Import Packages

In [2]:
```python
import pandas as pd
import random
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
from numpy.random import randint
from random import choice
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Data Preparation

In [3]:
```python
data = pd.read_csv("diabetes.csv")
data = pd.DataFrame(data)
data
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [4]:
```python
# Check For Missing Values
data.isnull().any()
```

Out[4]:
```
Pregnancies                 False
Glucose                     False
BloodPressure               False
SkinThickness               False
Insulin                     False
BMI                         False
DiabetesPedigreeFunction    False
Age                         False
Outcome                     False
dtype: bool
```

In [5]:
```python
X = data.drop(['Outcome'],axis=1)
y = data['Outcome']
```

In [6]:
```python
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size=0.2 , random_state=42)
```

In [7]:
```python
print("X_train",X_train.shape)
print("X_test",X_test.shape)
```

```
X_train (614, 8)
X_test (154, 8)
```

Features Scaling

In [8]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
x_train_salced = scaler.transform(X_train)
x_test_salced = scaler.transform(X_test)
```

In [9]:
```python
pd.DataFrame(x_train_salced).head().style.set_caption('Data After Scaling')
```

Out[9]:

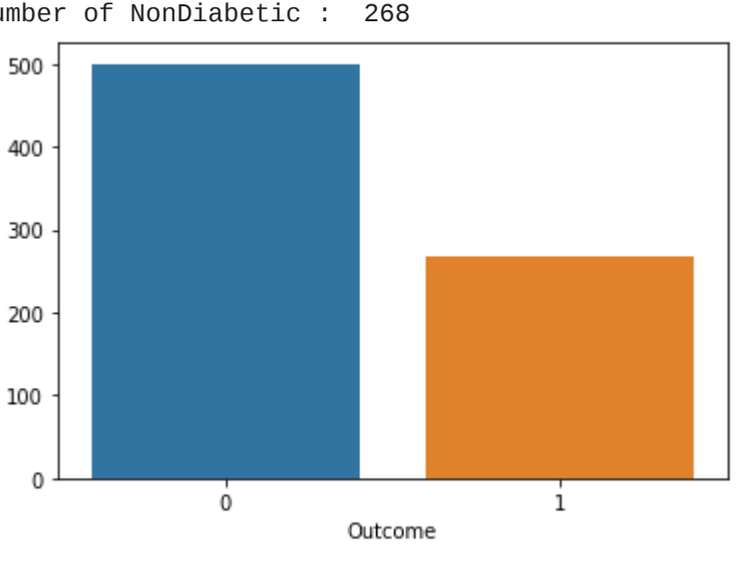
Data After Scaling

	0	1	2	3	4	5	6	7
0	-0.526397	-1.151398	-3.752683	-1.322774	-0.701206	-4.135256	-0.490735	-1.035940
1	1.588046	-0.276643	0.680345	0.233505	-0.701206	-0.489169	2.415030	1.487101
2	-0.828460	0.566871	-1.265862	-0.090720	0.013448	-0.424522	0.549161	-0.948939
3	-1.130523	1.254179	-1.049617	-1.322774	-0.701206	-1.303720	-0.639291	2.792122
4	0.681856	0.410665	0.572222	1.076490	2.484601	1.838121	-0.686829	1.139095

Data Visualization

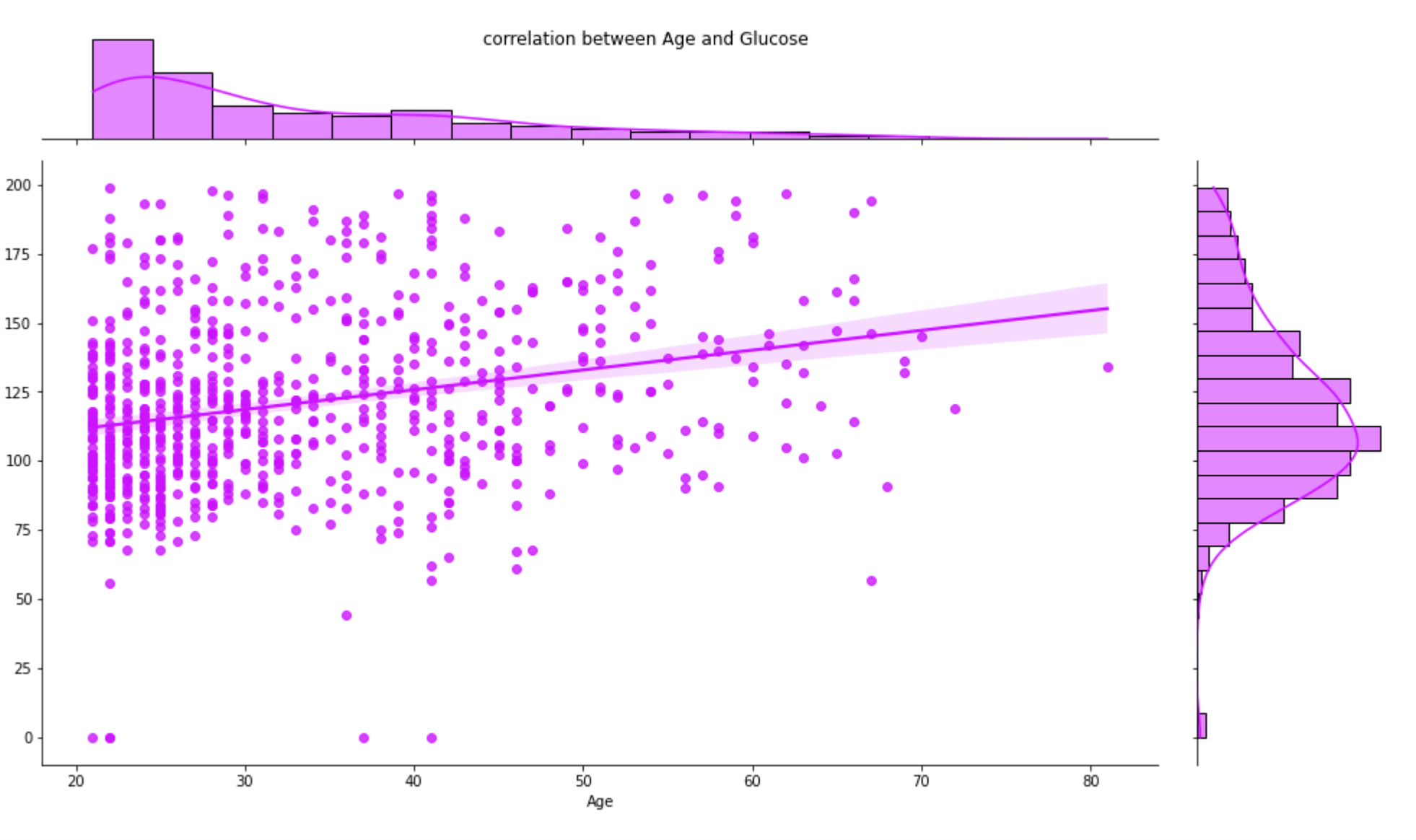
In [21]:
```python
ax = sns.countplot(y,label="Count")       # M = 212, B = 357
Diabetic, NonDiabetic = y.value_counts()
print('Number of Diabetic: ',Diabetic)
print('Number of NonDiabetic : ',NonDiabetic)
```

```
Number of Diabetic:  500
Number of NonDiabetic :  268
```

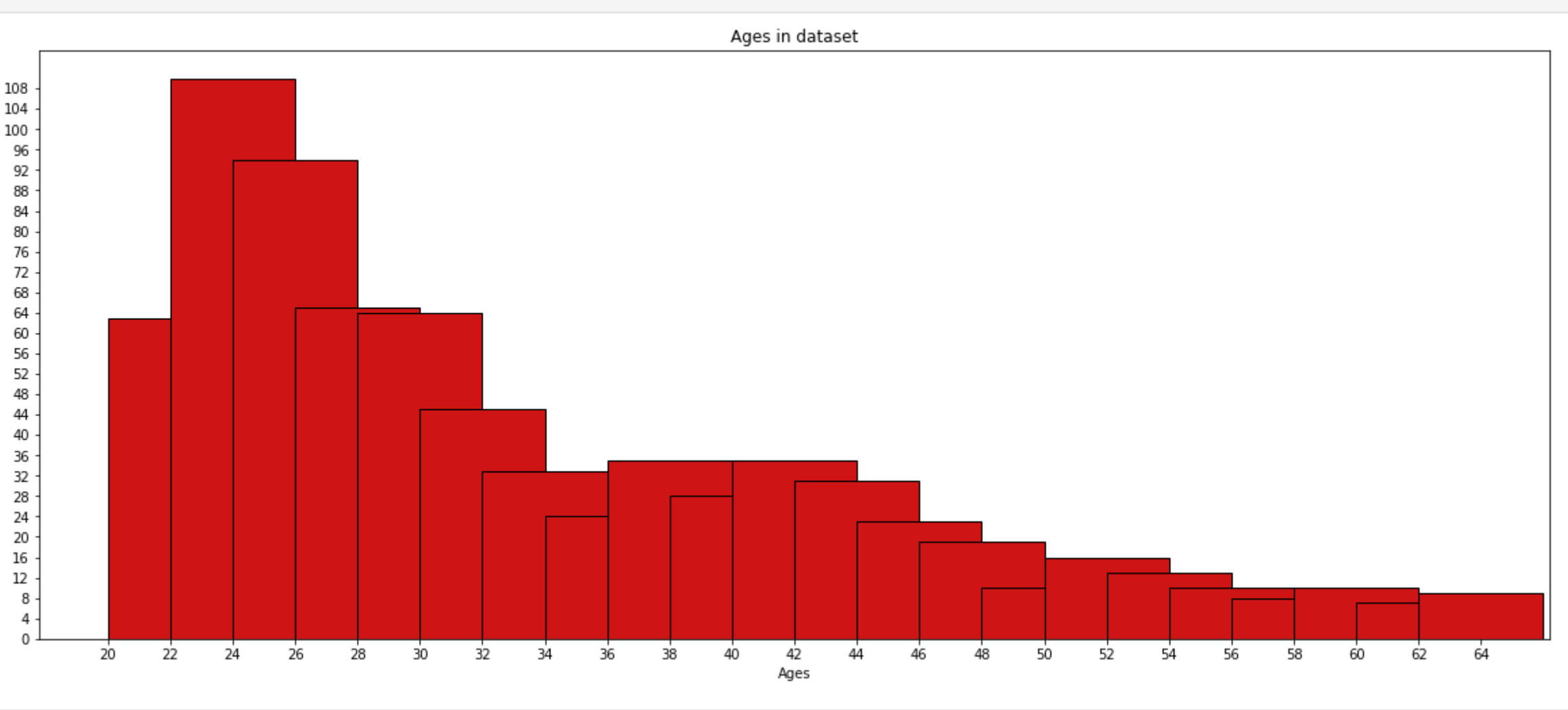


In [25]:
```python
correlation =  sns.jointplot(data.loc[:,'Age'], data.loc[:,'Glucose'], kind="reg", color="#ca13FF")
correlation.fig.set_figwidth(15)
correlation.fig.set_figheight(8)
correlation.fig.suptitle("correlation between Age and Glucose ")
```

Out[25]:
```
Text(0.5, 0.98, 'correlation between Age and Glucose ')
```

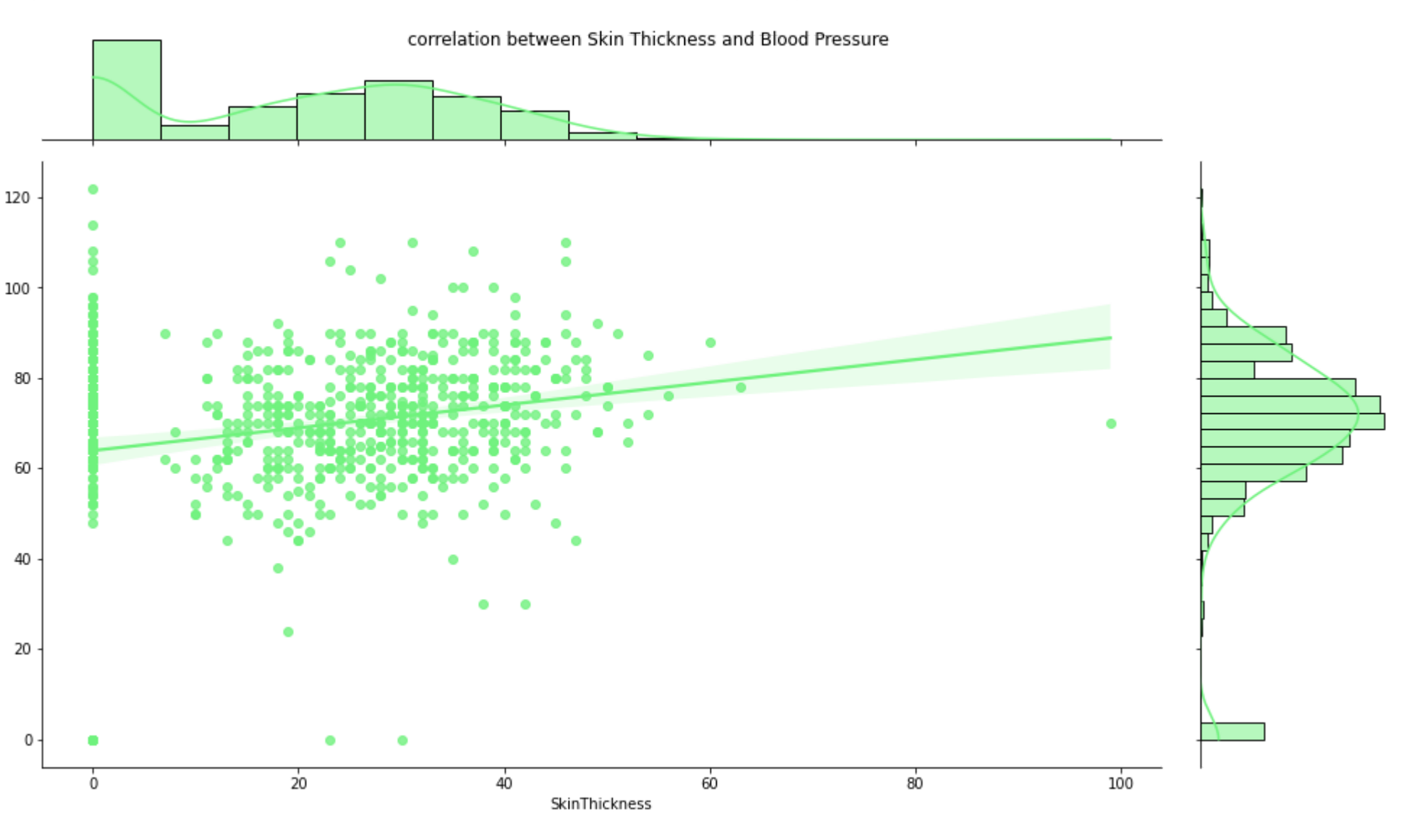


In [100…
```python
bins = list(range(20,65,2))
plt.figure(figsize = (20,8))
plt.hist(data['Age'].astype(int), width = 4, align = 'mid',
bins = bins, color = '#ce1414', edgecolor = 'black')
plt.xticks(bins)
plt.xlabel('Ages')
plt.title('Ages in dataset')
plt.yticks(np.arange(0,110,4))
plt.show()
```



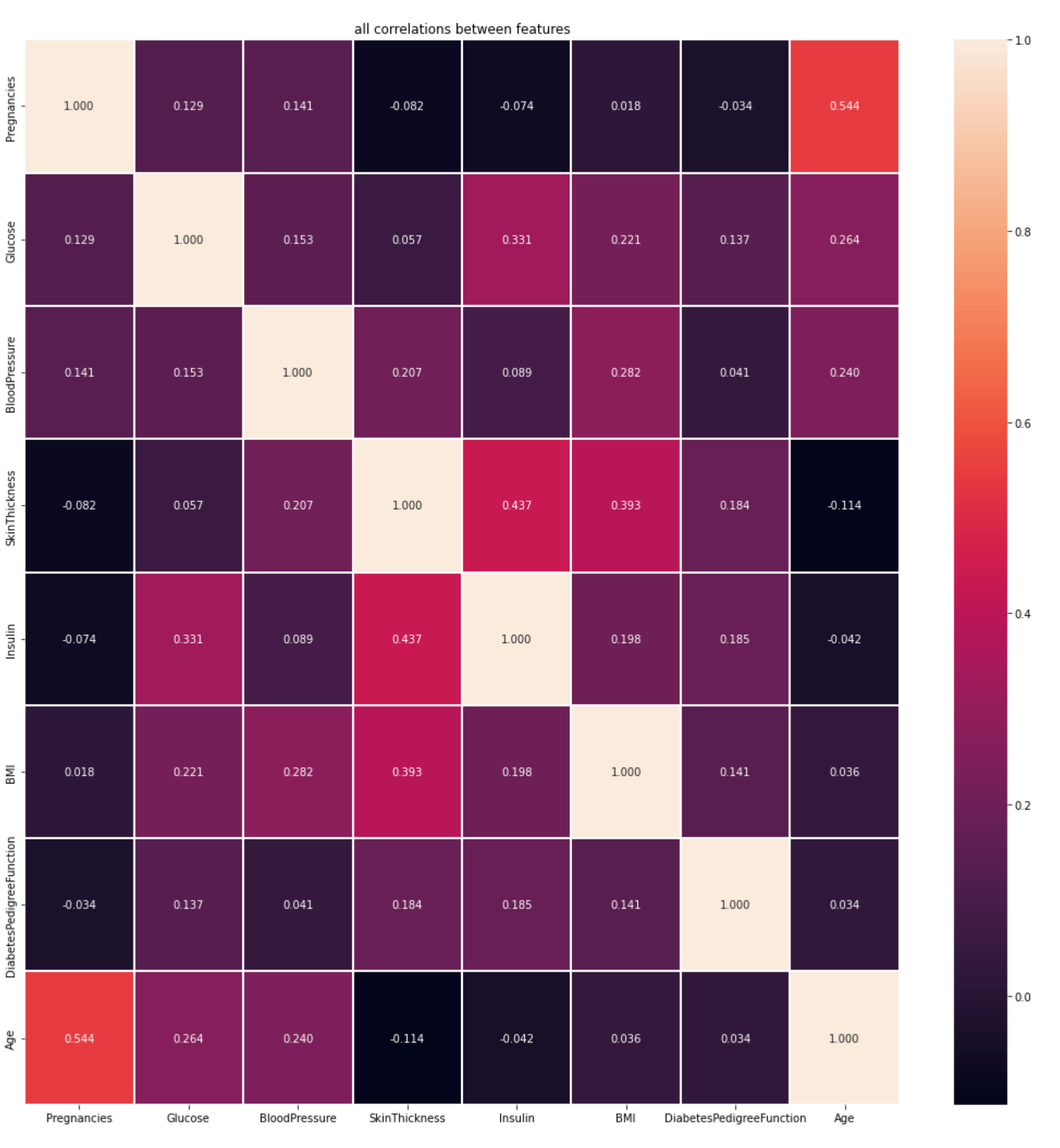
In [121…
```python
correlation =  sns.jointplot(data.loc[:,'SkinThickness'], data.loc[:,'BloodPressure'], kind="reg", color="#6ff27d")
correlation.fig.set_figwidth(15)
correlation.fig.set_figheight(8)
correlation.fig.suptitle("correlation between Skin Thickness and Blood Pressure ")
```

Out[121…
```
Text(0.5, 0.98, 'correlation between Skin Thickness and Blood Pressure ')
```



In [131…
```python
f,ax = plt.subplots(figsize=(18, 18))
plt.title("all correlations between features")
sns.heatmap(X.corr(), annot=True, linewidths=.2, fmt= '.3f',ax=ax)
```

Out[131…
```
<AxesSubplot:title={'center':'all correlations between features'}>
```



SVM Model Without GA

In [83]:
```python
svm = SVC(random_state=1)
svm.fit(x_train_salced,y_train)
```

Out[83]:
```
SVC
SVC(random_state=1)
```

In [84]:
```python
y_pred = svm.predict(x_train_salced)
y_pred_test = svm.predict(x_test_salced)
```

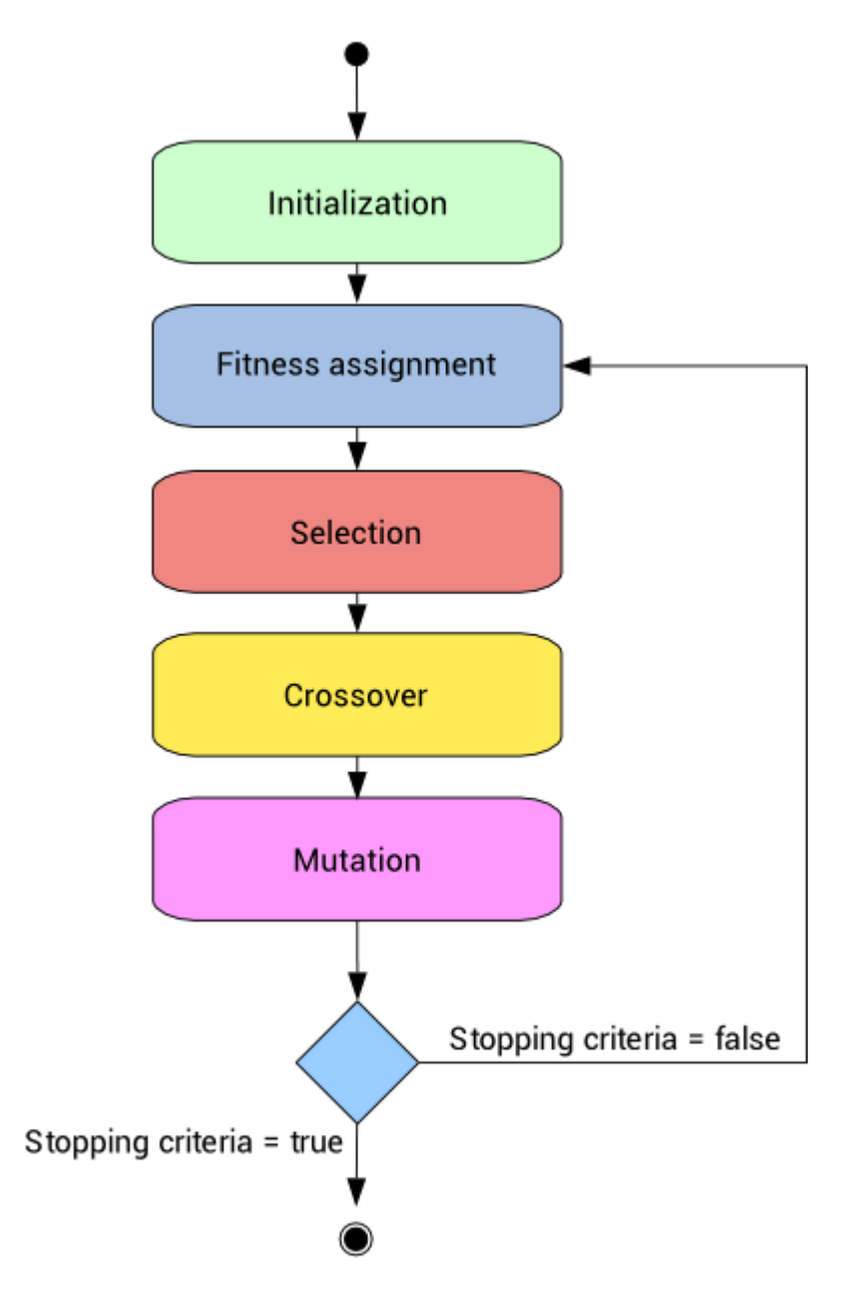
In [85]:
```python
print(accuracy_score(y_train,y_pred))
print(accuracy_score(y_test,y_pred_test))
```

```
0.8338762214983714
0.7337662337662337
```

SVM Model With GA

In [86]:
```python
def model_SVM(C ,gamma  , kernel):
    model = SVC(C=C ,gamma=gamma  , kernel=kernel)
    model.fit(x_train_salced,y_train)
    return model
```

Genetic Algorithm Steps



Initialization

In [87]:
```python
def initialization():
    parameters = {}

    C = randint(1,10)
    parameters["C"] = C

    gamma = choice(['auto' , 'scale'])
    parameters["gamma"] = gamma

    kernel = choice(['linear', 'poly', 'rbf', 'sigmoid'])
    parameters["kernel"] = kernel


    return parameters
```

In [88]:
```python
initialization()
```

Out[88]:
```
{'C': 3, 'gamma': 'auto', 'kernel': 'sigmoid'}
```

In [89]:
```python
def generate_population(n):
    population = []
    for i in range(n):
        chromosome = initialization()
        population.append(chromosome)
    return population
```

In [90]:
```python
generate_population(4)
```

Out[90]:
```
[{'C': 1, 'gamma': 'scale', 'kernel': 'sigmoid'},
 {'C': 7, 'gamma': 'scale', 'kernel': 'rbf'},
 {'C': 9, 'gamma': 'auto', 'kernel': 'poly'},
 {'C': 6, 'gamma': 'scale', 'kernel': 'rbf'}]
```

Fitness functions

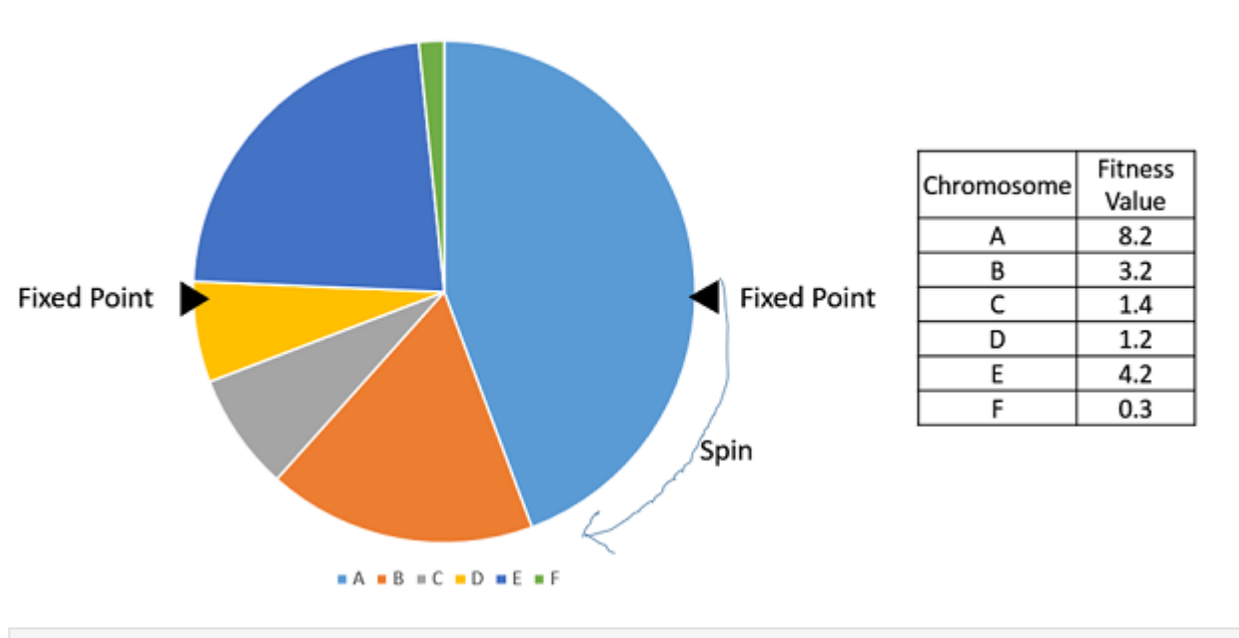
`are used in genetic programming and genetic algorithms to guide simulations towards optimal design solutions`

In [91]:
```python
def fitness_evaluation(model):
    y_pred_test = model.predict(x_test_salced)
    acc = accuracy_score(y_test,y_pred_test)
    return acc
```

Selection

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, **all the parents are chosen in just one spin** of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

`The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.`



In [92]:
```python
def selection(population_fitness):
    total = sum(population_fitness)
    percentage = [round((x/total) * 100) for x in population_fitness]
    selection_wheel = []
    for pop_index,num in enumerate(percentage):
        selection_wheel.extend([pop_index]*num)
    parent1_ind = choice(selection_wheel)
    parent2_ind = choice(selection_wheel)
    return [parent1_ind, parent2_ind]
```

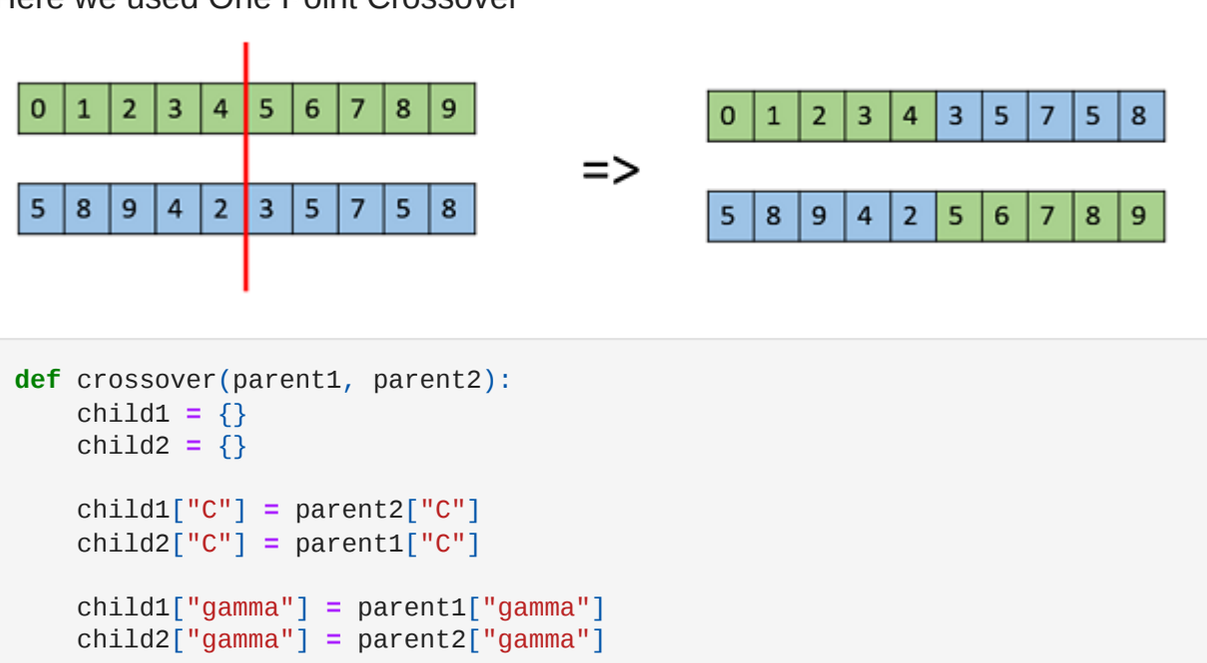
CrossOver

`a genetic operator used to combine the genetic information of two parents to generate new offspring`

Types of CrossOver

- One Point Crossover
- Multi Point Crossover
- Uniform Crossover
- Whole Arithmetic Recombination
- Davis' Order Crossover

Here we used One Point Crossover



In [93]:
```python
def crossover(parent1, parent2):
    child1 = {}
    child2 = {}

    child1["C"] = parent2["C"]
    child2["C"] = parent1["C"]

    child1["gamma"] = parent1["gamma"]
    child2["gamma"] = parent2["gamma"]

    child1["kernel"] = parent2["kernel"]
    child2["kernel"] = parent1["kernel"]

    return [child1, child2]
```

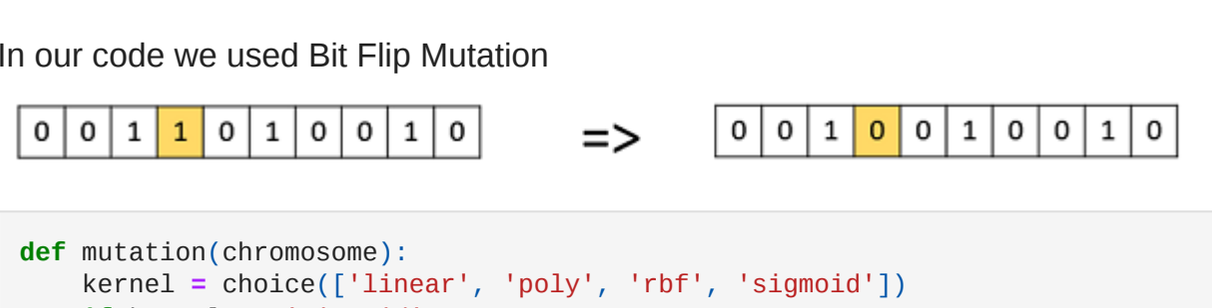
Mutation

`mutation may be defined as a small random tweak in the chromosome, to get a new solution`

Types of Mutation

- Bit Flip Mutation
- Random Resetting
- Swap Mutation
- Scramble Mutation
- Inversion Mutation

In our code we used Bit Flip Mutation



In [95]:
```python
def mutation(chromosome):
    kernel = choice(['linear', 'poly', 'rbf', 'sigmoid'])
    if kernel == 'sigmoid':
        chromosome["kernel"] = 'linear'
    return chromosome
```

Start GA

In [96]:
```python
generations = 500
acc_threshold = 90
num_pop = 10


population = generate_population(num_pop)
for generation in range(generations):
    population_fitness = []
    for chromosome in population:
        C = chromosome['C']
        gamma = chromosome['gamma']
        kernel = chromosome['kernel']

        model = model_SVM(C ,gamma  , kernel)

        acc = fitness_evaluation(model)
        print("Parameters: ", chromosome)
        print("Accuracy: ", round(acc,3))
        population_fitness.append(acc)

    parents_ind = selection(population_fitness)
    parent1 = population[parents_ind[0]]
    parent2 = population[parents_ind[1]]
    children = crossover(parent1, parent2)
    child1 = mutation(children[0])
    child2 = mutation(children[1])
    population.append(child1)
    population.append(child2)
    print("Generation ", generation+1," Outcome: ")

    if max(population_fitness) >= acc_threshold:
        print("Obtained desired accuracy: ", max(population_fitness))
        break
    else:
        print("Maximum accuracy in generation {} : {}".format(generation+1, max(population_fitness)))

    first_min = min(population_fitness)

    first_min_ind = population_fitness.index(first_min)

    population.remove(population[first_min_ind])

    second_min = min(population_fitness)

    second_min_ind = population_fitness.index(second_min)

    population.remove(population[second_min_ind])
```

```
Parameters:  {'C': 4, 'gamma': 'auto', 'kernel': 'sigmoid'}
Accuracy:  0.649
Parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'sigmoid'}
Accuracy:  0.643
Parameters:  {'C': 9, 'gamma': 'auto', 'kernel': 'rbf'}
Accuracy:  0.708
Parameters:  {'C': 6, 'gamma': 'scale', 'kernel': 'poly'}
Accuracy:  0.766
Parameters:  {'C': 9, 'gamma': 'scale', 'kernel': 'poly'}
Accuracy:  0.753
Parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'poly'}
Accuracy:  0.747
Parameters:  {'C': 8, 'gamma': 'auto', 'kernel': 'rbf'}
Accuracy:  0.701
Parameters:  {'C': 9, 'gamma': 'auto', 'kernel': 'sigmoid'}
Accuracy:  0.643
Parameters:  {'C': 7, 'gamma': 'scale', 'kernel': 'rbf'}
Accuracy:  0.708
Parameters:  {'C': 8, 'gamma': 'scale', 'kernel': 'sigmoid'}
Accuracy:  0.636
Generation  1  Outcome:
Maximum accuracy in generation 1 : 0.7662337662337663
Parameters:  {'C': 4, 'gamma': 'auto', 'kernel': 'sigmoid'}
Accuracy:  0.649
Parameters:  {'C': 1, 'gamma': 'scale', 'kernel': 'sigmoid'}
```

In [97]:
```
svm_acc = round(accuracy_score(y_test,y_pred_test),2)
best_acc = round(max(population_fitness),2)
print("SVM without GA: " , svm_acc)
print("SVM with GA: " , best_acc)
```

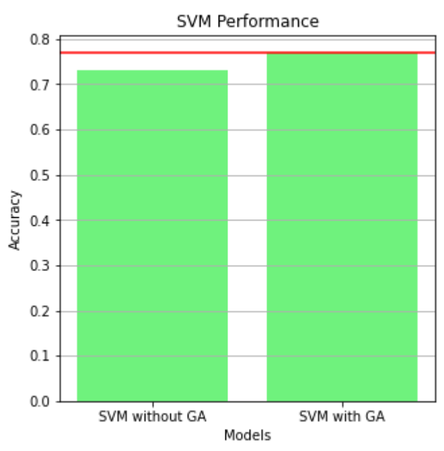
```
SVM without GA:  0.73
SVM with GA:  0.77
```

In [115…
```
acc = {"SVM without GA" : svm_acc, "SVM with GA" : best_acc}
models = list(acc.keys())
values = list(acc.values())

fig = plt.figure(figsize = (5, 5))

# creating the bar plot
plt.bar(models, values, color ='#6ff27d')

plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.grid(axis='y')
plt.axhline(y=0.77, color='#ff0000', linestyle='-')
plt.title("SVM Performance")
plt.show()
```



KNN without GA

When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2

In [76]:
```
knn =KNeighborsClassifier(p=1)
knn.fit(x_train_salced,y_train)
```

Out[76]:
```
KNeighborsClassifier(p=1)
```

In [77]:
```
y_pred_knn = knn.predict(x_train_salced)
y_pred_test_knn = knn.predict(x_test_salced)
```

In [78]:
```
print(accuracy_score(y_train,y_pred_knn))
print(accuracy_score(y_test,y_pred_test_knn))
```

```
0.8061889250814332
0.6623376623376623
```

KNN with GA

In [36]:
```
def model_KNN(n_neighbors , p):
    knn_GA =KNeighborsClassifier(n_neighbors=n_neighbors  ,  p=p )
    knn_GA.fit(x_train_salced,y_train)
    return knn_GA
```

In [51]:
```
def initialization_KNN():
    parameters = {}

    n_neighbors = randint(5,50)
    parameters["n_neighbors"] = n_neighbors

    p = choice([2,1])
    parameters["p"] = p


    return parameters
```

In [38]:
```
def generate_population_KNN(n):
    population_KNN = []
    for i in range(n):
        chromosome = initialization_KNN()
        population_KNN.append(chromosome)
    return population_KNN
```

In [39]:
```
generate_population_KNN(4)
```

Out[39]:
```
[{'n_neighbors': 20, 'p': 2},
 {'n_neighbors': 34, 'p': 2},
 {'n_neighbors': 12, 'p': 2},
 {'n_neighbors': 16, 'p': 2}]
```

In [45]:
```
def fitness_evaluation_KNN(model):
    y_pred_test_KNN_GA = model.predict(x_test_salced)
    acc_KNN = accuracy_score(y_test,y_pred_test_KNN_GA)
    return acc_KNN
```

In [47]:
```
def selection_KNN(population_fitness_KNN):
    total = sum(population_fitness_KNN)
    percentage = [round((x/total) * 100) for x in population_fitness_KNN]
    selection_wheel = []
    for pop_index,num in enumerate(percentage):
        selection_wheel.extend([pop_index]*num)
    parent1_ind_KNN = choice(selection_wheel)
    parent2_ind_KNN = choice(selection_wheel)
    return [parent1_ind_KNN, parent2_ind_KNN]
```

In [42]:
```
def crossover_KNN(parent1_KNN, parent2_KNN):
    child1_KNN = {}
    child2_KNN = {}

    child1_KNN["n_neighbors"] = parent2_KNN["n_neighbors"]
    child2_KNN["n_neighbors"] = parent1_KNN["n_neighbors"]

    child1_KNN["p"] = parent2_KNN["p"]
    child2_KNN["p"] = parent1_KNN["p"]

    return [child1_KNN, child2_KNN]
```

In [43]:
```
def mutation_KNN(chromosome):
    flag = randint(1,50)
    if flag >= 40:
        chromosome["n_neighbors"] = 20
    return chromosome
```

In [53]:
```
generations = 500
acc_threshold = 90
num_pop = 10


population_KNN = generate_population_KNN(num_pop)
for generation in range(generations):
    population_fitness_KNN = []
    for chromosome in population_KNN:
        n_neighbors = chromosome['n_neighbors']
        p = chromosome['p']

        model_KNN_GA = model_KNN(n_neighbors , p)

        acc_KNN_GA = fitness_evaluation_KNN(model_KNN_GA)
        print("Parameters: ", chromosome)
        print("Accuracy: ", round(acc_KNN_GA,3))
        population_fitness_KNN.append(acc_KNN_GA)

    parents_ind_KNN = selection_KNN(population_fitness_KNN)
    parent1_KNN = population_KNN[parents_ind_KNN[0]]
    parent2_KNN = population_KNN[parents_ind_KNN[1]]
    children_KNN = crossover_KNN(parent1_KNN, parent2_KNN)
    child1_KNN = mutation_KNN(children_KNN[0])
    child2_KNN = mutation_KNN(children_KNN[1])
    population_KNN.append(child1_KNN)
    population_KNN.append(child2_KNN)
    print("Generation ", generation+1," Outcome: ")

    if max(population_fitness_KNN) >= acc_threshold:
        print("Obtained desired accuracy: ", max(population_fitness_KNN))
        break
    else:
        print("Maximum accuracy in generation {} : {}".format(generation+1, max(population_fitness_KNN)))

    first_min_KNN = min(population_fitness_KNN)

    first_min_ind_KNN = population_fitness_KNN.index(first_min_KNN)

    population_KNN.remove(population_KNN[first_min_ind_KNN])

    second_min_KNN = min(population_fitness_KNN)

    second_min_ind_KNN = population_fitness_KNN.index(second_min_KNN)

    population_KNN.remove(population_KNN[second_min_ind_KNN])
```

```
Parameters:  {'n_neighbors': 36, 'p': 2}
Accuracy:  0.753
Parameters:  {'n_neighbors': 48, 'p': 1}
Accuracy:  0.773
Parameters:  {'n_neighbors': 16, 'p': 2}
Accuracy:  0.747
Parameters:  {'n_neighbors': 42, 'p': 2}
Accuracy:  0.753
Parameters:  {'n_neighbors': 43, 'p': 1}
Accuracy:  0.786
Parameters:  {'n_neighbors': 46, 'p': 1}
Accuracy:  0.773
Parameters:  {'n_neighbors': 12, 'p': 2}
Accuracy:  0.721
Parameters:  {'n_neighbors': 28, 'p': 2}
Accuracy:  0.753
Parameters:  {'n_neighbors': 31, 'p': 1}
Accuracy:  0.766
Parameters:  {'n_neighbors': 25, 'p': 2}
Accuracy:  0.74
Generation  1  Outcome:
Maximum accuracy in generation 1 : 0.7857142857142857
```

After Chosing Best Parameter

In [82]:
```
knn_GA_last =KNeighborsClassifier(n_neighbors=43,p=1)
knn_GA_last.fit(x_train_salced,y_train)
y_pred_knn_last = knn_GA_last.predict(x_train_salced)
y_pred_test_knn_last = knn_GA_last.predict(x_test_salced)
print("KNN without GA: ",accuracy_score(y_test,y_pred_test_knn))
print("KNN with GA: " , accuracy_score(y_test,y_pred_test_knn_last))
```

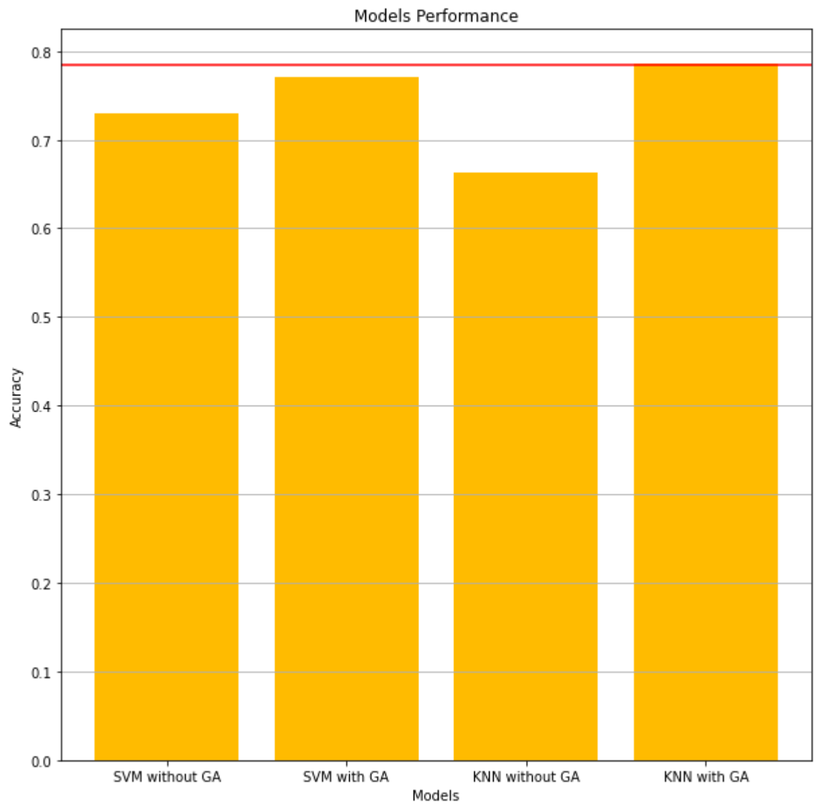
```
KNN without GA:  0.6623376623376623
KNN with GA:  0.7857142857142857
```

In [99]:
```
KNN_acc = accuracy_score(y_test,y_pred_test_knn)
best_knn_acc = accuracy_score(y_test,y_pred_test_knn_last)
```

In [114…
```
acc = {"SVM without GA" : svm_acc,
       "SVM with GA" : best_acc ,
       "KNN without GA" : KNN_acc,
      "KNN with GA" : best_knn_acc
      }
models = list(acc.keys())
values = list(acc.values())

fig = plt.figure(figsize = (10, 10))

# creating the bar plot
plt.bar(models, values, color ='#ffbb00')
plt.axhline(y=0.785, color='#ff0000', linestyle='-')
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.grid(axis='y')
plt.title("Models Performance")
plt.show()
```



This Part for explanation only for how selection function work

In [165…
```
test_population_fitness = [72, 51,91,99]
```

In [182…
```
total = sum(test_population_fitness)
percentage = [round((x/total) * 100) for x in test_population_fitness]
selection_wheel = []
for pop_index,num in enumerate(percentage):
    selection_wheel.extend([pop_index]*num)
parent1_ind_KNN = choice(selection_wheel)
parent2_ind_KNN = choice(selection_wheel)
```

In [189…
```
percentage
```

Out[189…
```
[23, 16, 29, 32]
```

In [198…
```
mylabels = [23, 16, 29, 32]
plt.figure(figsize=(20,8))
plt.pie(percentage,labels=mylabels)
plt.title("selection wheel")
plt.show()
```