dataset generation (30 row , 20 columns , 0 or 1)

In [33]:

```
import numpy as np

np.random.seed(42)  # Set the random seed for reproducibility

num_rows = 30
num_cols = 20

random_array = np.random.randint(2, size=(num_rows, num_cols))

print(random_array)
```

```
[[0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0]
 [1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 0]
 [0 0 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 0 0 0]
 [0 0 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1 1 0 1]
 [0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0]
 [1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1]
 [0 0 1 1 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 0]
 [0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 0]
 [0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0]
 [0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1]
 [1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0]
 [0 1 0 0 0 1 1 1 0 0 1 1 1 1 0 1 0 1 0 1]
 [1 1 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1]
 [0 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 1]
 [1 0 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1]
 [0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 0 0]
 [1 0 0 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0]
 [1 1 1 0 0 1 0 0 0 0 0 0 1 1 0 1 1 1 1 0]
 [1 1 0 0 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1]
 [1 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0]
 [0 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0]
 [1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1]
 [1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 1]
 [0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1]
 [1 0 1 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 1]
 [0 1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 1 1 1 1]
 [1 0 0 0 1 1 1 1 0 0 1 0 0 1 0 1 0 1 1 1]
 [1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 0 0 1 1]]
```

In [34]:

```
import pandas as pd
df = pd.DataFrame(data=random_array)
df
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	1	1	0
1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	0	1	0	0	0
2	0	0	1	1	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	0
3	0	0	0	0	0	1	1	0	1	1	1	1	0	1	0	1	1	1	0	1
4	0	1	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0
5	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0	1	1	0
6	1	0	1	0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
7	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1
8	0	0	1	1	1	0	1	0	0	1	1	0	0	1	1	1	0	0	0	0
9	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0
10	0	1	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	1	0
11	0	0	1	1	1	1	0	1	0	0	1	1	1	1	1	1	1	1	0	1
12	1	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	1	0
13	0	1	0	0	0	1	1	1	0	0	1	1	1	1	0	1	0	1	0	1
14	1	1	1	0	1	0	0	0	0	1	0	0	0	1	1	1	1	0	0	1
15	0	0	0	1	1	0	1	1	1	1	1	0	1	0	0	1	0	0	0	1
16	1	0	1	1	1	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1
17	0	1	0	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1	0	0
18	1	0	0	1	0	0	1	1	0	0	1	1	1	1	0	1	1	0	1	0
19	1	1	1	0	0	1	0	0	0	0	0	0	1	1	0	1	1	1	1	0
20	1	1	0	0	1	1	1	1	1	1	0	1	1	0	0	0	0	1	1	1
21	1	1	1	0	1	0	0	1	0	1	0	1	0	1	1	1	1	1	0	0
22	0	1	0	1	1	0	0	1	0	1	1	1	1	1	0	0	1	1	0	0
23	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1
24	1	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1
25	0	0	0	1	0	0	1	0	1	1	0	1	0	1	1	0	1	1	1	1
26	1	0	1	0	1	0	0	1	1	1	0	1	0	0	0	1	1	0	0	1
27	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	1	1	1	1	1
28	1	0	0	0	1	1	1	1	0	0	1	0	0	1	0	1	0	1	1	1
29	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	1	0	0	1	1

sort the dataset in ascending order according to fitness function

In [35]:

```
generation = 0

while True:
    # ith generation
    generation = generation + 1

    # defining fitness function
    # here fitness function takes binary value and gives corresponding integer value
    p = 19
    sum = 0
    fitness_list = list()
    for i in range(df.shape[0]):
        for j in range(df.shape[1]):
            sum = sum + df.iloc[i,j] * (2 ** p)
            p = p - 1
        fitness_list.append(sum)
        sum = 0
        p = 19

    # adding fitness to df
    df['fitness'] = fitness_list

    # sorting df in ascending order according to fitness
    df = df.sort_values(by='fitness', ascending=True)

    # printing chromosome with lowest fitness and it's efficiency
    print('Generation No :',generation)
    print('Chromose with lowest fitness :',list(df.iloc[0,:-1]))
```

```
    print('Lowest fitness :',df['fitness'][0])
    print('Efficiency :',1/df['fitness'][0])
    print()

    # taking first half of df
    length = int(len(df) / 2)

    # breaking condition
    if length == 1:
        break

    df = df[:length][:]

    # resetting index and removing index column
    df = df.reset_index()
    df = df.drop(['fitness','index'],axis=1)

    # reproduction through corss-over and mutation
    new_child = list()
    for i in range(1):
        list1 = list(df.iloc[i])
        for j in range(1,len(df)):
            list2 = list(df.iloc[j])
            l = list1[:5] + list2[5:]
            l[10] = 1 - l[10]
            new_child.append(l)

    # creating new generation with children produced from previous generation
    df = pd.DataFrame(data=new_child)
```

```
Generation No : 1
Chromose with lowest fitness : [0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1
]
Lowest fitness : 279598
Efficiency : 3.576563494731722e-06

Generation No : 2
Chromose with lowest fitness : [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0
]
Lowest fitness : 12143
Efficiency : 8.23519723297373e-05

Generation No : 3
Chromose with lowest fitness : [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0
]
Lowest fitness : 6764
Efficiency : 0.00014784151389710232

Generation No : 4
Chromose with lowest fitness : [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0
]
Lowest fitness : 9756
Efficiency : 0.0001025010250102501
```