In [429]:
```
# Importing Libraries
import numpy as np
import pandas as pd
```

In [430]:
```
#reading .csv file and storing it in "dataset" dataframe
dataset=pd.read_csv("/home/ankit/Desktop/Iris.csv")
```

In [431]:
```
print(dataset.head()) # printing the whole dataset
print(dataset['Species'].unique()) # printing the unique classes of iris da
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [432]:
```
print(dataset.shape) # size of the dataset
```

```
(150, 6)
```

In [433]:
```
#X= dataset.drop('Species',axis=1)
#Y= dataset['Species']
```

In [434]:
```
#print(X)
```

In [435]:
```
#print(Y)
```

In [436]:
```
data = dataset.to_numpy() # storing iris dataset as numpy array
```

In [437]:
```
for i in range(150):
    data[i][0]=1 # adding 1st column as bias=1

```

In [438]:
```
print(data)
```

```
[[1 5.1 3.5 1.4 0.2 'Iris-setosa']
 [1 4.9 3.0 1.4 0.2 'Iris-setosa']
 [1 4.7 3.2 1.3 0.2 'Iris-setosa']
 [1 4.6 3.1 1.5 0.2 'Iris-setosa']
 [1 5.0 3.6 1.4 0.2 'Iris-setosa']
 [1 5.4 3.9 1.7 0.4 'Iris-setosa']
 [1 4.6 3.4 1.4 0.3 'Iris-setosa']
 [1 5.0 3.4 1.5 0.2 'Iris-setosa']
 [1 4.4 2.9 1.4 0.2 'Iris-setosa']
 [1 4.9 3.1 1.5 0.1 'Iris-setosa']
 [1 5.4 3.7 1.5 0.2 'Iris-setosa']
 [1 4.8 3.4 1.6 0.2 'Iris-setosa']
 [1 4.8 3.0 1.4 0.1 'Iris-setosa']
 [1 4.3 3.0 1.1 0.1 'Iris-setosa']
 [1 5.8 4.0 1.2 0.2 'Iris-setosa']
 [1 5.7 4.4 1.5 0.4 'Iris-setosa']
 [1 5.4 3.9 1.3 0.4 'Iris-setosa']
 [1 5.1 3.5 1.4 0.3 'Iris-setosa']
 [1 5.7 3.8 1.7 0.3 'Iris-setosa']
 [1 5.1 3.8 1.5 0.3 'Iris-setosa']
```

In [439]:
```
from matplotlib import pyplot as plt


for i in range(len(data)):
    if data[i][-1]=='Iris-setosa':
        data[i][-1]=1 # relabeling Iris setosa class as "0"
    if data[i][-1]=='Iris-versicolor':
        data[i][-1]=2 # relabeling Iris versicolor class as "1"
```

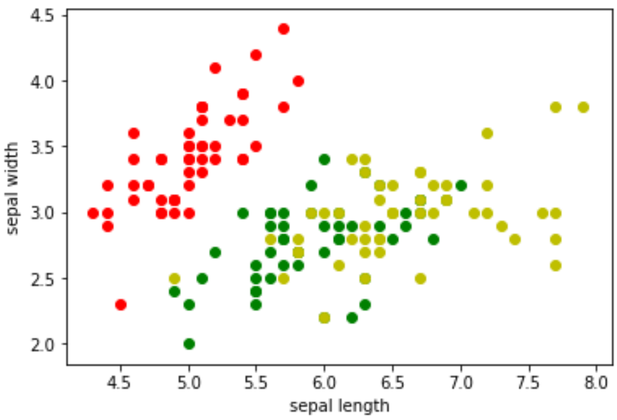
```
    if data[i][-1]=='Iris-virginica':
        data[i][-1]=3 # relabeling Iris virginica class as "0"
    data[i][0]=1 # initializing first column as 1 for bias
```

In [440]:

```
#data
```

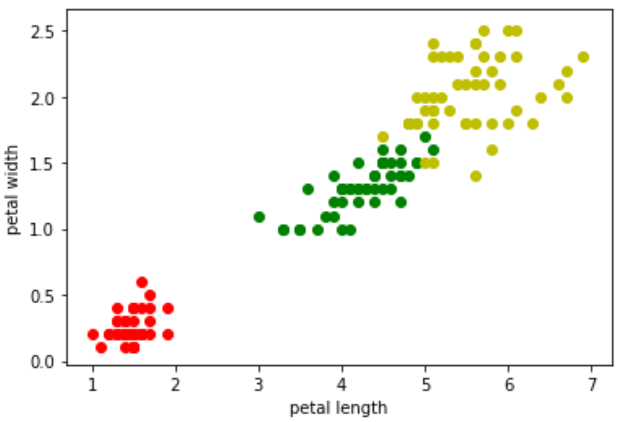
In [441]:

```
for i in range(len(data)):
    if data[i][-1]==1:
        plt.scatter(data[i,1],data[i,2],c='r')
    if data[i][-1]==2:
        plt.scatter(data[i,1],data[i,2],c='g')
    if data[i][-1]==3:
        plt.scatter(data[i,1],data[i,2],c='y')
    plt.xlabel('sepal length')
    plt.ylabel('sepal width')

plt.show()     # a plot between sepal length and sepal width
```



In [442]:

```
for i in range(len(data)):
    if data[i][-1]==1:
        plt.scatter(data[i,3],data[i,4],c='r')
    if data[i][-1]==2:
        plt.scatter(data[i,3],data[i,4],c='g')
    if data[i][-1]==3:
        plt.scatter(data[i,3],data[i,4],c='y')
    plt.xlabel('petal length')
    plt.ylabel('petal width')

plt.show()  # a plot between petal length and petal width
```



Implementing Forward Propagation MLP for Iris dataset

In [443]:

```
input_no_of_nodes = 4 # as features are 4-dimensional i.e. sepal length,pet
# and petal width
no_of_hidden_layers = 1 # just for simplicity , I am taking one hidden laye
```

```python
no_of_classes = 3 # setosa,virginica and versicolor

# initializing input weights for output layer

weight1 = [] # initializing weights


for k in range(3):
    w=[]
    for j in range(5):
        w.append(np.random.randn())
    weight1.append(w)

print("input weight vector for output layer:\n ",weight1)

# initializing input weights for hidden layer

weight2 = [] # initializing weights

bias= 1



for k in range(4):
    w=[]
    for j in range(5):
        w.append(np.random.randn())
    weight2.append(w)

print("\ninput weight vector for hidden layer:\n ",weight2)
```

```
input weight vector for output layer:
  [[0.18204651248854653, -0.3195494800785854, 0.8506967596147637, -1.527883060
1839293, 0.8468775949912822], [-0.7105553426733896, 1.5427545553212112, -0.521
2290329153958, 1.3199837146608846, 1.3027364165285165], [-1.2581738482758533,
0.1954909533767402, -0.5293114051138316, 0.8982568826527799, -1.26789205684944
4]]

input weight vector for hidden layer:
  [[0.012079880287338243, -1.0994786150270182, 0.6706515214739192, 0.696574387
9098967, 0.874415795996476], [1.0953417415957751, 2.2584676519018334, 0.326125
3541951244, 0.32904108755751343, 0.5200729353112302], [1.1755338637157928, 0.1
0241462477627439, 0.9286554122455126, 0.6679682394666533, -0.223888796096033
7], [0.309703588072365, -1.740283841682843, -0.2776661070260824, 1.73557043354
72464, -0.9442540729403734]]
```

In [444]:

```python
#for dk(p), if class label belongs to class 1, we make 1st value as 1 and
#others

labels=[]
for i in range(150):
    if data[i][-1]==1:
        labels.append([1,0,0])
    if data[i][-1]==2:
        labels.append([0,1,0])
    if data[i][-1]==3:
        labels.append([0,0,1])
```

In [445]:

```python
        forward_prop_output=[]

    # Input on input layer -1 (hidden layer )  by using McCullouch Pit mode

        u1=[]
        for j in range(4):
            sum=0
            for i in range(5):
                sum += weight2[j][i]*data[j][i]
            u1.append(sum)
         #print("Input on layer 1 (hidden layer) : ",u)
```

```python

      # For output on layer 1, by using sigmoid function,

     v1=[]
     v1.append(1) # appending bias=1

     for i in range(4):
         sig=1/(1+ np.exp(-u1[i]))
         v1.append(sig)
     #print("output on layer 1 : ",v)


     # For input on layer 2 by using McCullouch Pit model,

     u2=[]
     for j in range(3):
         sum=0
         for i in range(5):
             sum += weight1[j][i]*v1[i]
         u2.append(sum)
        #print("input of layer 2",u)


     # For output on layer 2(output layer), by using sigmoid function,

     v2=[]

     for i in range(3):
         sig=1/(1+ np.exp(-u2[i]))
         v2.append(sig)
        #print("output on layer 2 (output layer) : ",v)
     forward_prop_output.append(v2)
     yk=forward_prop_output
```

In [446]:

```python
def forward_prop(p):
     forward_prop_output=[]

 # Input on input layer -1 (hidden layer )  by using McCullouch Pit mode

     u1=[]
     for j in range(4):
         sum=0
         for i in range(5):
             sum += weight2[j][i]*data[p][i]
         u1.append(sum)
      #print("Input on layer 1 (hidden layer) : ",u)

      # For output on layer 1, by using sigmoid function,

     v1=[]
     v1.append(1) # appending bias=1

     for i in range(4):
         sig=1/(1+ np.exp(-u1[i]))
         v1.append(sig)
     #print("output on layer 1 : ",v)


     # For input on layer 2 by using McCullouch Pit model,

     u2=[]
     for j in range(3):
         sum=0
         for i in range(5):
             sum += weight1[j][i]*v1[i]
         u2.append(sum)
        #print("input of layer 2",u)
```

```
        # For output on layer 2(output layer), by using sigmoid function,

        v2=[]

        for i in range(3):
            sig=1/(1+ np.exp(-u2[i]))
            v2.append(sig)
           #print("output on layer 2 (output layer) : ",v)
        forward_prop_output.append(v2)
        yk=forward_prop_output

        return yk
```

In [447]:

```
#for dk(p), if class label belongs to class 1, we make 1st value as 1 and c
#others

labels=[]
for i in range(150):
    if data[i][-1]==1:
        labels.append([1,0,0])
    if data[i][-1]==2:
        labels.append([0,1,0])
    if data[i][-1]==3:
        labels.append([0,0,1])
```

In [448]:

```
labels
```

Out[448]:

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

Implementing Backpropagation MLP for iris dataset

In [449]:

```
def back_prop(p):
    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming


    forward_prop_output=[]

    # Input on input layer -1 (hidden layer )  by using McCullouch Pit mode

    u1=[]
    for j in range(4):
        sum=0
        for i in range(5):
            sum += weight2[j][i]*data[p][i]
        u1.append(sum)
```

```
        #print("Input on layer 1 (hidden layer) : ",u)

        # For output on layer 1, by using sigmoid function,

    v1=[]
    v1.append(1) # appending bias=1

    for i in range(4):
        sig=1/(1+ np.exp(-u1[i]))
        v1.append(sig)
    #print("output on layer 1 : ",v)


    # For input on layer 2 by using McCullouch Pit model,

    u2=[]
    for j in range(3):
        sum=0
        for i in range(5):
            sum += weight1[j][i]*v1[i]
        u2.append(sum)
        #print("input of layer 2",u)


    # For output on layer 2(output layer), by using sigmoid function,

    v2=[]

    for i in range(3):
        sig=1/(1+ np.exp(-u2[i]))
        v2.append(sig)
        #print("output on layer 2 (output layer) : ",v)
    forward_prop_output.append(v2)
    yk=forward_prop_output




    for k in range(3):
        for j in range(5):
            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k]

    delta_w_kj_1=learning_rate*delta_w_kj_1
    #print(delta_w_kj_1)


    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming


    for k in range(3):
        for j in range(5):
            for l in range(5):
                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0][

    delta_w_ji_0=learning_rate*delta_w_ji_0
    #print(delta_w_ji_0)

    # modifying weight1 and weight2 according to delta_w
    #print("previous weight1 matrix : ", weight1)
    temp1=0
    for k in range(3):
        for j in range(5):
            temp1 =weight1[k][j]
            weight1[k][j] =temp1+delta_w_kj_1


```

```python
    #print("modified input weight vector for output layer:\n ",weight1)

    #print("Previous weight2 matrix : ", weight2)

    temp2=0
    for k in range(4):
        for j in range(5):
            temp2=weight2[k][j]
            weight2[k][j] = temp2+ delta_w_ji_0


    #print("modified input weight vector for hidden layer:\n ",weight2)

    return

```

In [450]:

```python
error1=0
for p in range(150):
    output1=forward_prop(p)
    #print(output)
    for j in range(3):
        error1 += pow((labels[p][j] - output1[0][j]),2)
    E1 = error1/2
print(E1)     #error after forward propagation initially
```

```
56.751504851834625
```

In [451]:

```python
for n in range(50):# number of iterations
    error=0
    for i in range(150):
        output=forward_prop(i)
        #print(output)
        #error = 0
        for j in range(3):
            error += pow((labels[i][j] - output[0][j]),2)
        E = error/2
        #print(E)
        #if E<0.1:
         #   break
        back_prop(i)
print("Minimized error after 100 iteration: ", E) # Our objective was to Mi

# if error difference is not much then incearease the number of iteration b
```

```
Minimized error after 100 iteration:  54.775238213652585
```

In [452]:

```python
print("Final input weight matrix for hidden layer: \n",weight2)
print("\nFinal input weight matrix for output layer: \n",weight1)
```

```
Final input weight matrix for hidden layer:
 [[-0.2431105734390827, -1.3546690687534335, 0.4154610677474975, 0.44138393418
347427, 0.6192253422700504], [0.8401512878693501, 2.003277198175414, 0.0709349
0046870354, 0.07385063383109278, 0.26488248158481115], [0.9203434099893685, -
0.15277582895014669, 0.673464958519087, 0.41277778574023155, -0.47907924982245
27], [0.05451313434594388, -1.9954742954092584, -0.5328565607525014, 1.4803799
79820831, -1.1994445266667888]]

Final input weight matrix for output layer:
 [[0.2529551252114036, -0.24864086735572838, 0.9216053723376181, -1.4569744474
610702, 0.9177862077141367], [-0.6396467299505352, 1.6136631680440703, -0.4503
204201925382, 1.3908923273837437, 1.3736450292513755], [-1.1872652355529942,
0.2663995660995988, -0.45840279239097464, 0.9691654953756343, -1.1969834441265
85]]
```

Incorporating momentum factor

In [423]:

```python
gamma=0.9
```

```
mom1=0
mom2=0
```

In [424]:

```
#Modifying little bit the definition of backpropagation for error learning

def back_prop(p,mom1,mom2):
    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming

    forward_prop_output=[]

    # Input on input layer -1 (hidden layer )  by using McCullouch Pit mod

    u1=[]
    for j in range(4):
        sum=0
        for i in range(5):
            sum += weight2[j][i]*data[p][i]
        u1.append(sum)
     #print("Input on layer 1 (hidden layer) : ",u)

    # For output on layer 1, by using sigmoid function,

    v1=[]
    v1.append(1) # appending bias=1

    for i in range(4):
        sig=1/(1+ np.exp(-u1[i]))
        v1.append(sig)
    #print("output on layer 1 : ",v)


    # For input on layer 2 by using McCullouch Pit model,

    u2=[]
    for j in range(3):
        sum=0
        for i in range(5):
            sum += weight1[j][i]*v1[i]
        u2.append(sum)
        #print("input of layer 2",u)


    # For output on layer 2(output layer), by using sigmoid function,

    v2=[]

    for i in range(3):
        sig=1/(1+ np.exp(-u2[i]))
        v2.append(sig)
        #print("output on layer 2 (output layer) : ",v)
    forward_prop_output.append(v2)
    yk=forward_prop_output


    for k in range(3):
        for j in range(5):
            delta_w_kj_1 = (labels[p][k] - yk[0][k])*(yk[0][k])*(1-yk[0][k

    delta_w_kj_1=learning_rate*delta_w_kj_1
    #print(delta_w_kj_1)


    # computing delta w_ji(0) (learning rule) #weight2

    learning_rate = 0.5 #assuming


```

```
    for k in range(3):
        for j in range(5):
            for l in range(5):
                delta_w_ji_0 = (labels[p][k] - yk[0][k])*yk[0][k]*(1-yk[0]

    delta_w_ji_0=learning_rate*delta_w_ji_0
    #print(delta_w_ji_0)

    # modifying weight1 and weight2 according to delta_w
    #print("previous weight1 matrix : ", weight1)

    # Here, I am using the concept of momentum factor
    mom1 += pow(gamma,i)*(delta_w_kj_1)
    mom2 += pow(gamma,i)*(delta_w_ji_0)

    temp1=0
    for k in range(3):
        for j in range(5):
            temp1 =weight1[k][j]
            weight1[k][j] =temp1+(mom1 + delta_w_kj_1)


    #print("modified input weight vector for output layer:\n ",weight1)

    #print("Previous weight2 matrix : ", weight2)

    temp2=0
    for k in range(4):
        for j in range(5):
            temp2=weight2[k][j]
            weight2[k][j] = temp2+ (mom2 + delta_w_ji_0)


    #print("modified input weight vector for hidden layer:\n ",weight2)

    return mom1,mom2

```

In [425]:

```
for n in range(100):# number of iterations
    error=0
    for i in range(150):
        output=forward_prop(i)
        #print(output)
        #error = 0
        for j in range(3):
            error += pow((labels[i][j] - output[0][j]),2)
        E = error/2
        #print(E)
        #if E<0.1:
         #   break

        back_prop(i,mom1,mom2)

print("Minimized error after 100 iteration: ", E) # Our objective was to Mi

# if error difference is not much then incearease the number of iteration b
```

```
Minimized error after 100 iteration:  43.510786166850934
```

In [ ]:

```

```