Part 1 -

```
import random
import string


#population
def gen_pop(Len, popLen):
    pop = []
    for _ in range(popLen):
        pop.append(''.join(random.choices(string.ascii_lowercase,k=Len)))
    return pop

def fitness(Input,Word):
    score=0
    for i in range(len(Input)):
        if(Input[i]==Word[i]):
            score+=1
    return score/len(Input)
def eval(Input,Pop):
    fitPop = {}
    for i in Pop:
        fitPop[i] = fitness(Input, i)
    return fitPop
def BPop(FitPop):
    population=list(FitPop.keys())
    fit=list(FitPop.values())
    members=random.choices(population,fit,k=20)
    Bpop={}
    for i in members:
        Bpop[i]=FitPop[i]
    return Bpop

def create_new_population(breeding_population):
    new_population = []
    BPMem=list(breeding_population.keys())
    for i in range(0, len(breeding_population), 2):
        parent1 = BPMem[i]
        parent2 = BPMem[i+1 if i+1 < len(breeding_population) else 0]
        child1, child2 = '', ''
        for j in range(len(parent1)):
            if random.random() < 0.5:
                child1 += parent1[j]
                child2 += parent2[j]
            else:
                child1 += parent2[j]
                child2 += parent1[j]
        print(child1)
        print(child2)
        new_population.append(child1)
        new_population.append(child2)
    return new_population

def StopC(NPop,threshold=0.6):
    MaxS=max(list(NPop.values()))
    if MaxS>=threshold:
        return True

def GenAlgo():
    Input=input("Enter Word : ")

    Population=gen_pop(len(Input),100)
    Pop=Population
    generation=0
    while True:
        generation+=1
        Pop=eval(Input,Pop)

        if StopC(Pop,threshold=0.6):
            members=list(Pop.keys())
            fit=list(Pop.values())
            print(f"Word guessed or threshold met in generation {generation}")
            best_match = members[fit.index(max(fit))]
            print(f"Best matching word: {best_match} with fitness {max(fit)}")
            break
```

```
        breeding_population = BPop(Pop)

        Pop = create_new_population(breeding_population)


GenAlgo()
```

```
Enter Word : pune
pinw
divs
pdfc
bxue
pdij
pfmz
rtee
ekle
uwze
supk
yxaz
psnp
ekme
pflz
pdnc
pifw
rtee
bxue
pspp
sunk
pwzj
udie
pflz
sunk
edme
ukie
rdee
ptnc
bxfw
piue
psnp
supk
sunk
supk
piee
rdue
ukme
edie
punk
ssnp
Word guessed or threshold met in generation 5
Best matching word: punk with fitness 0.75
```

Part - 2

```
!pip install deap
```

```
Collecting deap
  Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from deap) (1.26.4)
Downloading deap-1.4.1-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (135 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.4/135.4 kB 2.8 MB/s eta 0:00:00
Installing collected packages: deap
Successfully installed deap-1.4.1
```

```
import random
import numpy as np
from sklearn import datasets
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from deap import base, creator, tools, algorithms

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Define the fitness function
def svm_fitness(individual):
    C = max(individual[0], 0.0001)  # Ensure C is positive
```

```
        C = max(individual[0], 0.0001)  # Ensure C is positive
        gamma = max(individual[1], 0.0001)  # Ensure gamma is positive
        # Define the model with current hyperparameters
        model = SVC(C=C, gamma=gamma)
        # Perform cross-validation
        accuracy = cross_val_score(model, X, y, cv=5).mean()
        return accuracy,

    # Create the fitness function and individual (chromosome) structure
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    # Define the parameter space (e.g., C and gamma)
    toolbox = base.Toolbox()
    toolbox.register("attr_float", random.uniform, 0.0, 100)  # C: 0.1 to 100
    toolbox.register("attr_float2", random.uniform, 0.0001, 1)  # gamma: 0.0001 to 1

    # Structure of an individual
    toolbox.register("individual", tools.initCycle, creator.Individual,
                     (toolbox.attr_float, toolbox.attr_float2), n=1)
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    # Register genetic algorithm operations
    toolbox.register("mate", tools.cxBlend, alpha=0.5)
    toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", svm_fitness)

    # Run the genetic algorithm
    def main():
        pop = toolbox.population(n=20)  # Population size
        hof = tools.HallOfFame(1)       # Hall of Fame to store the best individual
        stats = tools.Statistics(lambda ind: ind.fitness.values)
        stats.register("avg", np.mean)
        stats.register("max", np.max)

        # Use the algorithm.eaSimple method for evolution
        algorithms.eaSimple(pop, toolbox, cxpb=0.7, mutpb=0.2, ngen=20,  # 20 generations
                            stats=stats, halloffame=hof, verbose=True)

        print("Best individual: ", hof[0])
        print("Best fitness: ", hof[0].fitness.values[0])

    if __name__ == "__main__":
        main()
```

```
gen     nevals  avg             max
0       20      0.960667        0.986667
1       13      0.962333        0.986667
2       15      0.967333        0.986667
3       18      0.983333        0.986667
4       18      0.981           0.986667
5       12      0.982667        0.986667
6       19      0.986667        0.986667
7       17      0.986667        0.986667
8       16      0.986667        0.986667
9       16      0.986667        0.986667
10      19      0.982667        0.986667
11      14      0.982667        0.986667
12      18      0.986333        0.986667
13      17      0.986667        0.986667
14      16      0.979667        0.986667
15      17      0.986667        0.986667
16      14      0.986667        0.986667
17      15      0.986667        0.986667
18      13      0.986667        0.986667
19      14      0.986667        0.986667
20      13      0.981           0.986667
Best individual:  [20.355441047761357, 0.01982223039733335]
Best fitness:  0.9866666666666667
```