1. GENETIC ALGORITHM

x^2 minimization function

objective = x^2 + y^2 x belongs to [-10,10, y also to [-10, 10] has an optima at f(0, 0) = 0.0

```python
def objective(x):
    return x[0]**2 + x[1]**2
```

```python
bounds = [[-10.0, 10.0], [-10.0, 10.0]]
```

```python
from numpy.random import randint
from numpy.random import rand
```

```python
def decode(bounds, n_bits, bitstrings):
    decoded = list()
    largest = 2**n_bits

    for i in range(len(bounds)):

        start, end = i*n_bits, (i*n_bits)+n_bits
        substring = bitstrings[start: end]
        chars = ''.join([str(s) for s in substring])

        intval = int(chars, 2)

        value = bounds[i][0] + (intval/largest) *  (bounds[i][1] - bounds[i][0])
        decoded.append(value)

    return decoded
```

+ Code + Markdown

```python
def selection(pop, scores, k=3):
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k-1):
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix

    return pop[selection_ix]
```

```python
def crossover(p1, p2, r_cross):
    c1, c2 = p1.copy(), p2.copy()

    if rand() < r_cross:
        pt = randint(1, len(p1)-2)

        c1 = p1[:pt]+p2[pt:]
        c2 = p2[:pt]+p1[pt:]


    return [c1, c2]
```

```python
def mutation(bs, r_mut):
    for i in range(len(bs)):
        if rand() < r_mut:
            bs[i] = 1-bs[i]
```

```python
def gen_alg(obj, bounds, n_bits, n_iter, n_pop, r_cross, r_mut):
    pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
    best, best_eval = 0, obj(decode(bounds, n_bits, pop[0]))

    for gen in range(n_iter):
        decoded = [decode(bounds, n_bits, p) for p in pop]

        scores = [obj(d) for d in decoded]

        for i in range(n_pop):
            if scores[i] < best_eval:
                best, best_eval = pop[i], scores[i]
                print(">%d, new best f(%s) = %f" % (gen,  decoded[i], scores[i]))

        selected = [selection(pop, scores) for _ in range(n_pop)]

        children = list()
        for i in range(0, n_pop, 2):
            p1, p2 = selected[i], selected[i+1]

            for c in crossover(p1, p2, r_cross):
                mutation (c, r_mut)
                children.append(c)


        pop = children

    return [best, best_eval]
```

Python

```python
n_iter = 100
n_bits = 16
n_pop = 100
r_cross = 0.7
r_mut = 1.0/ (float (n_bits) * len(bounds))

best, score = gen_alg(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut)

print("\nDone!!!")

decoded = decode(bounds, n_bits, best)

print('f(%s) = %f' % (decoded, score))
```

Python

```
>0, new best f([0.64666748046875, -6.6241455078125]) = 44.297483
>0, new best f([-3.38592529296875, 3.5015869140625]) = 23.725601
>0, new best f([0.63140869140625, 3.380126953125]) = 11.823935
>0, new best f([-2.40264892578125, 0.8447265625]) = 6.486285
>0, new best f([-1.62445068359375, -1.7718505859375]) = 5.778295
>0, new best f([-0.965576171875, -0.22857666015625]) = 0.984585
>0, new best f([-0.614013671875, 0.36468505859375]) = 0.510008
>1, new best f([-0.611572265625, 0.36468505859375]) = 0.507016
>4, new best f([0.400390625, 0.504150390625]) = 0.414480
>4, new best f([0.440673828125, 0.10406494140625]) = 0.205023
>5, new best f([0.328369140625, 0.10284423828125]) = 0.118403
>6, new best f([0.050048828125, 0.101318359375]) = 0.012770
>7, new best f([0.050048828125, 0.084228515625]) = 0.009599
>8, new best f([0.048828125, 0.084228515625]) = 0.009479
>9, new best f([0.006103515625, 0.079345703125]) = 0.006333
>9, new best f([0.006103515625, 0.02593994140625]) = 0.000710
>10, new best f([0.006103515625, 0.00518798828125]) = 0.000064
>11, new best f([0.006103515625, 0.001220703125]) = 0.000039
>14, new best f([0.0, 0.00396728515625]) = 0.000016
>15, new best f([0.0, 0.00152587890625]) = 0.000002
>15, new best f([0.0, 0.001220703125]) = 0.000001
>32, new best f([0.00030517578125, 0.0006103515625]) = 0.000000
>33, new best f([0.0, 0.0006103515625]) = 0.000000
>38, new best f([0.00030517578125, 0.0]) = 0.000000
>40, new best f([0.0, 0.0]) = 0.000000

Done!!!
f([0.0, 0.0]) = 0.000000
```

2. K MEANS

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_blobs

x,y = make_blobs(n_samples = 1000, centers = 6, random_state= 110)
```

Python

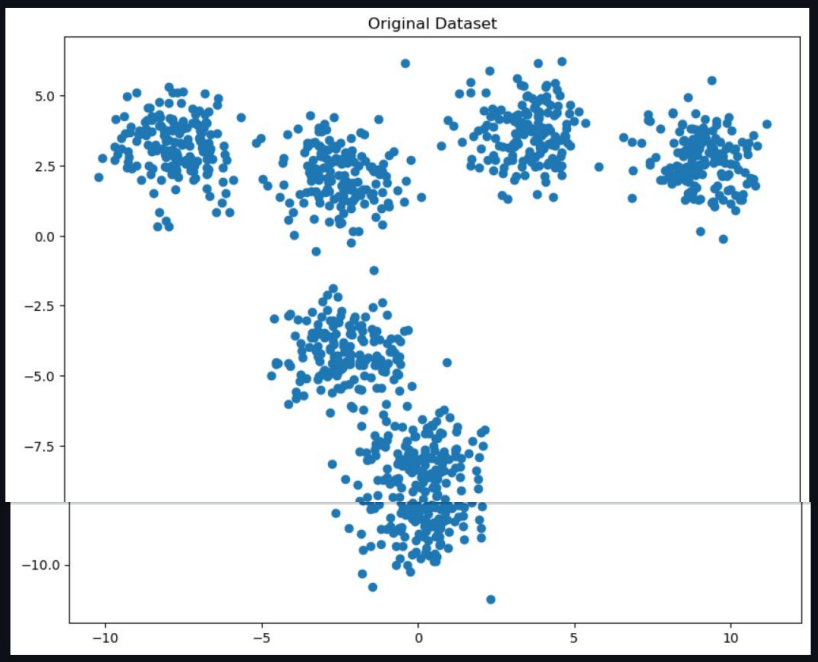
```python
m = x.shape[0]
n = x.shape[1]

n_iters = 100
```

Python

```python
plt.scatter(x[:,0],x[:,1])
plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})
plt.title('Original Dataset')
```

Python

```
Text(0.5, 1.0, 'Original Dataset')
```




```python
K = 6

import random


centroids = np.array([]).reshape(n,0)
```

Python

+ Code + Markdown

```python
for k in range(K):
    centroids = np.c_[centroids, x[random.randint(0, m-1)]]
```

Python

```python
centroids
```

Python

```
array([[ -2.32983715,   7.90976104,   0.81379922,  -7.69959809,
         -1.46056884,  -0.28878026],
       [  1.79572622,   2.00598192,  -7.03300272,   3.75881288,
        -10.78443288,   1.95510437]])
```

```python
output = {}

euclid = np.array([]).reshape(m, 0)

for k in range(K):
    dist = np.sum((x-centroids[:, k])**2, axis = 1)
    euclid = np.c_[euclid, dist]
```

Python

```python
minimum=np.argmin(euclid,axis=1)+1
```

Python

```python
# computing the mean of separated clusters
cent={}
for k in range(K):
    cent[k+1]=np.array([]).reshape(2,0)

# assigning of clusters to points
for k in range(m):
    cent[minimum[k]]=np.c_[cent[minimum[k]],x[k]]
for k in range(K):
    cent[k+1]=cent[k+1].T

# computing mean and updating it
for k in range(K):
    centroids[:,k]=np.mean(cent[k+1],axis=0)
```

Python

```python
# repeating the above steps again and again
for i in range(n_iters):
    euclid=np.array([]).reshape(m,0)
    for k in range(K):
        dist=np.sum((x-centroids[:,k])**2,axis=1)
        euclid=np.c_[euclid,dist]
    C=np.argmin(euclid,axis=1)+1
    cent={}
    for k in range(K):
        cent[k+1]=np.array([]).reshape(2,0)
    for k in range(m):
        cent[C[k]]=np.c_[cent[C[k]],x[k]]
    for k in range(K):
        cent[k+1]=cent[k+1].T
    for k in range(K):
        centroids[:,k]=np.mean(cent[k+1],axis=0)
    final=cent
```

Python

```python
for k in range(K):
    plt.scatter(final[k+1][:,0],final[k+1][:,1])
plt.scatter(centroids[0,:],centroids[1,:],s=300,c='yellow')
plt.rcParams.update({'figure.figsize':(10,7.5), 'figure.dpi':100})
plt.show()
```

Python