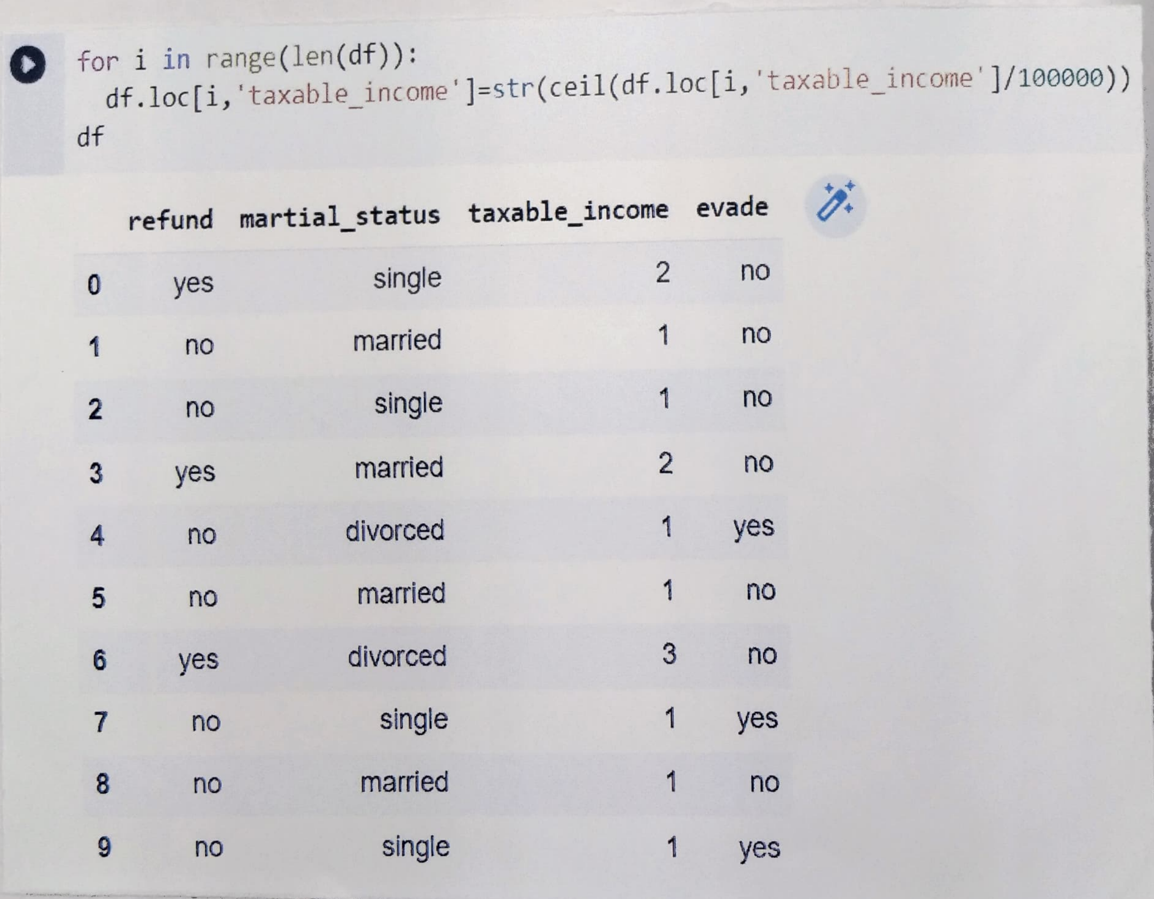
The above program evaluating the results of machine learning is executed successfully.

22/3/22



```
[2]  df = pd.DataFrame()
     df['refund'] = ['yes', 'no', 'no', 'yes', 'no', 'no', 'yes', 'no', 'no', 'no']
     df['martial_status'] = ['single', 'married', 'single', 'married','divorced', 'married','
     df['taxable_income'] = [125000,100000,70000,120000,95000,60000,220000,85000,75000,90000]
     df['evade'] = [ 'no','no','no','no','yes','no','no','yes','no','yes' ]
     df
```

```
   refund  martial_status  taxable_income  evade
0     yes          single          125000     no
1      no         married          100000     no
2      no          single           70000     no
3     yes         married          120000     no
4      no        divorced           95000    yes
5      no         married           60000     no
6     yes        divorced          220000     no
7      no          single           85000    yes
8      no         married           75000     no
9      no          single           90000    yes
```



```
for i in range(len(df)):
  df.loc[i,'taxable_income']=str(ceil(df.loc[i,'taxable_income']/100000))
df
```

```
   refund  martial_status  taxable_income  evade
0     yes          single               2     no
1      no         married               1     no
2      no          single               1     no
3     yes         married               2     no
4      no        divorced               1    yes
5      no         married               1     no
6     yes        divorced               3     no
7      no          single               1    yes
8      no         married               1     no
9      no          single               1    yes
```

Expt. No. 5

Page No. 32

Expt. Name. Implement Classification Algorithm

Date :

Aim:

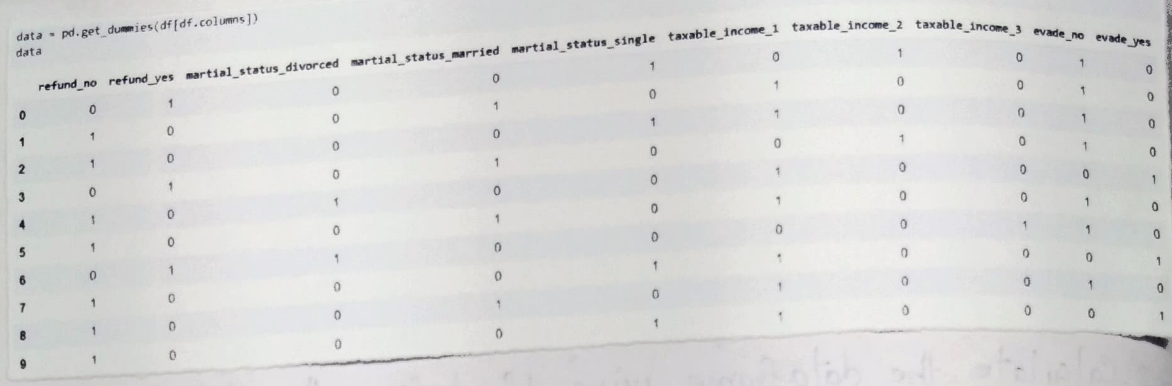
To write a python program implementing clasification algorithm

Algorithm:

```
step 1: start
step2: import the necessary packages
step 3: Calculate the dataframe using df define the dataframe
        refund, flexible income , evrode.
step 4: Using for loop find the length of df
step 5: using ceil function calculate 'flexable-income'
step6: Calculate the strna , strnb, prby [J, prbn[J)
step7: Calculate the dummy values in df
step 8: print the evade of x
step 9: print the prb+na(data)
step10: stop
```

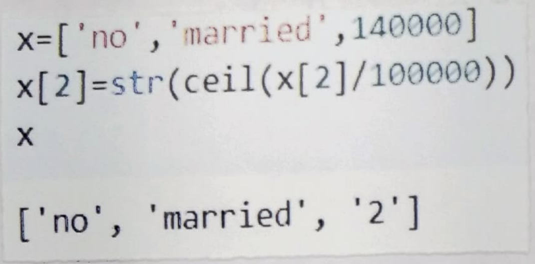
Program:

```
import pandas as pd
import numpy as np
from math import *
df = pd. Dataframe()
df ['refund'] = ['Yes', 'no', 'no', 'yes', 'no', 'no', 'Yes', 'no', 'no', 'no']
df ['martial-status'] = ['single', 'married', 'single', 'married', 'divorced',
                         'married', 'divorced', 'single', 'married', 'single]
df ['texable-income'] = [125000, 100000, 70000, 120000, 95000, 60000,
                         220000, 85000, 75000, 90000]
df ['evade'] = ['no', 'no', 'no', 'no', 'yes', 'no', 'Yes', 'no', 'Yes']
df
```



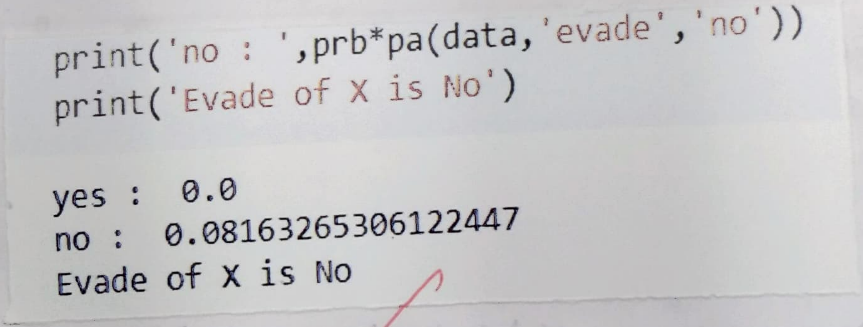
```
data = pd.get_dummies(df[df.columns])
data
```

	refund_no	refund_yes	martial_status_divorced	martial_status_married	martial_status_single	taxable_income_1	taxable_income_2	taxable_income_3	evade_no	evade_yes
0	0	1	0	0	1	0	1	0	1	0
1	1	0	0	1	0	1	0	0	1	0
2	1	0	0	0	1	1	0	0	1	0
3	0	1	0	1	0	0	1	0	1	0
4	1	0	1	0	0	1	0	0	0	1
5	1	0	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	0	1	1	0
7	1	0	0	0	1	1	0	0	0	1
8	1	0	0	1	0	1	0	0	1	0
9	1	0	0	0	1	1	0	0	0	1



```
x=['no','married',140000]
x[2]=str(ceil(x[2]/100000))
x

['no', 'married', '2']
```



```
print('no : ',prb*pa(data,'evade','no'))
print('Evade of X is No')

yes :  0.0
no :  0.08163265306122447
Evade of X is No
```

```
for i in range (len(df)):
    df.loc [i, 'taxable_income'] = str (ceil (df. loc [i, 'taxable_income'] / 100000))
df
data = pd. get_dummies (df [df. columns])
data
for i in range (1,4):
    if ('taxable_income' + str(i) not in data columns):
        data ['taxable_income' + str(i)] = [0 in range (10)]
x = ['no', 'married', 14000]
x[2] = str (ceil (x [2] /100000))
x
def pa (data, cls, sbcls):
    sbrn = str(cls) + '_' + str (sbcls)
    return (sum (data [strn]) / len (strn))
def pa_b (data, clsa, sbcls, clsb, sbc/sb):
    str na = str (clsa) + '_' + str (sbc/sa)
    str nb = str (clsb) + '_' + str (sbc/sb)
    amb = 0
    for i in range (len (data)):
        if (data. loc [i, strna] & data. loc [i, strnb]):
            anb += 1
return (amb / sum (data [strnb]))
prby = []
prbn = []
Col = df. Columns
for i in range (len(x)):
    prby. append (pa_b (data, Col[i], x[i], 'evade', 'Yes'))
for i in prby:
    prb *= 1
```

```
print("Yes: ", prb* pa (data ,'evade', 'Yes'))
for i in range (len (x)):
    prb.append (pa_b (data, Col[i], x[i] , 'evade', 'no'))
prb =1
for i in prbn:
    prb*=i
print("no:", prb* pa (data , 'evade', 'no'))
print ("Evade of x is No");
```

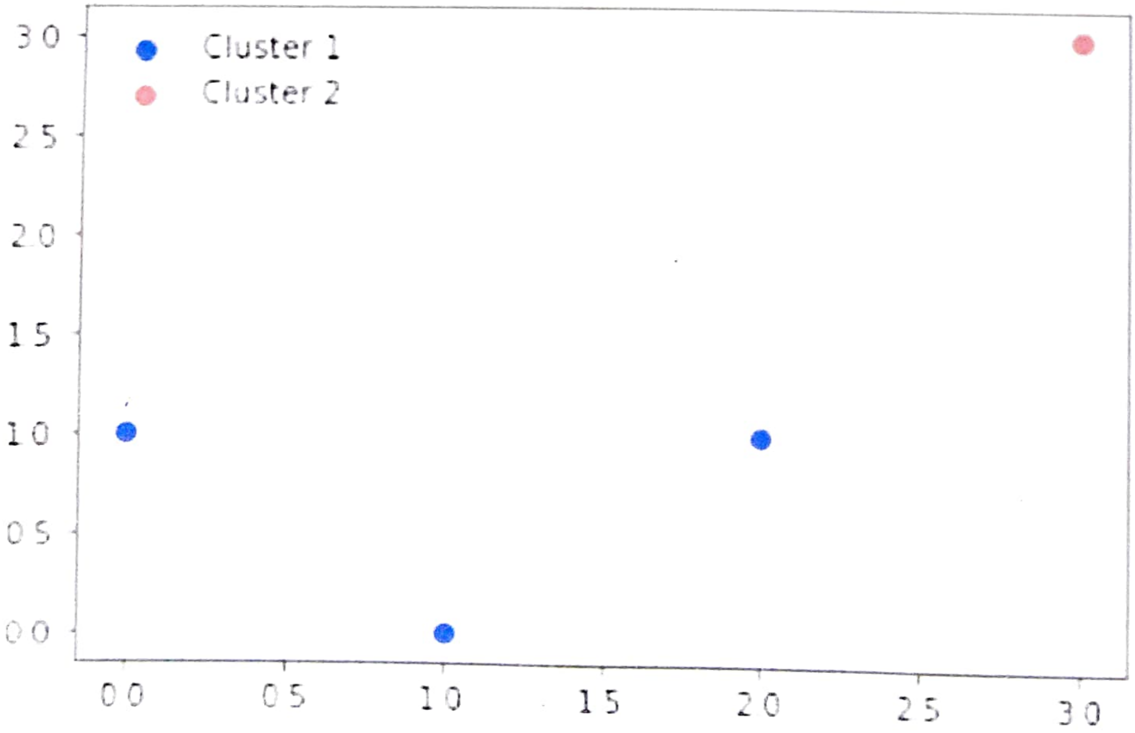
Result:

The above program of Implementing Classification Algorithm is successfully executed.

22/3/22

```
[ ]  y=[]
     for i in p1:
      x.append(i[0])
      y.append(i[1])
     plt.scatter(x,y,c='b',label='Cluster 1')
     x=[]
     y=[]
     for i in p2:
      x.append(i[0])
      y.append(i[1])
     plt.scatter(x,y,c='r',label='Cluster 2')
     plt.legend()
     plt.show()
```

```
After Epoch 1  ERROR: 7.5
After Epoch 2  ERROR: 2.6666666666666667
Data Objects:
('x1', [1, 0])
('x2', [0, 1])
('x3', [2, 1])
('x4', [3, 3])
Cluster 1: ['x1', 'x2', 'x3']
Cluster 2: ['x4']
```



Expt. No. 6

Page No. 35

Expt. Name. k-means algorithm

Date :

Aim:

To write a python program implementing k-means algorithm

Algorithm:

```
step 1: start
step2: select the number k to decide the number of cluster
step 3: select random k points or centroid
step4: Calculate the variance and place a new centroid of each cluster
step 5: Assign each data point to their closest centroid will assign k cluster
step6: Repeat step 3 ontill it nears to centroid of cluster
step7: if ony reassignment occurs, then go to step 4, goto finish
step 8: stop
```

Program:

```
import matplotlib.pyplot as plt
import numpy as np

class k_meanalg:
    def __init__(self):
        self.m1 = None
        self.m2 = None
        self.c1, self.c2, self.nc1, self.nc2 = [], [], [], []
        self.pts = {}
        self.err = 0
    def error(self):
        e1 = 0
        e2 = 0
        for i in self.c1:
            e1+=((i[0]-self.m1[0])**2+(i[1]-self.m1[1])**2)
        for i in self.c2:
            e2+=((i[0]-self.m2[0])**2+(i[1]-self.m2[1])**2)
        return (e1+e2)
    def fitting(self, c1, c2, *args):
        def mean(cl):
            n=len(cl)
            a=0
            b=0
            for i in cl:
                a+=i[0]/n
                b+=i[1]/n
            return [a,b]
        def dist(a,b):
            return ((a[0]-b[0])**2+(a[1]-b[1])**2)**(1/2)
        self.c1=c1
        self.c2=c2
        _con=1
        for i in args:
            self.pts['x'+str(_con)]=i
            _con+=1
        epo=1
        while(True):
            self.m1=mean(self.c1)
            self.m2=mean(self.c2)
            for i in args:
                p=dist(i, self.m1)
                q=dist(i, self.m2)
                if(p>q):
```

```
            self.nc2.append(i)
        else:
            self.nc1.append i)
    print ('After Epoch', epo, 'Error: ', self.Error())
    epo +=1
    if(self.c1 == self.nc1 and self.c2 == self.nc2):
        break;
    else:
      self.c1, self.c2 = self.nc1.copy(), self.nc2.copy()
      self.nc1 , self.nc2 = [], []
  for i,j in sorted(self.pts.items()):
     for k in range(len(self.c1)):
        if(j==self.c1[k]):
            self.c1 [k]=i
    for l in range(len (self.c2)):
        if (j== self.c2[l]):
            self.c2[l] = i
def display (self):
    print("Cluster 1: ", self.c1, "\n Cluster 2: ", self.c2)

x1 = [1,0]
x2 = [0,1]
x3 = [2,1]
x4 = [3,3]
c1 = [x1, x3]
c2 = [x2, x4]
model = k_meanalg()
model.fitting (c1, c2, x1, x2, x3, x4)
print("Data Objects: ")
```

```
for i in sorted(model.pts.items()):
    print(i)
model.display()

p1 = [model.pts[i] for i in model.c1]
p2 = [model.pts[i] for i in model.c2]
x = []
y = []
for i in p1:
    x.append(i[0])
    y.append(i[1])
plt.scatter(x, y, c='b', label='cluster 1')
x = []
y = []
for i in p2:
    x.append(i[0])
    y.append(i[1])
plt.scatter(x, y, c='r', label='cluster 2')
plt.legend()
plt.show()
```

Result:

The above program implementing of k-means algorithm is executed successfully.