DES

```
p8_table = [6, 3, 7, 4, 8, 5, 10, 9]
p10_table = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
p4_table = [2, 4, 3, 1]
IP = [2, 6, 3, 1, 4, 8, 5, 7]
IP_inv = [4, 1, 3, 5, 7, 2, 8, 6]
expansion = [4, 1, 2, 3, 2, 3, 4, 1]
s0 = [[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]]
s1 = [[0, 1, 2, 3], [2, 0, 1, 3], [3, 0, 1, 0], [2, 1, 0, 3]]



def checkKeyStr(key):
  one = key.count('1')
  zero = key.count('0')
  return 10 - abs(one-zero)


def permutate(original, fixed_key):
  new=''
  for i in fixed_key:
    new+=original[i-1]
  return new


def left_half(bits):
  return bits[:int(len(bits)/2)]

def right_half(bits):
  return bits[int(len(bits)/2):]


def shift(bits):
  rotate_right = right_half(bits)[1:] + right_half(bits)[0]
  rotate_left = left_half(bits)[1:] + left_half(bits)[0]
  return rotate_left + rotate_right

def key1():
  return permutate(shift(permutate(KEY,p10_table)), p8_table)

def key2():
  return permutate(shift(shift(shift(permutate(KEY,p10_table)))), p8_table)


def lookup_in_sbox(bits, sbox):
  row = int(bits[0]+bits[3],2)
  col = int(bits[1]+bits[2],2)
  return '{0:02b}'.format(sbox[row][col])

def xor(bits, key):
  new = ''
  for bit, key_bit in zip(bits, key):
    new += str(((int(bit)+int(key_bit))%2))
  return new
```

```
def f_k(bits, key):
  L = left_half(bits)
  R = right_half(bits)
  bits = permutate(R, expansion)
  bits = xor(bits, key)
  bits = lookup_in_sbox(right_half(bits),s0) + lookup_in_sbox(left_half(bits),s1)
  bits = permutate(bits, p4_table)
  return xor(bits, L)

def encrypt(plaintext):
  bits = permutate(plaintext, IP)
  temp = f_k(bits, key1())
  bits = right_half(bits)+temp
  bits = f_k(bits, key2())
  return permutate(bits+temp, IP_inv)

def decrypt(ciphertext):
  bits = permutate(ciphertext, IP)
  temp = f_k(bits, key2())
  bits = right_half(bits)+temp
  bits = f_k(bits, key1())
  return permutate(bits+temp, IP_inv)


KEY = '0101101101'
print("str :" , checkKeyStr(KEY))
plaintext = '11101010'
cipher = encrypt(plaintext)
print(cipher)
pl = decrypt(cipher)
print(pl)
```

```
      str : 8
      10101111
      11101010
```

RSA

```
from random import randint
from math import gcd


p=53
q=59
n = p * q
fin = (p-1) * (q-1)
e = 3


while True:
  e = randint(1, fin)
  if gcd(e, fin) == 1 :
   break
```

```
for i in range(1, fin):
  if (i*fin +1) % e == 0 :
    d = (i*fin+1)//e
    break




def encrypt(plain):
  return (plain ** e)%n

def decrypt(cipher):
  return (cipher ** d)%n


c = encrypt(98)
m = decrypt(c)
print(c, m)

      2430 98
```

Diffie

```
p = 59
G = 0
sa = 3
sb = 4


def fpow(a, b, m):
  if b == 0:
    return 1
  r= fpow(a, b//2,m)
  r = (r*r)%m
  if b%2 == 1:
    r= (r*a)%m
  return r

for r in range(1, p,1):
  s=set()
  for x in range(p-1):
    s.add(fpow(r,x,p))
  if len(s) == p-1:
    G=r
    break

print(G)

pa = (G**sa) % p
pb = (G**sb) % p

ka = (pb**sa) % p
```

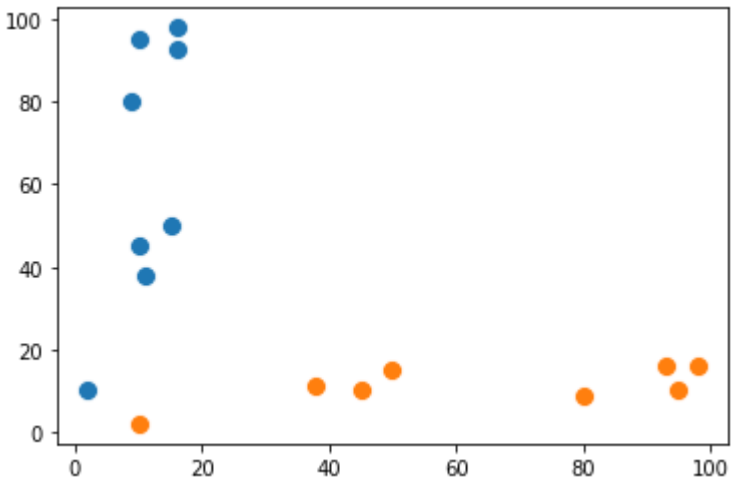
```
kb = (pa**sb) % p

if ka == kb :
  print(True)
else:
  print(False)

print(ka, kb, pa, pb)
```

```
2
True
25 25 8 16
```

linear reg

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


data = [
        (10, 95),
        (9, 80),
        (2, 10),
        (15, 50),
        (10, 45),
        (16, 98),
        (11, 38),
        (16, 93),
]


x = [pt[0] for pt in data]
y = [pt[1] for pt in data]


sns.scatterplot(x=x, y=y, s=100)
sns.scatterplot(x=y, y=x, s=100)
```

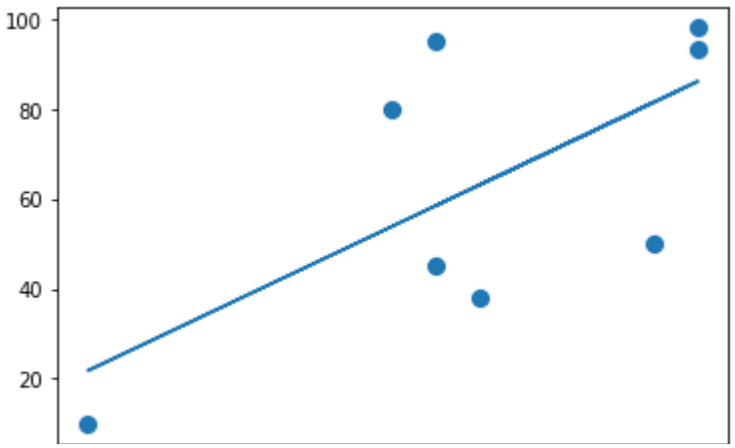
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb707419310>
```



```
n=len(x)
xx=[a*a for a in x]
xy=[x[i]*y[i] for i in range(n)]


sum_x = np.sum(x)
sum_y = np.sum(y)
sum_xx = np.sum(xx)
sum_xy = np.sum(xy)


m = (n*sum_xy- sum_x*sum_y)/(sum_xx*n-sum_x * sum_x)


b = (sum_y - m*sum_x)/n

print(f'LINE EQUATION: y = {round(m,2)} * x + {round(b,2)}')
```

```
LINE EQUATION: y = 4.59 * x + 12.58
```

```
def plot_graph(x, y, intercept, slope):
  axes = sns.scatterplot(x=x,y=y,s=100)
  x_value = np.array(x)
  y_value = intercept + slope*x_value
  plt.plot(x_value, y_value)


plot_graph(x,y,b,m)
```



decision tree

```
import numpy as np
import pandas as pd


dataset = [
```

```
['<21', 'High', 'Male', 'Single', 'No'],
['<21', 'High', 'Male', 'Married', 'No'],
['21-35', 'High', 'Male', 'Single', 'Yes'],
['>35', 'Medium', 'Male', 'Single', 'Yes'],
['>35', 'Low', 'Female', 'Single', 'Yes'],
['>35', 'Low', 'Female', 'Married', 'No'],
['21-35', 'Low', 'Female', 'Married', 'Yes'],
['<21', 'Medium', 'Male', 'Single', 'No'],
['<21', 'Low', 'Female', 'Married', 'Yes'],
['>35', 'Medium', 'Female', 'Single', 'Yes'],
['<21', 'Medium', 'Female', 'Married', 'Yes'],
['21-35', 'Medium', 'Male', 'Married', 'Yes'],
['21-35', 'High', 'Female', 'Single', 'Yes'],
['>35', 'Medium', 'Male', 'Married', 'No']
]
```

```
colums = ['Age', 'Income', 'Gender', 'Marital Status', 'Buys']
df = pd.DataFrame(dataset, columns=colums)
df
```

	Age	Income	Gender	Marital Status	Buys
0	<21	High	Male	Single	No
1	<21	High	Male	Married	No
2	21-35	High	Male	Single	Yes
3	>35	Medium	Male	Single	Yes
4	>35	Low	Female	Single	Yes
5	>35	Low	Female	Married	No
6	21-35	Low	Female	Married	Yes
7	<21	Medium	Male	Single	No
8	<21	Low	Female	Married	Yes
9	>35	Medium	Female	Single	Yes
10	<21	Medium	Female	Married	Yes
11	21-35	Medium	Male	Married	Yes
12	21-35	High	Female	Single	Yes
13	>35	Medium	Male	Married	No

```
test_data = [[0,0,0,0]]
test = pd.DataFrame(test_data, columns=['Age', 'Income', 'Gender', 'Marital Status'])
test
```

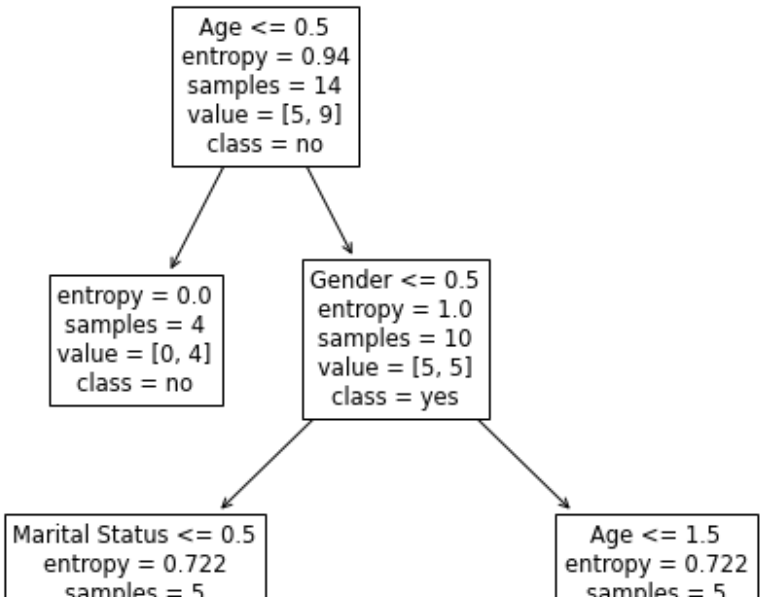
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for i in range(5):
  df[colums[i]] = le.fit_transform(df[colums[i]])

df
```

	Age	Income	Gender	Marital Status	Buys
0	1	0	1	1	0
1	1	0	1	0	0
2	0	0	1	1	1
3	2	2	1	1	1
4	2	1	0	1	1
5	2	1	0	0	0
6	0	1	0	0	1
7	1	2	1	1	0
8	1	1	0	0	1
9	2	2	0	1	1
10	1	2	0	0	1
11	0	2	1	0	1
12	0	0	0	1	1
13	2	2	1	0	0

```
dummy = df.copy()
dummy.drop('Buys', axis=1,inplace=True )
X=dummy


from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree


dtree = DecisionTreeClassifier(criterion = 'entropy')
dtree.fit(X,df['Buys'])
dtree.predict(test)

      array([1])


import matplotlib.pyplot as plt
plt.figure(figsize=[12,12])
```

```
d_tree = plot_tree(decision_tree= dtree, feature_names=df.columns,class_names=["yes","no"]
plt.plot()
```

```
[]
```



kmeans clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
```

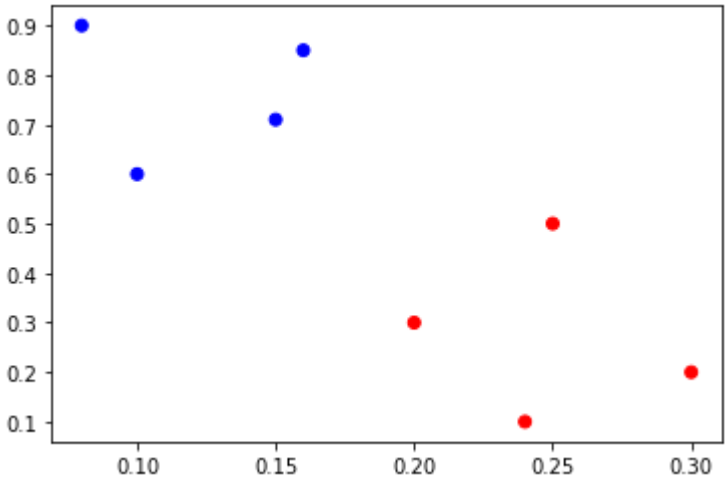
```
P1=[0.1,0.6]
P2=[0.15,0.71]
```

```
P3=[0.08,0.9]
P4=[0.16, 0.85]
P5=[0.2,0.3]
P6=[0.25,0.5]
P7=[0.24,0.1]
P8=[0.3,0.2]
K=2
points=[P1,P2,P3,P4,P5,P6,P7,P8]



def dist(p1, p2):
  xd = (p1[0]-p2[0])**2
  yd = (p1[1]-p2[1])**2
  r =  math.sqrt(xd+yd)
  return r


def cluster(C1,C2):
 cluster1 = list()
 cluster2 = list()
 c1=C1
 c2=C2

for i in points:
   d1 = dist(i,c1)
   d2= dist(i,c2)
   if d1 < d2:
     cluster1.append(i)
   else:
     cluster2.append(i)

 x1=0
 y1=0
for i in cluster1:
   x1 +=i[0]
   y1 +=i[1]

 x1 = x1/(len(cluster1))
 y1 = y1/(len(cluster1))

 x2=0
 y2=0
for i in cluster2:
   x2 +=i[0]
   y2 +=i[1]

 x2 = x2/(len(cluster1))
 y2 = y2/(len(cluster1))

 centroid1 = [x1,y1]
 centroid2 = [x2,y2]

if centroid1[0] == c1[0] and centroid1[1] == c1[1] and centroid2[0] == c2[0] and centroid2

   print(c1,c2)
   p=list()
   for i in points:
     if i in cluster1:
       p.append(0)
     else:
```

```
        p.append(1)

    X = np.array(points)
    colours = list(map(lambda x:'blue' if x==1 else 'red',p))
    plt.scatter(X[:,0], X[:,1], c=colours, marker ='o', picker=True)
    plt.show()

else:
    cluster(centroid1,centroid2)
```

```
c1=P1
c2=P2
cluster(c1,c2)
print(points)
```

```
[0.2475, 0.275] [0.1225, 0.765]
```




```
[[0.1, 0.6], [0.15, 0.71], [0.08, 0.9], [0.16, 0.85], [0.2, 0.3], [0.25, 0.5],
```

KNN

```
import pandas as pd
import numpy as np


X=[[2,0],[4,0],[4,1],[4,0],[6,1],[6,0]]
y=[[4,0],[2,0],[4,1],[6,0],[2,1],[4,0]]


from sklearn.neighbors import KNeighborsClassifier
cl = KNeighborsClassifier(n_neighbors=3)
cl.fit(X,y)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
x_test = np.array([6,6])
y_pred=cl.predict([x_test])
```

```
print('kn',y_pred)
```

```
      kn [[4 1]]
```

```
cl = KNeighborsClassifier(n_neighbors=3,weights='distance')
cl.fit(X,y)
```

```
      KNeighborsClassifier(n_neighbors=3, weights='distance')
```

```
x_test = np.array([6,6])
y_pred=cl.predict([x_test])
print('knw',y_pred)
```

```
      knw [[4 1]]
```