```
# Importing the required libraries
import numpy as np
import pandas as pd
import math
from random import randint
import random


# Sudoku representation as a string of numbers. A candidate.
my_num = 
'530070000600195000098000060800060003400803001700020006060000280000419005000080079'


# Function converts the string of numbers into 9x9 matrix
def sep_list(string):
    my_list =[]
    for i in string:
        my_list.append(int(i))
    return np.array(my_list).reshape(9,9)


# Initializing the sudoku
sudoku = sep_list(my_num)
# sudoku


# Storing the non-zero indices in the sudoku for later purposes
non_zero_indi = list()
new_sudoku = sudoku.flatten()
for i in range(sudoku.size):
    if new_sudoku[i]!=0:
        non_zero_indi.append(i)


# length of zero positions in sudoku. this will be the length of our genome.
len_of_genome = sudoku.size - np.count_nonzero(sudoku)


# Generate population for the sudoku
def population(integer):
    pop = list()
    for i in range(integer):
        pop.append(np.random.randint(1,9,size=len_of_genome))
    return pop


# converting the population into dictionary
def dict_pop1(pop):
    dict_pop = {}
    for i in range(len(pop)):
        dict_pop['pop{}'.format(i)] = pop[i]
    return dict_pop


# generate the genome
def genome(sudoku,pop):

    all_sudoku = list()

    for p in range(len(pop)):
        new_sudoku = sudoku.flatten()
        j=0
        for i in range(new_sudoku.size):
            if new_sudoku[i] == 0:
                new_sudoku[i] = pop[p][j]
                j+=1
        all_sudoku.append(new_sudoku)

    return all_sudoku


# fitness calculation
def fitness_calc(genes):

    all_fitness = {}
    reshaped_sudoku = list()

    for gene_no,gene in enumerate(genes):
        fitness = 0
        r1 = 0

        temp = gene.reshape(9,9)

        for i in range(9):
            fitness+=len(np.unique(temp[:,i]))
            fitness+=len(np.unique(temp[i,:]))

        for k in range(0,3):
            c1 = 0
            for q in range(0,3):
                temp1 = temp[0 + r1:3 + r1, 0 + c1:3 + c1].flatten()
                fitness+=len(np.unique(temp1))
                c1+=3
            r1+=3
        all_fitness['pop{}'.format(gene_no)]= fitness/float(243)

    return all_fitness



# eliminating the genome with minumum fitness values
def elimination(fitn_l,p,n):

    for i in range(0,int(p*n)):
        min_val = min(fitn_l.values())
        fitn_l = {key: value for key, value in fitn_l.items()
             if value is not min_val}

    return fitn_l



# calculating the fitness proportion
def crossover(fit_list):

    cross_sum = 0

    for i in fit_list.values():
        cross_sum+=i
    fit_prop_list = {}

    for key,value in fit_list.items():
        fit_prop_list[key] = value/cross_sum
    return fit_prop_list



# for roulette wheel
def rolette(cross):

    roulette_dic = {}
    roulette_sum = 0

    for key, value in cross.items():
        roulette_dic[key]= value+roulette_sum
        roulette_sum+=value

    return roulette_dic


# to selelct the parents for crossover
def comparison(r, rolette2):

    total = 0

    for key, values in rolette2.items():
        if r <= values:
            return key



# generating the children using crossover and mutation
def child(rolette1,dict_pop1,n,p):

    child_list = list()
    index = 0

    for length in range(0,int(n*p)):
        gene1 = list()
        gene2 = list()

        r1 = random.uniform(0,1)
        r2 = random.uniform(0,1)

        key1 = comparison(r1,rolette1)
        key2 = comparison(r2,rolette1)

        gene1 = dict_pop1[key1]
        gene2 = dict_pop1[key2]

        r3 = random.randint(int(len(gene1)/3),len(gene1)-10)

        gene1[0:r3],gene2[0:r3] = gene2[0:r3],gene1[0:r3]

        for i in range(0,5):
            r_mut1 = random.randint(0,int(len(gene1)/2)-1)
            r_mut2 = random.randint(int(len(gene1)/2),len(gene1)-1)
            temp = gene1[r_mut1]
            gene1[r_mut1] =gene1[r_mut2]
            gene1[r_mut2] =temp
        child_list.append(gene1)

    return child_list
```

```
# generating the population. Initialization
pop = population(10000)



# running the GA for 20 generations
for i in range(0,20):
    dict_pop = dict_pop1(pop)
    new_su = genome(sudoku,pop)

    fitn_list = fitness_calc(new_su)
    elimi_list = elimination(fitn_list,10000,0.3)

#     print('sum' + '=' + str(sum(elimi_list.values())/7000))
#     print('maximum' + '=' +  str(max(elimi_list.values())))

    cross_fit = crossover(elimi_list)
    rolette_result = rolette(cross_fit)
    chil1 =child(rolette_result,dict_pop,10000,0.3)
#     print(chil1)
    k_l = list()

    for keys in elimi_list.keys():
        k_l.append(int(keys[3:]))
    n_l = []
    n_l = range(int(max(k_l))+1)
    v = list(set(n_l)-set(k_l))

    for i,x in zip(v,chil1):
        pop[i] = x
```