In [1]:
```
import random
import seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from pandas.plotting import scatter_matrix


seaborn.set(style='whitegrid'); seaborn.set_context('talk')
%matplotlib inline

from sklearn.datasets import load_iris
iris_data = load_iris()
```

Import datasets

In [2]:
```
dataset = pd.read_csv("Iris.csv")
```

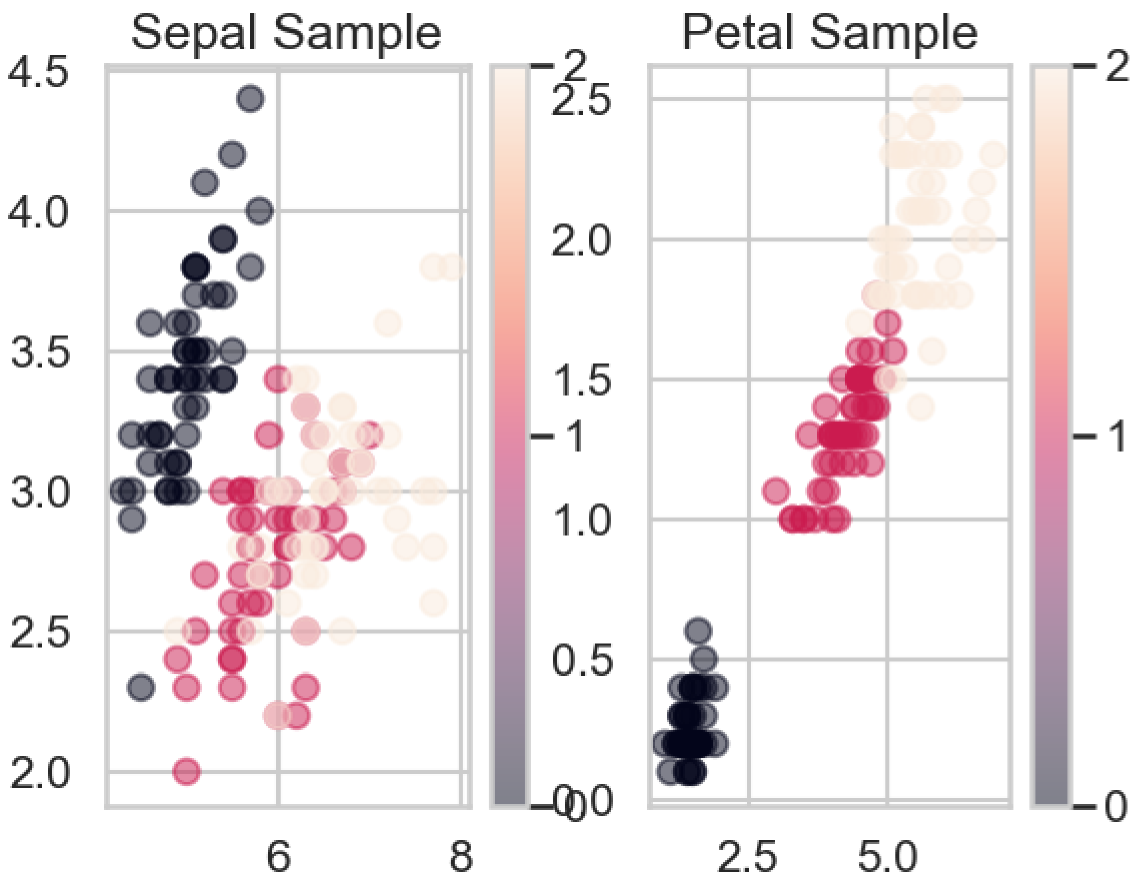
In [3]:
```
print(iris_data['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
```

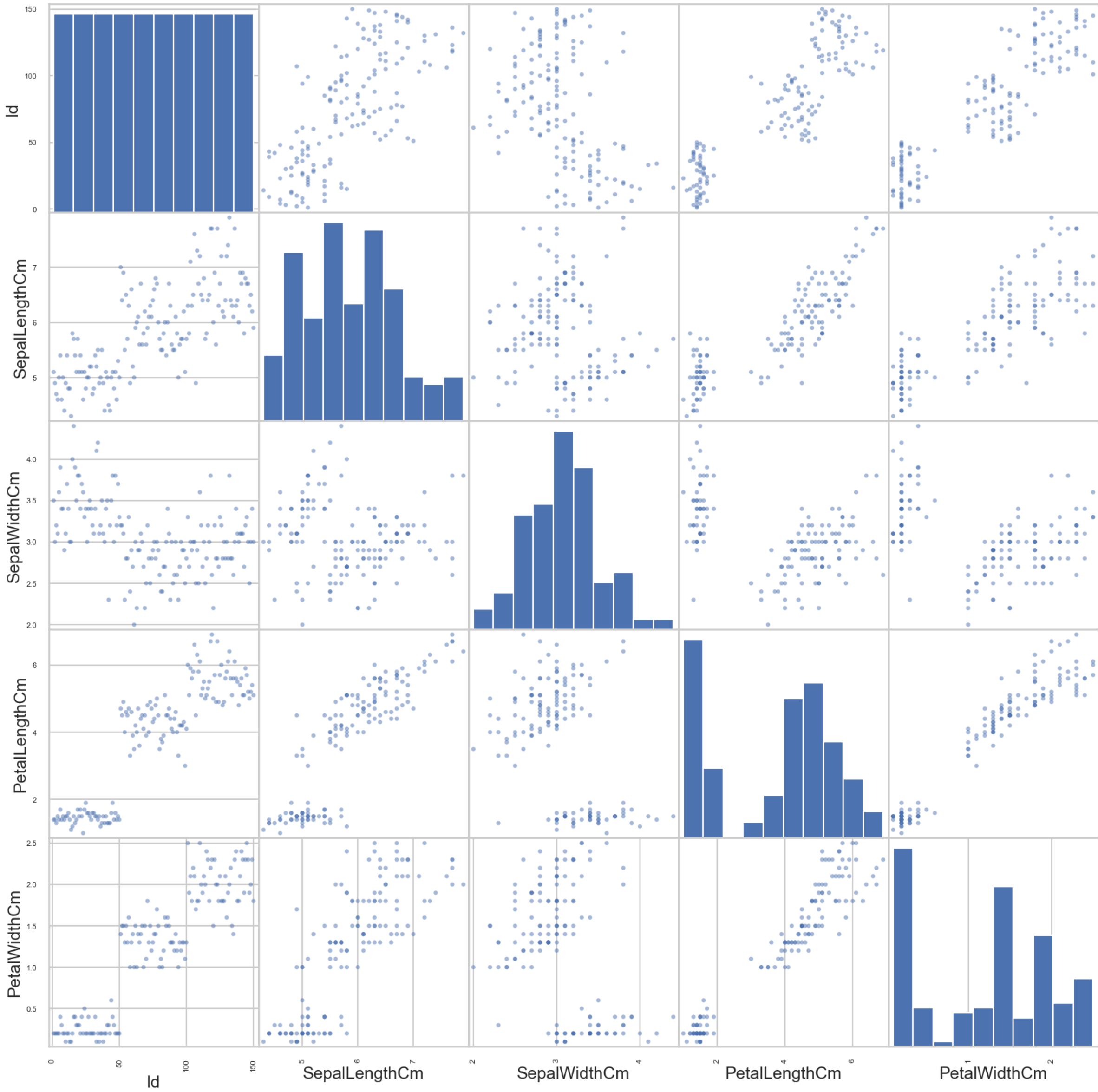
```
   conceptual clustering system finds 3 classes in the data.
 - Many, many more ...
```

In [4]:
```
n_samples, n_features = iris_data.data.shape

plt.subplot(1, 2, 1)
scatter_plot = plt.scatter(iris_data.data[:,0], iris_data.data[:,1], alpha=0.5,
                           c=iris_data.target)
plt.colorbar(ticks=([0, 1, 2]))
plt.title('Sepal Sample')

plt.subplot(1, 2, 2)
scatter_plot_2 = plt.scatter(iris_data.data[:,2], iris_data.data[:,3], alpha=0.5,
                           c=iris_data.target)
plt.colorbar(ticks=([0, 1, 2]))
plt.title('Petal Sample')
```

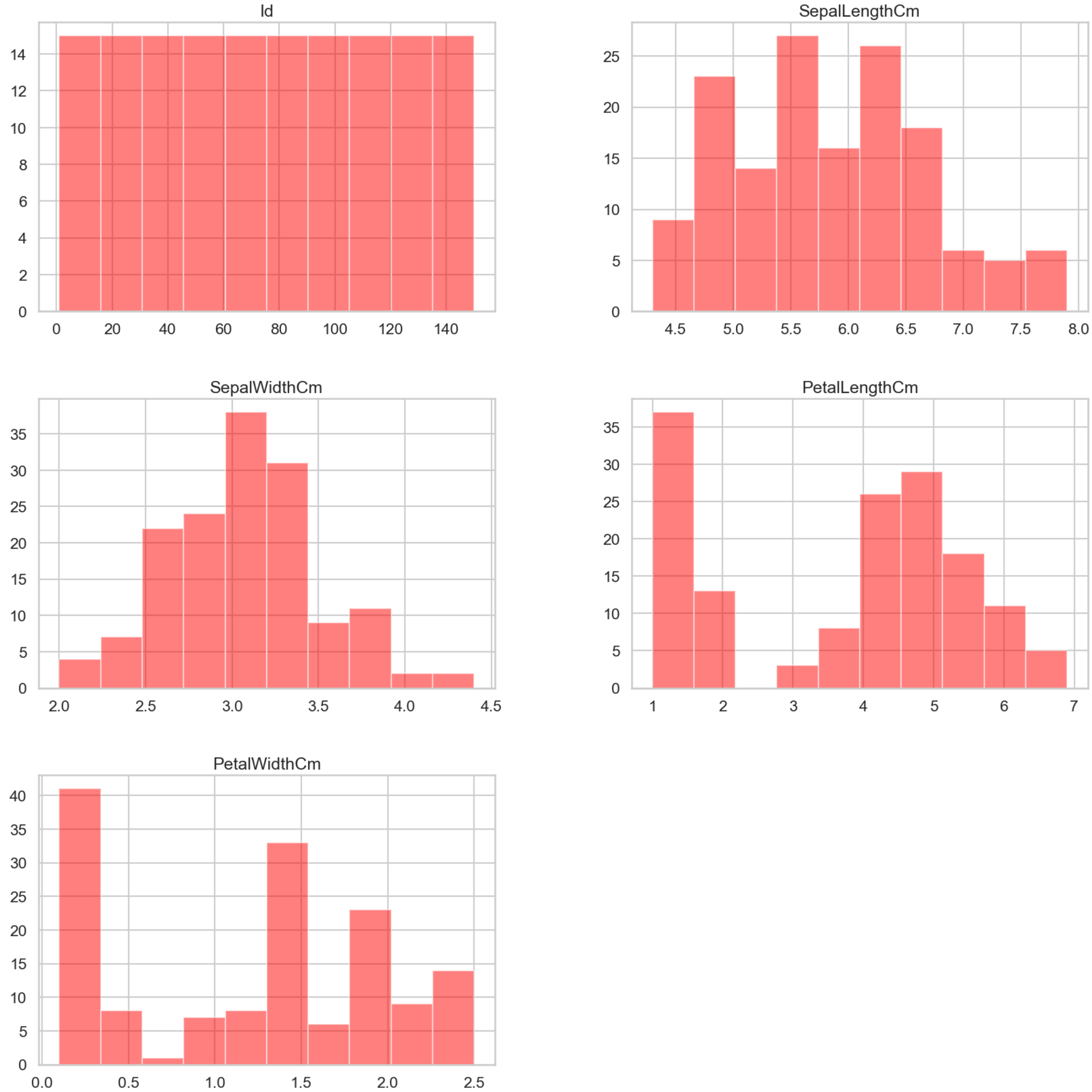
Out[4]:
```
Text(0.5, 1.0, 'Petal Sample')
```



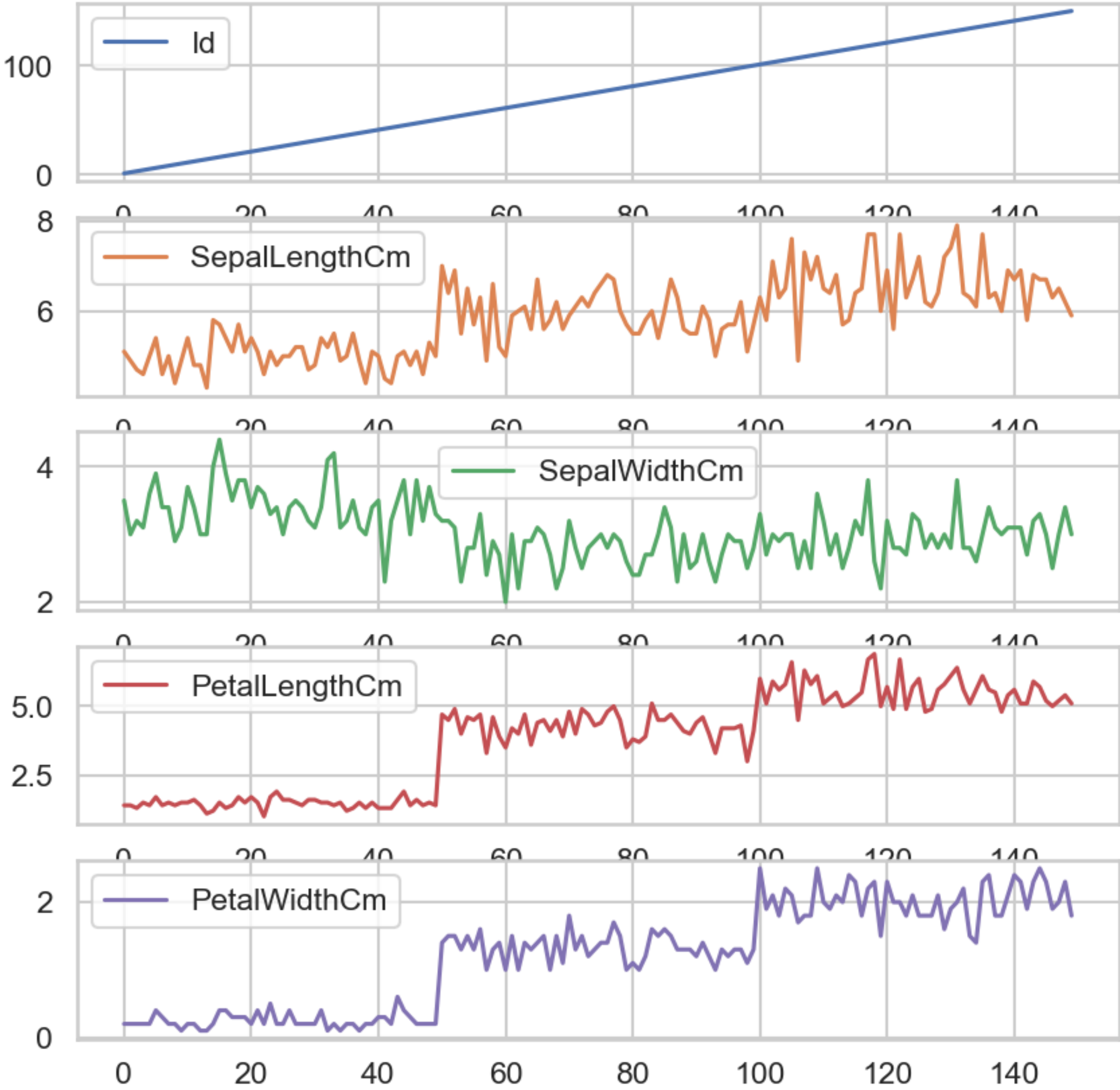
In [5]:
```
scatter_matrix(dataset, alpha=0.5, figsize=(20, 20))
plt.show()
```



In [6]:
```
dataset.hist(alpha=0.5, figsize=(20, 20), color='red')
plt.show()
```



In [7]:
```
dataset.plot(subplots=True, figsize=(10, 10), sharex=False, sharey=False)
plt.show()
```



Manually separating our dataset

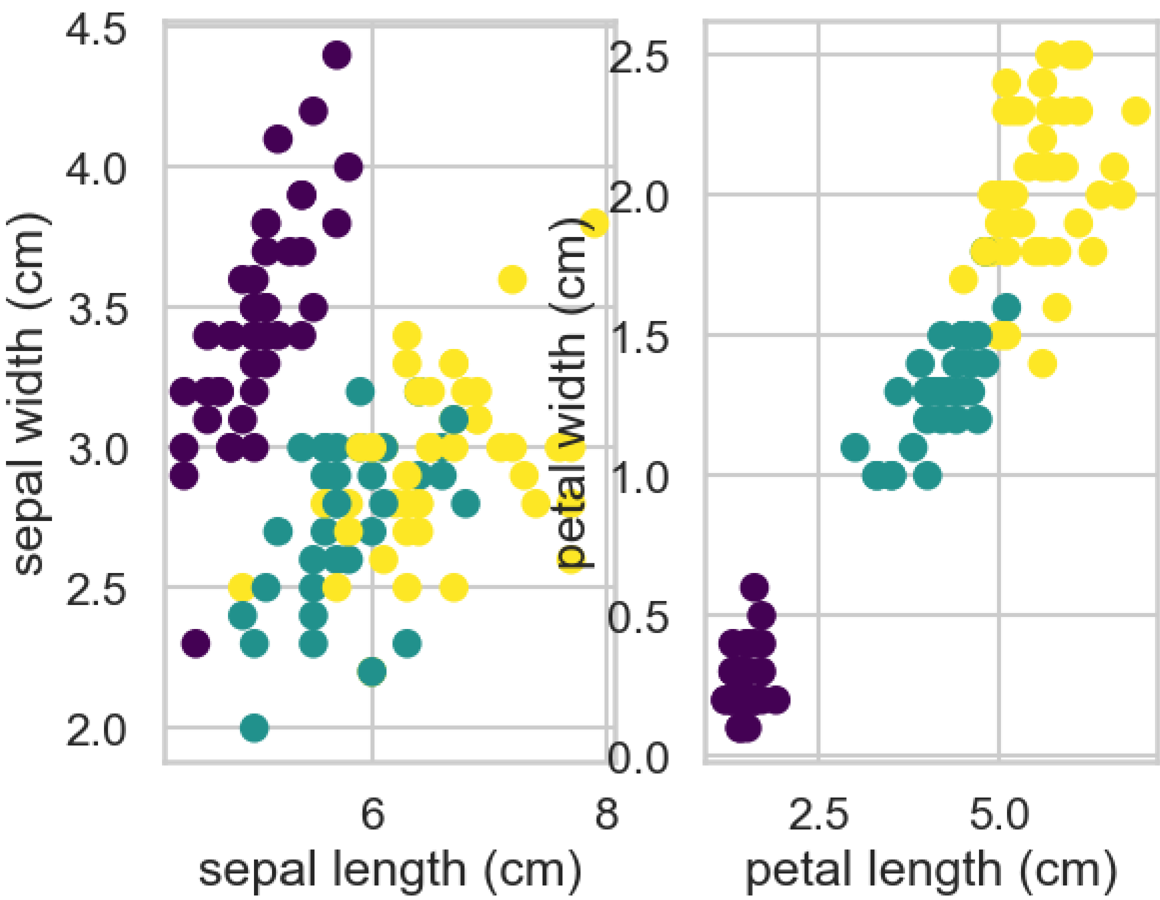
In [8]:
```
random.seed(123)

def separate_data():
    A = iris_dataset[0:40]
    tA = iris_dataset[40:50]
    B = iris_dataset[50:90]
    tB = iris_dataset[90:100]
    C = iris_dataset[100:140]
    tC = iris_dataset[140:150]
    train = np.concatenate((A,B,C))
    test =  np.concatenate((tA,tB,tC))
    return train,test

train_porcent = 80 # Porcent Training
test_porcent = 20 # Porcent Test
iris_dataset = np.column_stack((iris_data.data,iris_data.target.T)) #Join X and Y
iris_dataset = list(iris_dataset)
random.shuffle(iris_dataset)

Filetrain, Filetest = separate_data()

train_X = np.array([i[:4] for i in Filetrain])
train_y = np.array([i[4] for i in Filetrain])
```

```
test_X = np.array([i[:4] for i in Filetest])
test_y = np.array([i[4] for i in Filetest])
```

Plot our training Samples

In [9]:
```
import matplotlib.pyplot as plt
import matplotlib.cm as cm


plt.subplot(1, 2, 1)
plt.scatter(train_X[:,0],train_X[:,1],c=train_y,cmap=cm.viridis)
plt.xlabel(iris_data.feature_names[0])
plt.ylabel(iris_data.feature_names[1])

plt.subplot(1, 2, 2)
plt.scatter(train_X[:,2],train_X[:,3],c=train_y,cmap=cm.viridis)
plt.xlabel(iris_data.feature_names[2])
plt.ylabel(iris_data.feature_names[3])
```

Out[9]: Text(0, 0.5, 'petal width (cm)')

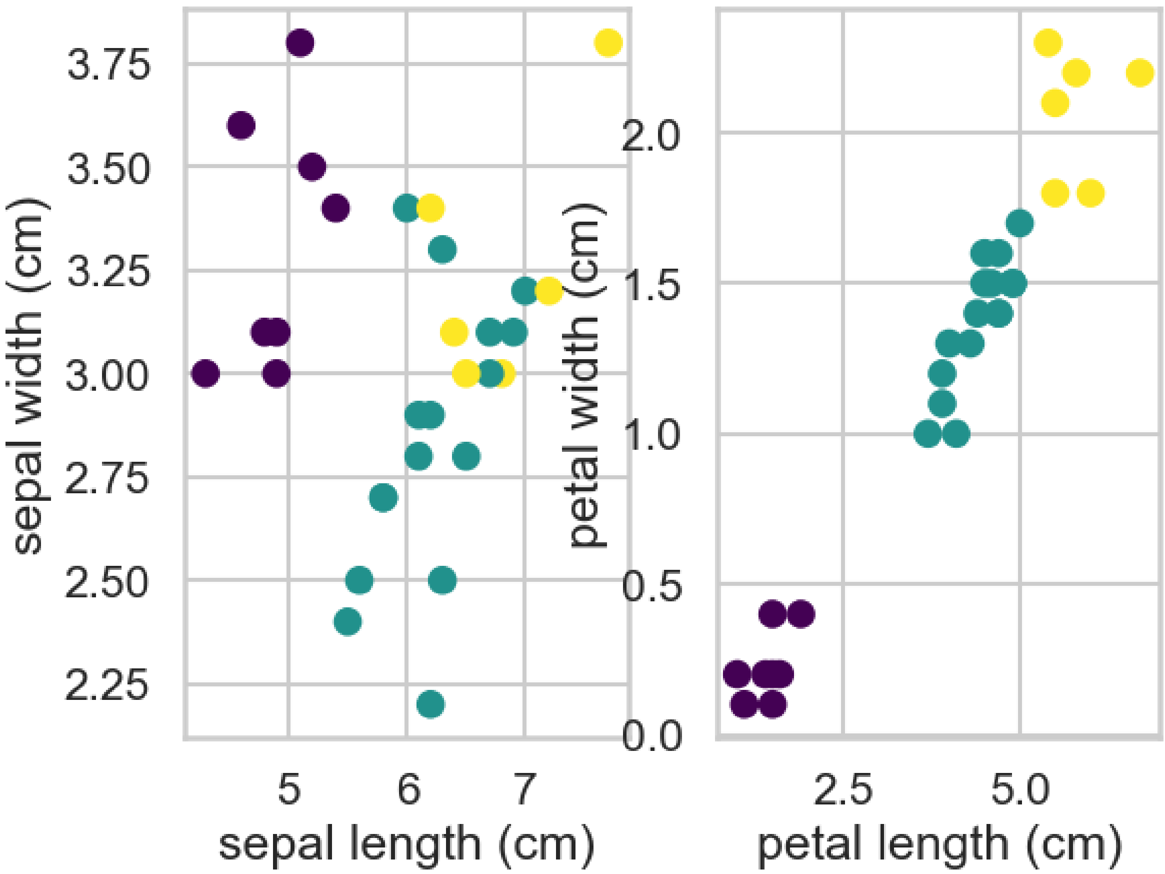


Plot our test Samples

In [10]:
```
plt.subplot(1, 2, 1)
plt.scatter(test_X[:,0],test_X[:,1],c=test_y,cmap=cm.viridis)
plt.xlabel(iris_data.feature_names[0])
plt.ylabel(iris_data.feature_names[1])

plt.subplot(1, 2, 2)
plt.scatter(test_X[:,2],test_X[:,3],c=test_y,cmap=cm.viridis)
```

```
plt.xlabel(iris_data.feature_names[2])
plt.ylabel(iris_data.feature_names[3])
```

Out[10]: Text(0, 0.5, 'petal width (cm)')



Implementation the Multilayer Perceptron in Python

In [11]:
```
from sklearn.base import BaseEstimator, ClassifierMixin, RegressorMixin
import random

class MultiLayerPerceptron(BaseEstimator, ClassifierMixin):
    def __init__(self, params=None):
        if (params == None):
            self.inputLayer = 4                        # Input Layer
            self.hiddenLayer = 5                       # Hidden Layer
            self.outputLayer = 3                       # Outpuy Layer
            self.learningRate = 0.005                  # Learning rate
            self.max_epochs = 600                      # Epochs
            self.iasHiddenValue = -1                   # Bias HiddenLayer
            self.BiasOutputValue = -1                  # Bias OutputLayer
            self.activation = self.ativacao['sigmoid'] # Activation function
            self.deriv = self.derivada['sigmoid']
        else:
            self.inputLayer = params['InputLayer']
            self.hiddenLayer = params['HiddenLayer']
            self.OutputLayer = params['OutputLayer']
            self.learningRate = params['LearningRate']
            self.max_epochs = params['Epocas']
            self.BiasHiddenValue = params['BiasHiddenValue']
            self.BiasOutputValue = params['BiasOutputValue']
            self.activation = self.ativacao[params['ActivationFunction']]
            self.deriv = self.derivada[params['ActivationFunction']]
```

```
        'Starting Bias and Weights'
        self.WEIGHT_hidden = self.starting_weights(self.hiddenLayer, self.inputLaye
        self.WEIGHT_output = self.starting_weights(self.OutputLayer, self.hiddenLay
        self.BIAS_hidden = np.array([self.BiasHiddenValue for i in range(self.hidde
        self.BIAS_output = np.array([self.BiasOutputValue for i in range(self.Outpu
        self.classes_number = 3

    pass

    def starting_weights(self, x, y):
        return [[2  * random.random() - 1 for i in range(x)] for j in range(y)]

    ativacao = {
         'sigmoid': (lambda x: 1/(1 + np.exp(-x))),
            'tanh': (lambda x: np.tanh(x)),
            'Relu': (lambda x: x*(x > 0)),
               }
    derivada = {
         'sigmoid': (lambda x: x*(1-x)),
            'tanh': (lambda x: 1-x**2),
            'Relu': (lambda x: 1 * (x>0))
               }

    def Backpropagation_Algorithm(self, x):
        DELTA_output = []
        'Stage 1 - Error: OutputLayer'
        ERROR_output = self.output - self.OUTPUT_L2
        DELTA_output = ((-1)*(ERROR_output) * self.deriv(self.OUTPUT_L2))

        arrayStore = []
        'Stage 2 - Update weights OutputLayer and HiddenLayer'
        for i in range(self.hiddenLayer):
            for j in range(self.OutputLayer):
                self.WEIGHT_output[i][j] -= (self.learningRate * (DELTA_output[j] *
                self.BIAS_output[j] -= (self.learningRate * DELTA_output[j])

        'Stage 3 - Error: HiddenLayer'
        delta_hidden = np.matmul(self.WEIGHT_output, DELTA_output)* self.deriv(self

        'Stage 4 - Update weights HiddenLayer and InputLayer(x)'
        for i in range(self.OutputLayer):
            for j in range(self.hiddenLayer):
                self.WEIGHT_hidden[i][j] -= (self.learningRate * (delta_hidden[j] *
                self.BIAS_hidden[j] -= (self.learningRate * delta_hidden[j])

    def show_err_graphic(self,v_erro,v_epoca):
        plt.figure(figsize=(9,4))
        plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
        plt.xlabel("Number of Epochs")
        plt.ylabel("Squared error (MSE) ")
        plt.title("Error Minimization")
        plt.show()

    def predict(self, X, y):
        'Returns the predictions for every element of X'
        my_predictions = []
        'Forward Propagation'
        forward = np.matmul(X,self.WEIGHT_hidden) + self.BIAS_hidden
        forward = np.matmul(forward, self.WEIGHT_output) + self.BIAS_output

        for i in forward:
```

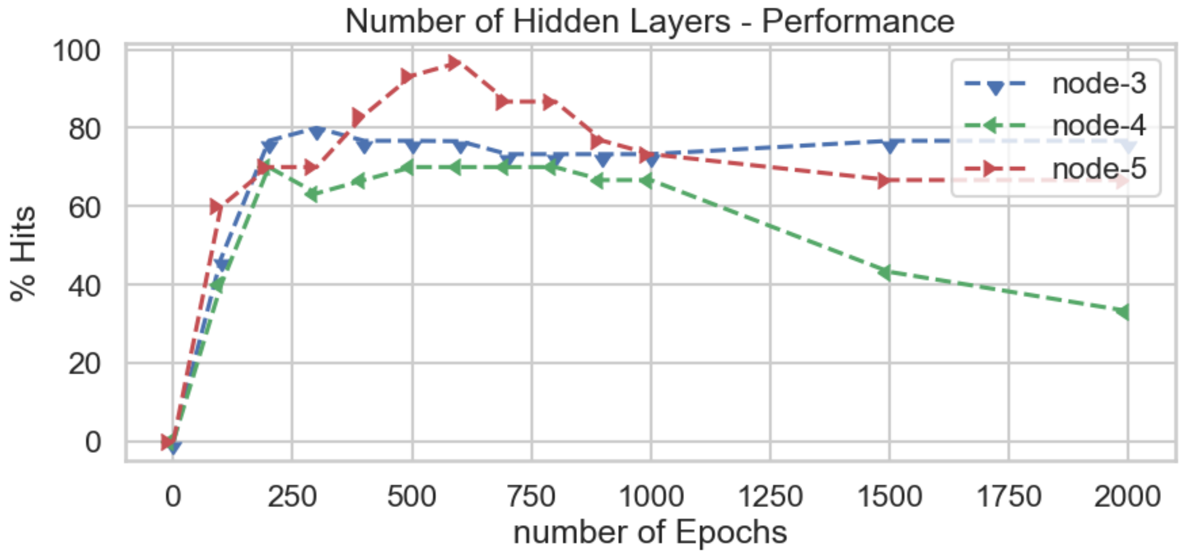
```
            my_predictions.append(max(enumerate(i), key=lambda x:x[1])[0])

        array_score = []
        for i in range(len(my_predictions)):
            if my_predictions[i] == 0:
                array_score.append([i, 'Iris-setosa', my_predictions[i], y[i]])
            elif my_predictions[i] == 1:
                array_score.append([i, 'Iris-versicolour', my_predictions[i], y[i]
            elif my_predictions[i] == 2:
                array_score.append([i, 'Iris-virginica', my_predictions[i], y[i]])

        dataframe = pd.DataFrame(array_score, columns=['_id', 'class', 'output', 'h
        return my_predictions, dataframe

    def fit(self, X, y):
        count_epoch = 1
        total_error = 0
        n = len(X);
        epoch_array = []
        error_array = []
        W0 = []
        W1 = []
        while(count_epoch <= self.max_epochs):
            for idx,inputs in enumerate(X):
                self.output = np.zeros(self.classes_number)
                'Stage 1 - (Forward Propagation)'
                self.OUTPUT_L1 = self.activation((np.dot(inputs, self.WEIGHT_hidden
                self.OUTPUT_L2 = self.activation((np.dot(self.OUTPUT_L1, self.WEIGH
                'Stage 2 - One-Hot-Encoding'
                if(y[idx] == 0):
                    self.output = np.array([1,0,0]) #Class1 {1,0,0}
                elif(y[idx] == 1):
                    self.output = np.array([0,1,0]) #Class2 {0,1,0}
                elif(y[idx] == 2):
                    self.output = np.array([0,0,1]) #Class3 {0,0,1}

                square_error = 0
                for i in range(self.OutputLayer):
                    erro = (self.output[i] - self.OUTPUT_L2[i])**2
                    square_error = (square_error + (0.05 * erro))
                    total_error = total_error + square_error

                'Backpropagation : Update Weights'
                self.Backpropagation_Algorithm(inputs)

            total_error = (total_error / n)
            if((count_epoch % 50 == 0)or(count_epoch == 1)):
                print("Epoch ", count_epoch, "- Total Error: ",total_error)
                error_array.append(total_error)
                epoch_array.append(count_epoch)

            W0.append(self.WEIGHT_hidden)
            W1.append(self.WEIGHT_output)


            count_epoch += 1
        self.show_err_graphic(error_array,epoch_array)

        plt.plot(W0[0])
        plt.title('Weight Hidden update during training')
        plt.legend(['neuron1', 'neuron2', 'neuron3', 'neuron4', 'neuron5'])
```

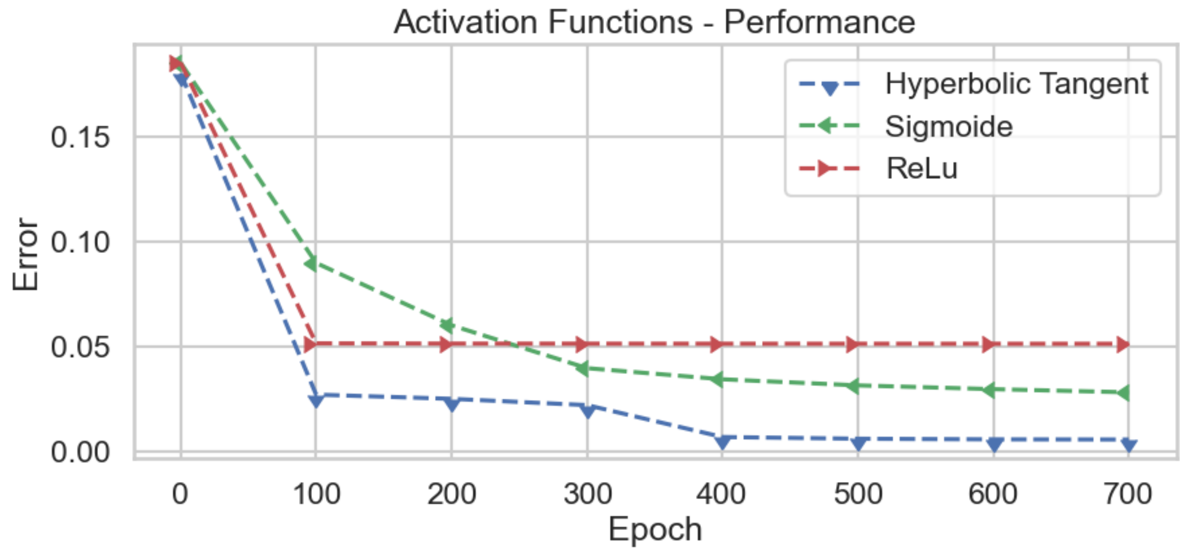
```
        plt.ylabel('Value Weight')
        plt.show()

        plt.plot(W1[0])
        plt.title('Weight Output update during training')
        plt.legend(['neuron1', 'neuron2', 'neuron3'])
        plt.ylabel('Value Weight')
        plt.show()
```

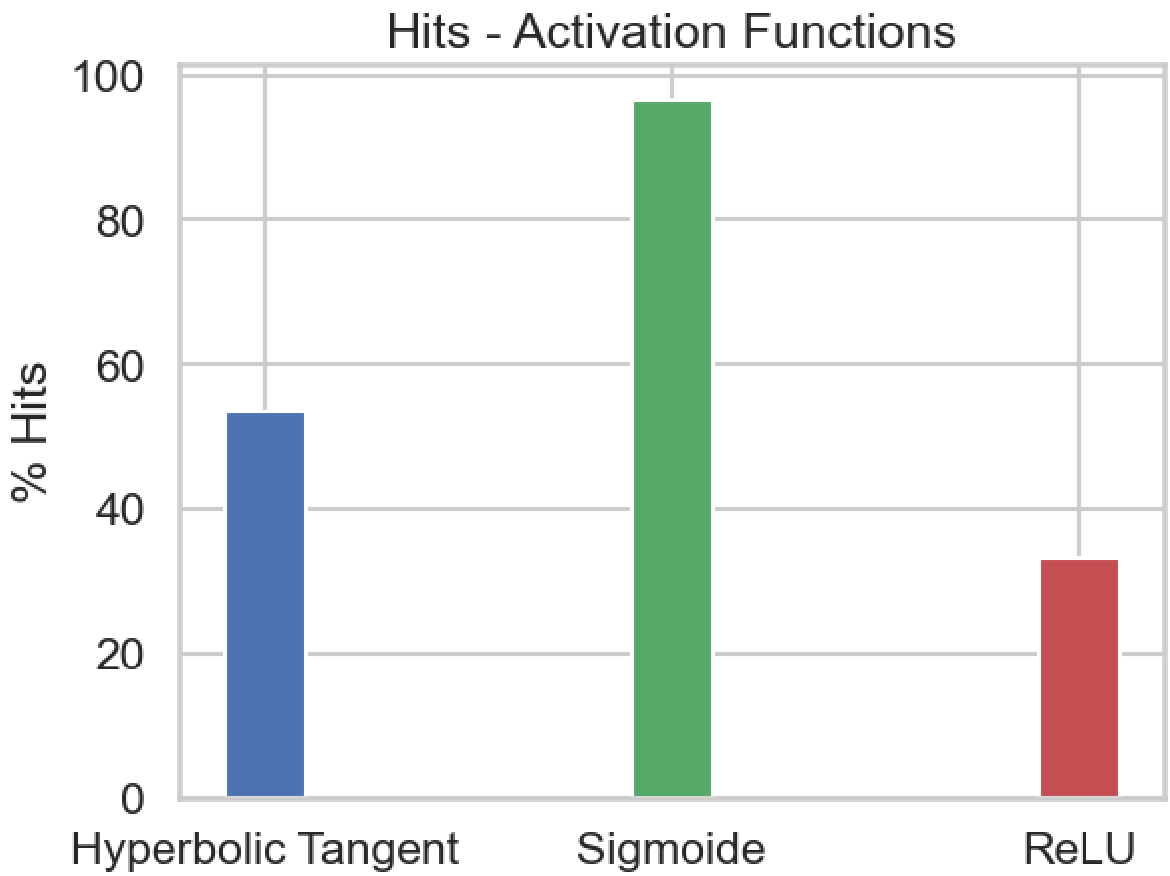
Finding the best parameters

In [12]:
```
def show_test():
    ep1 = [0,100,200,300,400,500,600,700,800,900,1000,1500,2000]
    h_5 = [0,60,70,70,83.3,93.3,96.7,86.7,86.7,76.7,73.3,66.7,66.7]
    h_4 = [0,40,70,63.3,66.7,70,70,70,70,66.7,66.7,43.3,33.3]
    h_3 = [0,46.7,76.7,80,76.7,76.7,76.6,73.3,73.3,73.3,73.3,76.7,76.7]
    plt.figure(figsize=(10,4))
    l1, = plt.plot(ep1, h_3, "--",color='b',label="node-3", marker=11)
    l2, = plt.plot(ep1, h_4, "--",color='g',label="node-4", marker=8)
    l3, = plt.plot(ep1, h_5, "--",color='r',label="node-5", marker=5)
    plt.legend(handles=[l1,l2,l3], loc=1)
    plt.xlabel("number of Epochs")
    plt.ylabel("% Hits")
    plt.title("Number of Hidden Layers - Performance")

    ep2 = [0,100,200,300,400,500,600,700]
    tanh = [0.18,0.027,0.025,0.022,0.0068,0.0060,0.0057,0.00561]
    sigm = [0.185,0.0897,0.060,0.0396,0.0343,0.0314,0.0296,0.0281]
    Relu = [0.185,0.05141,0.05130,0.05127,0.05124,0.05123,0.05122,0.05121]
    plt.figure(figsize=(10,4))
    l1 , = plt.plot(ep2, tanh, "--",color='b',label="Hyperbolic Tangent",marker=11)
    l2 , = plt.plot(ep2, sigm, "--",color='g',label="Sigmoide", marker=8)
    l3 , = plt.plot(ep2, Relu, "--",color='r',label="ReLu", marker=5)
    plt.legend(handles=[l1,l2,l3], loc=1)
    plt.xlabel("Epoch")
    plt.ylabel("Error")
    plt.title("Activation Functions - Performance")

    fig, ax = plt.subplots()
    names = ["Hyperbolic Tangent","Sigmoide","ReLU"]
    x1 = [2.0,4.0,6.0]
    plt.bar(x1[0], 53.4,0.4,color='b')
    plt.bar(x1[1], 96.7,0.4,color='g')
    plt.bar(x1[2], 33.2,0.4,color='r')
    plt.xticks(x1,names)
    plt.ylabel('% Hits')
    plt.title('Hits - Activation Functions')
    plt.show()
```

In [13]:
```
show_test()
```


Number of Hidden Layers - Performance
% Hits
number of Epochs
node-3
node-4
node-5
0
20
40
60
80
100
0
250
500
750
1000
1250
1500
1750
2000


Activation Functions - Performance
Error
Epoch
Hyperbolic Tangent
Sigmoide
ReLu
0.00
0.05
0.10
0.15
0
100
200
300
400
500
600
700


Hits - Activation Functions
% Hits
0
20
40
60
80
100
Hyperbolic Tangent
Sigmoide
ReLU

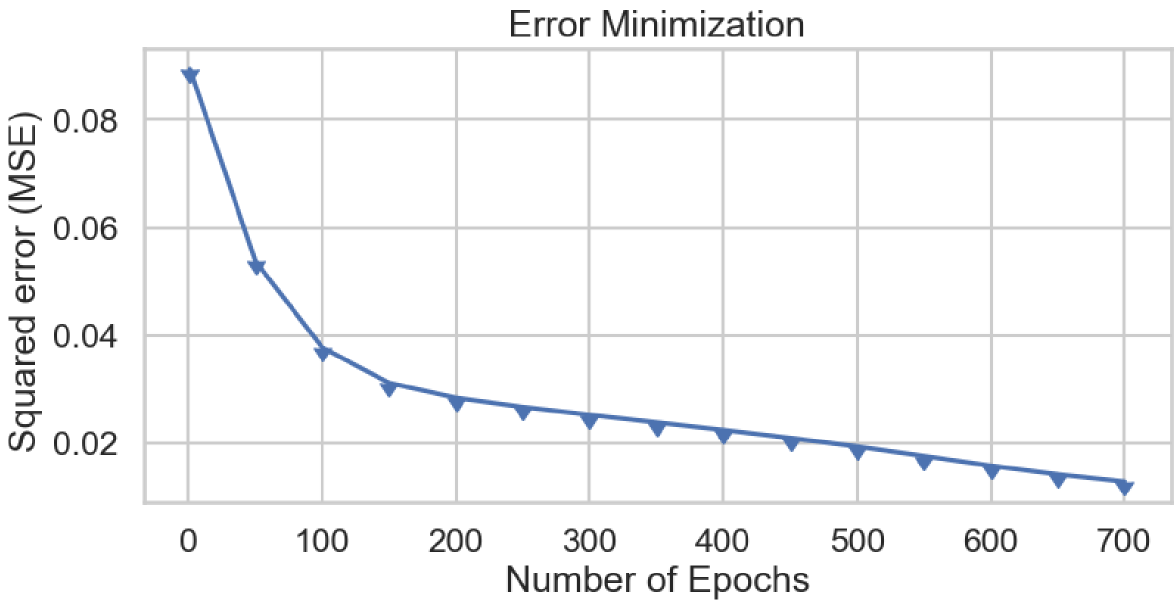
Training the Artificial Neural Network(MLP)

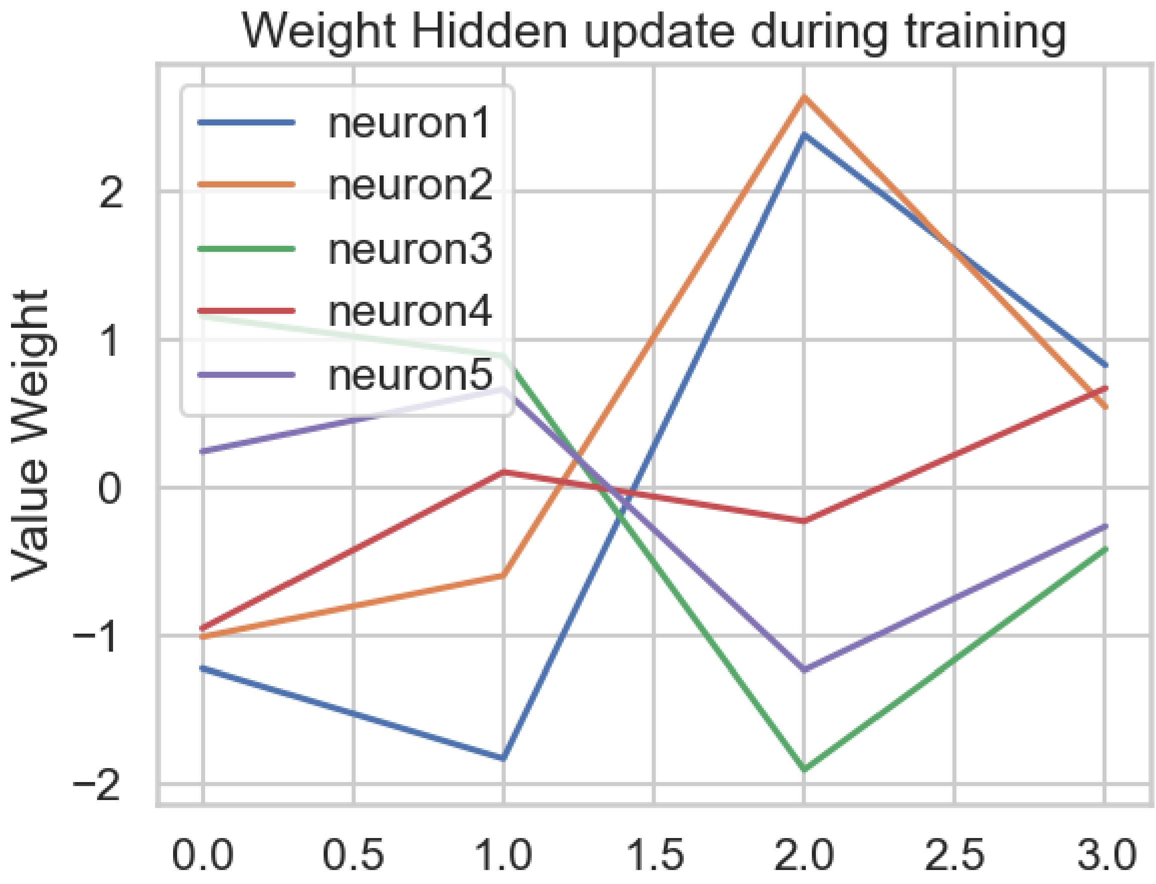
Step 1: training our MultiLayer Perceptron using sigmoid

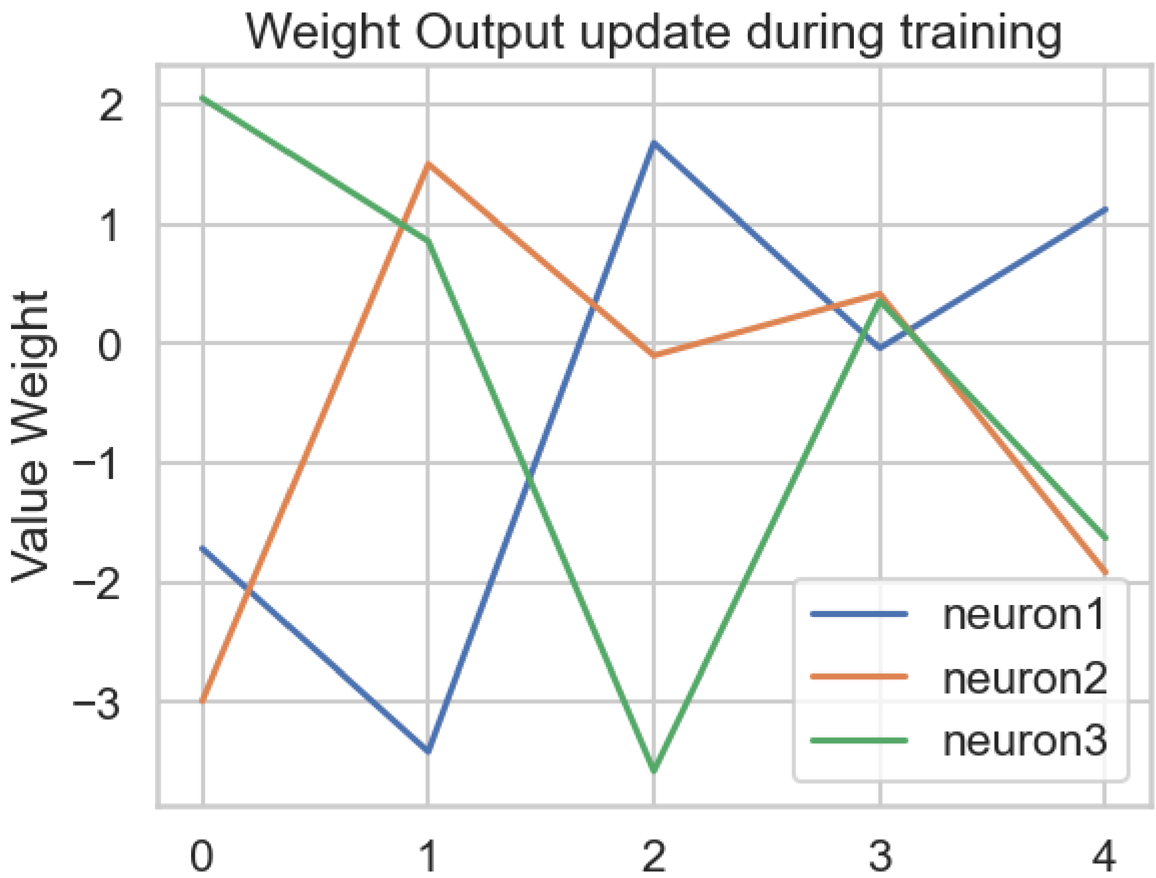
In [14]:
```
dictionary = {'InputLayer':4, 'HiddenLayer':5, 'OutputLayer':3,
              'Epocas':700, 'LearningRate':0.005,'BiasHiddenValue':-1,
              'BiasOutputValue':-1, 'ActivationFunction':'sigmoid'}

Perceptron = MultiLayerPerceptron(dictionary)
Perceptron.fit(train_X,train_y)
```

```
Epoch  1 - Total Error:  0.08939914265311076
Epoch  50 - Total Error:  0.05376852115527734
Epoch  100 - Total Error:  0.03773105860576402
Epoch  150 - Total Error:  0.0311644504972821
Epoch  200 - Total Error:  0.028406909135682057
Epoch  250 - Total Error:  0.02669256194214157
Epoch  300 - Total Error:  0.0252400340593952
Epoch  350 - Total Error:  0.02384525367109565
Epoch  400 - Total Error:  0.02242746475447137
Epoch  450 - Total Error:  0.020948415886624147
Epoch  500 - Total Error:  0.01936513137210057
Epoch  550 - Total Error:  0.017597424493928975
Epoch  600 - Total Error:  0.015792170595649142
Epoch  650 - Total Error:  0.014194648016335605
Epoch  700 - Total Error:  0.012855583329293764
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11540\1195911340.py:74: UserWarning: co
lor is redundantly defined by the 'color' keyword argument and the fmt string "m-"
(-> color='m'). The keyword argument will take precedence.
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```







Step 2: testing our results

In [15]:
```
prev, dataframe = Perceptron.predict(test_X, test_y)
hits = n_set = n_vers = n_virg = 0
score_set = score_vers = score_virg = 0
```

```
for j in range(len(test_y)):
    if(test_y[j] == 0): n_set += 1
    elif(test_y[j] == 1): n_vers += 1
    elif(test_y[j] == 2): n_virg += 1

for i in range(len(test_y)):
    if test_y[i] == prev[i]:
        hits += 1
    if test_y[i] == prev[i] and test_y[i] == 0:
        score_set += 1
    elif test_y[i] == prev[i] and test_y[i] == 1:
        score_vers += 1
    elif test_y[i] == prev[i] and test_y[i] == 2:
        score_virg += 1

hits = (hits / len(test_y)) * 100
faults = 100 - hits
```

In [16]:
```
dataframe
```

Out[16]:

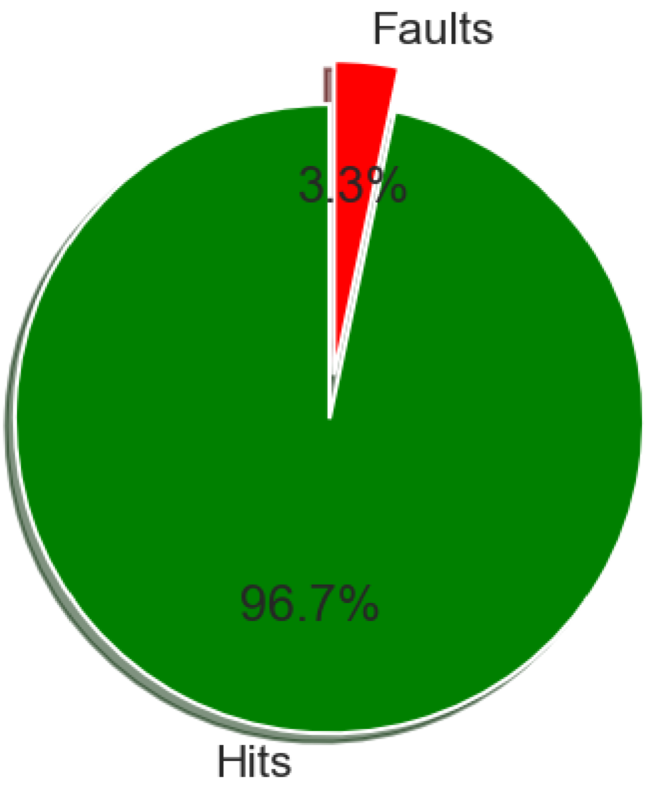
	_id	class	output	hoped_output
0	0	Iris-setosa	0	0.0
1	1	Iris-versicolour	1	1.0
2	2	Iris-versicolour	1	1.0
3	3	Iris-setosa	0	0.0
4	4	Iris-versicolour	1	1.0
5	5	Iris-versicolour	1	1.0
6	6	Iris-versicolour	1	1.0
7	7	Iris-versicolour	1	1.0
8	8	Iris-setosa	0	0.0
9	9	Iris-virginica	2	2.0
10	10	Iris-virginica	2	2.0
11	11	Iris-versicolour	1	1.0
12	12	Iris-versicolour	1	1.0
13	13	Iris-versicolour	1	1.0
14	14	Iris-setosa	0	0.0
15	15	Iris-virginica	2	2.0
16	16	Iris-virginica	2	1.0
17	17	Iris-versicolour	1	1.0
18	18	Iris-versicolour	1	1.0
19	19	Iris-versicolour	1	1.0
20	20	Iris-versicolour	1	1.0
21	21	Iris-virginica	2	2.0
22	22	Iris-versicolour	1	1.0
23	23	Iris-setosa	0	0.0
24	24	Iris-setosa	0	0.0
25	25	Iris-virginica	2	2.0
26	26	Iris-virginica	2	2.0
27	27	Iris-setosa	0	0.0
28	28	Iris-versicolour	1	1.0
29	29	Iris-setosa	0	0.0

Step 3. Accuracy and precision the Multilayer Perceptron

In [17]:
```
graph_hits = []
print("Porcents :","%.2f"%(hits),"% hits","and","%.2f"%(faults),"% faults")
print("Total samples of test",n_samples)
print("*Iris-Setosa:",n_set,"samples")
print("*Iris-Versicolour:",n_vers,"samples")
```

```
print("*Iris-Virginica:",n_virg,"samples")

graph_hits.append(hits)
graph_hits.append(faults)
labels = 'Hits', 'Faults';
sizes = [96.5, 3.3]
explode = (0, 0.14)

fig1, ax1 = plt.subplots();
ax1.pie(graph_hits, explode=explode,colors=['green','red'],labels=labels, autopct='
shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```

```
Porcents : 96.67 % hits and 3.33 % faults
Total samples of test 150
*Iris-Setosa: 8 samples
*Iris-Versicolour: 16 samples
*Iris-Virginica: 6 samples
```

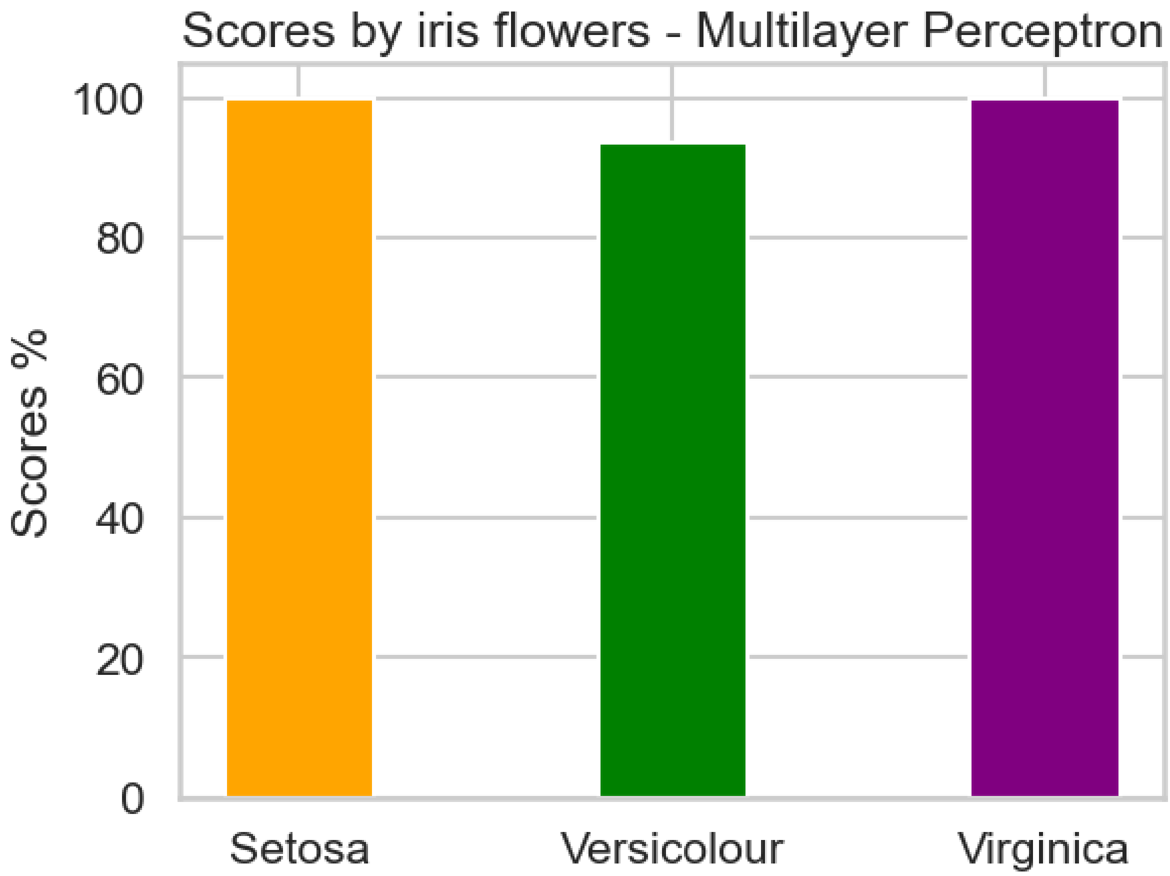


Step 4. Score for each one of the samples

In [18]:
```
acc_set = (score_set/n_set)*100
acc_vers = (score_vers/n_vers)*100
acc_virg = (score_virg/n_virg)*100
print("- Acurracy Iris-Setosa:","%.2f"%acc_set, "%")
print("- Acurracy Iris-Versicolour:","%.2f"%acc_vers, "%")
print("- Acurracy Iris-Virginica:","%.2f"%acc_virg, "%")
names = ["Setosa","Versicolour","Virginica"]
x1 = [2.0,4.0,6.0]
fig, ax = plt.subplots()
r1 = plt.bar(x1[0], acc_set,color='orange',label='Iris-Setosa')
r2 = plt.bar(x1[1], acc_vers,color='green',label='Iris-Versicolour')
r3 = plt.bar(x1[2], acc_virg,color='purple',label='Iris-Virginica')
plt.ylabel('Scores %')
```

```
plt.xticks(x1, names);plt.title('Scores by iris flowers - Multilayer Perceptron')
plt.show()
```

```
- Acurracy Iris-Setosa: 100.00 %
- Acurracy Iris-Versicolour: 93.75 %
- Acurracy Iris-Virginica: 100.00 %
```

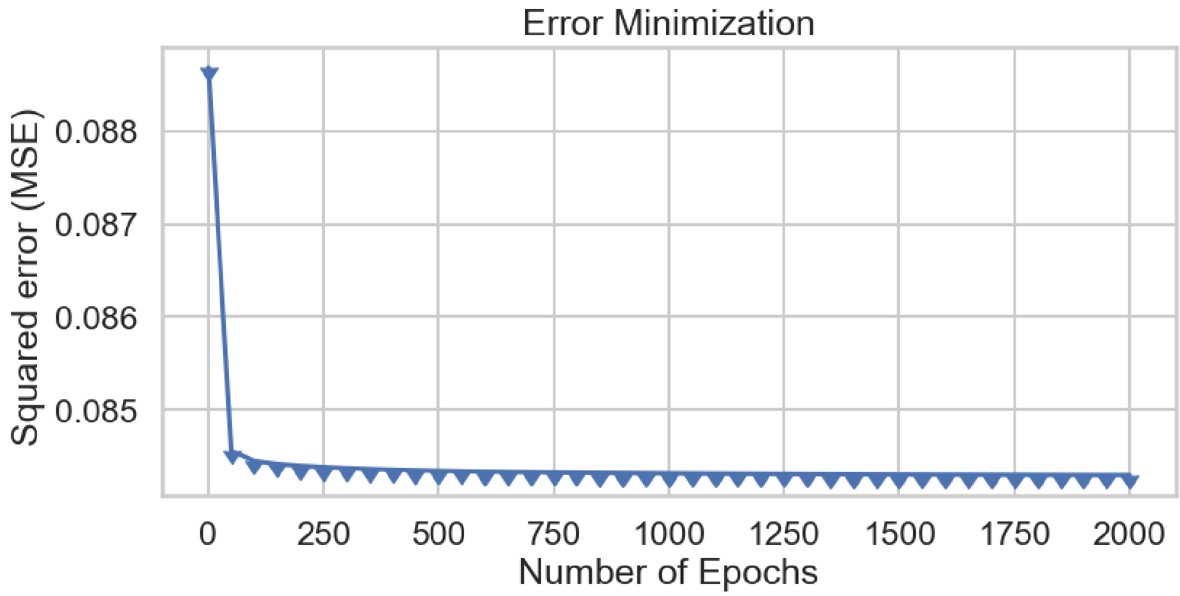


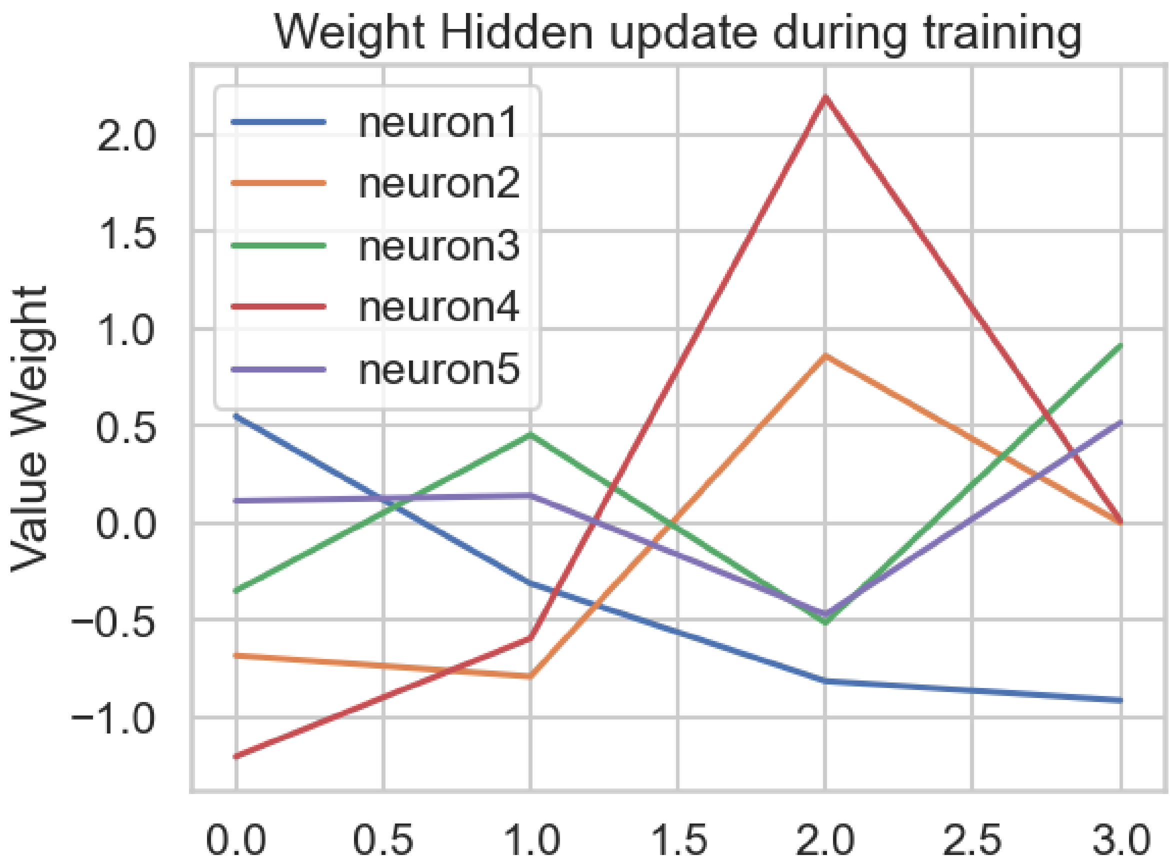
Step 1: training our MultiLayer Perceptron using relu

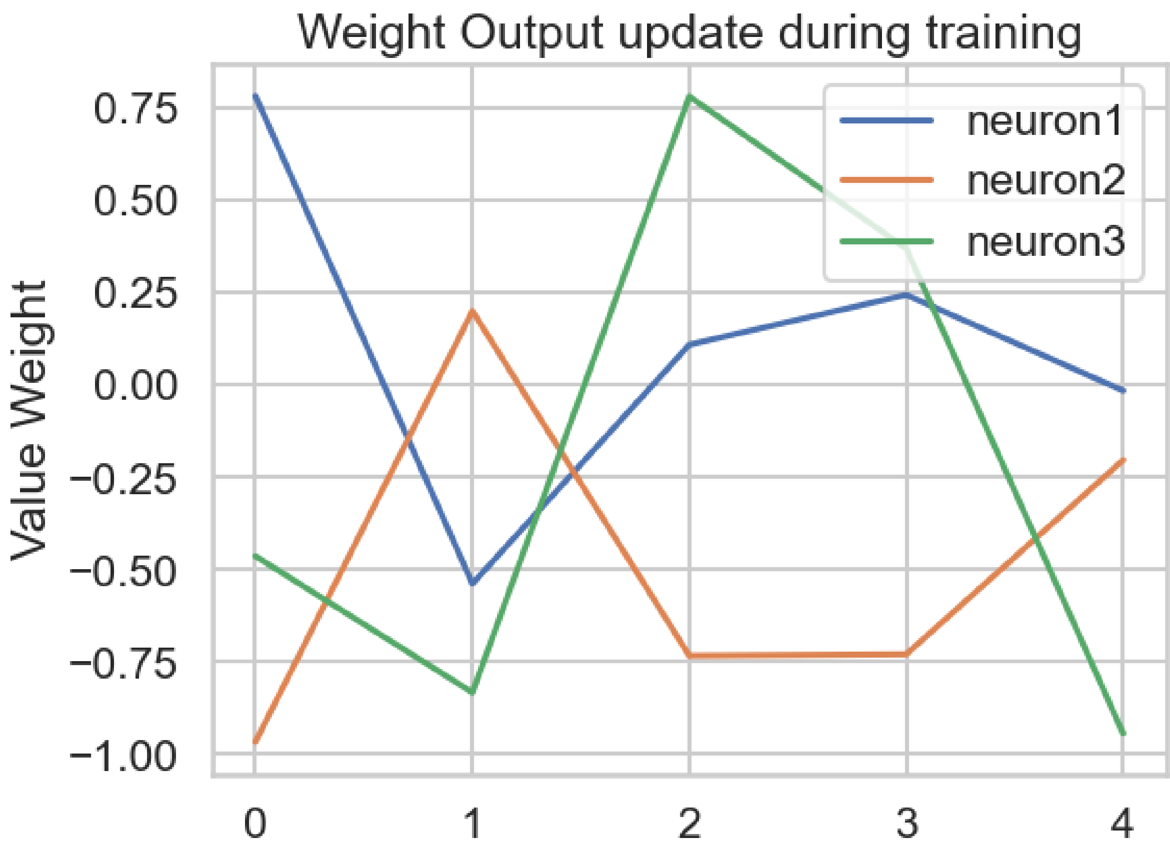
In [19]:
```
dictionary = {'InputLayer':4, 'HiddenLayer':5, 'OutputLayer':3,
              'Epocas':2000, 'LearningRate':0.005,'BiasHiddenValue':-1,
              'BiasOutputValue':-1, 'ActivationFunction':'Relu'}

Perceptron = MultiLayerPerceptron(dictionary)
Perceptron.fit(train_X,train_y)
```

```
Epoch  1 - Total Error:  0.08867982775797653
Epoch  50 - Total Error:  0.084561809525899
Epoch  100 - Total Error:  0.08445302856714122
Epoch  150 - Total Error:  0.08442077071370466
Epoch  200 - Total Error:  0.08440180407784893
Epoch  250 - Total Error:  0.08438795631461381
Epoch  300 - Total Error:  0.08437663428960811
Epoch  350 - Total Error:  0.08436745097690895
Epoch  400 - Total Error:  0.08435993034052351
Epoch  450 - Total Error:  0.0843536596596564
Epoch  500 - Total Error:  0.08434833835166745
Epoch  550 - Total Error:  0.08434375412081561
Epoch  600 - Total Error:  0.08433975509379131
Epoch  650 - Total Error:  0.08433622997528133
Epoch  700 - Total Error:  0.08433309518357154
Epoch  750 - Total Error:  0.08433025975517576
Epoch  800 - Total Error:  0.08432772809440009
Epoch  850 - Total Error:  0.08432551795866207
Epoch  900 - Total Error:  0.08432349825798831
Epoch  950 - Total Error:  0.08432164413059318
Epoch  1000 - Total Error:  0.08431993583554398
Epoch  1050 - Total Error:  0.08431835690358773
Epoch  1100 - Total Error:  0.08431689341175182
Epoch  1150 - Total Error:  0.08431553348979806
Epoch  1200 - Total Error:  0.08431426694348944
Epoch  1250 - Total Error:  0.08431308496005228
Epoch  1300 - Total Error:  0.08431197987496916
Epoch  1350 - Total Error:  0.08431094498540011
Epoch  1400 - Total Error:  0.08430997439951954
Epoch  1450 - Total Error:  0.08430906291383236
Epoch  1500 - Total Error:  0.08430820591250433
Epoch  1550 - Total Error:  0.08430739928417288
Epoch  1600 - Total Error:  0.08430663935275326
Epoch  1650 - Total Error:  0.08430592281953418
Epoch  1700 - Total Error:  0.08430524671444733
Epoch  1750 - Total Error:  0.08430460835483779
Epoch  1800 - Total Error:  0.08430400531040606
Epoch  1850 - Total Error:  0.08430343537325939
Epoch  1900 - Total Error:  0.08430289653221154
Epoch  1950 - Total Error:  0.08430238695063731
Epoch  2000 - Total Error:  0.08430190494731378
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11540\1195911340.py:74: UserWarning: co
lor is redundantly defined by the 'color' keyword argument and the fmt string "m-"
(-> color='m'). The keyword argument will take precedence.
  plt.plot(v_epoca, v_erro, "m-",color="b", marker=11)
```


Error Minimization
Squared error (MSE)
0.088
0.087
0.086
0.085
0
250
500
750
1000
1250
1500
1750
2000
Number of Epochs


Weight Hidden update during training
Value Weight
neuron1
neuron2
neuron3
neuron4
neuron5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0



Step 2: testing our results

In [20]:
```
prev, dataframe = Perceptron.predict(test_X, test_y)
hits = n_set = n_vers = n_virg = 0
score_set = score_vers = score_virg = 0
for j in range(len(test_y)):
    if(test_y[j] == 0): n_set += 1
    elif(test_y[j] == 1): n_vers += 1
    elif(test_y[j] == 2): n_virg += 1

for i in range(len(test_y)):
    if test_y[i] == prev[i]:
        hits += 1
    if test_y[i] == prev[i] and test_y[i] == 0:
        score_set += 1
    elif test_y[i] == prev[i] and test_y[i] == 1:
        score_vers += 1
    elif test_y[i] == prev[i] and test_y[i] == 2:
        score_virg += 1

hits = (hits / len(test_y)) * 100
faults = 100 - hits
```

In [21]:
```
dataframe
```

Out[21]:

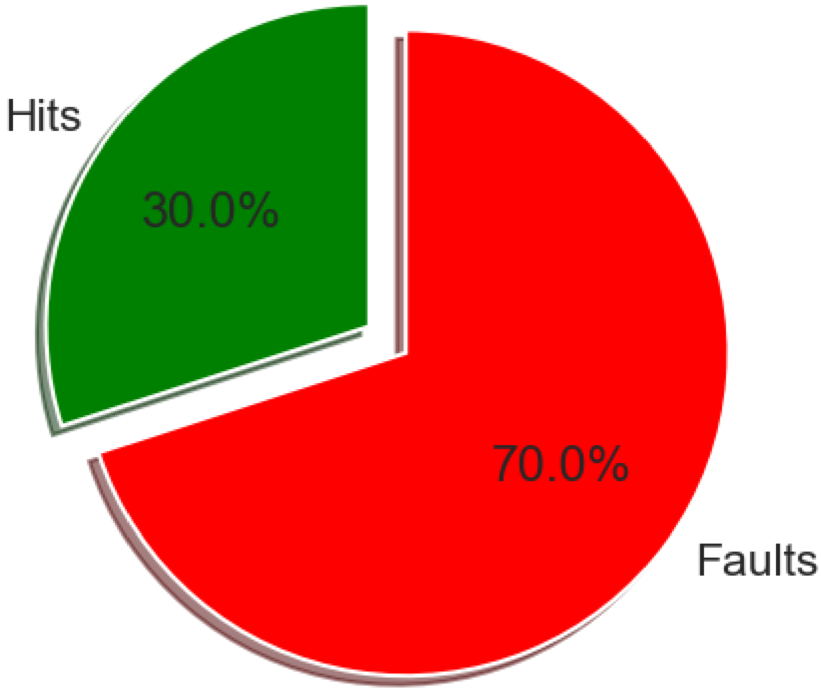
	_id	class	output	hoped_output
0	0	Iris-versicolour	1	0.0
1	1	Iris-virginica	2	1.0
2	2	Iris-virginica	2	1.0
3	3	Iris-versicolour	1	0.0
4	4	Iris-virginica	2	1.0
5	5	Iris-virginica	2	1.0
6	6	Iris-virginica	2	1.0
7	7	Iris-virginica	2	1.0
8	8	Iris-versicolour	1	0.0
9	9	Iris-virginica	2	2.0
10	10	Iris-virginica	2	2.0
11	11	Iris-versicolour	1	1.0
12	12	Iris-virginica	2	1.0
13	13	Iris-versicolour	1	1.0
14	14	Iris-versicolour	1	0.0
15	15	Iris-virginica	2	2.0
16	16	Iris-virginica	2	1.0
17	17	Iris-virginica	2	1.0
18	18	Iris-versicolour	1	1.0
19	19	Iris-virginica	2	1.0
20	20	Iris-virginica	2	1.0
21	21	Iris-virginica	2	2.0
22	22	Iris-virginica	2	1.0
23	23	Iris-versicolour	1	0.0
24	24	Iris-versicolour	1	0.0
25	25	Iris-virginica	2	2.0
26	26	Iris-virginica	2	2.0
27	27	Iris-versicolour	1	0.0
28	28	Iris-virginica	2	1.0
29	29	Iris-versicolour	1	0.0

Step 3. Accuracy and precision the Multilayer Perceptron

In [22]:
```
graph_hits = []
print("Porcents :","%.2f"%(hits),"% hits","and","%.2f"%(faults),"% faults")
print("Total samples of test",n_samples)
print("*Iris-Setosa:",n_set,"samples")
print("*Iris-Versicolour:",n_vers,"samples")
```

```
print("*Iris-Virginica:",n_virg,"samples")

graph_hits.append(hits)
graph_hits.append(faults)
labels = 'Hits', 'Faults';
sizes = [96.5, 3.3]
explode = (0, 0.14)

fig1, ax1 = plt.subplots();
ax1.pie(graph_hits, explode=explode,colors=['green','red'],labels=labels, autopct='
shadow=True, startangle=90)
ax1.axis('equal')
plt.show()
```

```
Porcents : 30.00 % hits and 70.00 % faults
Total samples of test 150
*Iris-Setosa: 8 samples
*Iris-Versicolour: 16 samples
*Iris-Virginica: 6 samples
```



Step 4. Score for each one of the samples

In [23]:
```
acc_set = (score_set/n_set)*100
acc_vers = (score_vers/n_vers)*100
acc_virg = (score_virg/n_virg)*100
print("- Acurracy Iris-Setosa:","%.2f"%acc_set, "%")
print("- Acurracy Iris-Versicolour:","%.2f"%acc_vers, "%")
print("- Acurracy Iris-Virginica:","%.2f"%acc_virg, "%")
names = ["Setosa","Versicolour","Virginica"]
x1 = [2.0,4.0,6.0]
fig, ax = plt.subplots()
r1 = plt.bar(x1[0], acc_set,color='orange',label='Iris-Setosa')
r2 = plt.bar(x1[1], acc_vers,color='green',label='Iris-Versicolour')
r3 = plt.bar(x1[2], acc_virg,color='purple',label='Iris-Virginica')
plt.ylabel('Scores %')
plt.xticks(x1, names);plt.title('Scores by iris flowers - Multilayer Perceptron')
plt.show()
```

```
- Acurracy Iris-Setosa: 0.00 %
- Acurracy Iris-Versicolour: 18.75 %
- Acurracy Iris-Virginica: 100.00 %
```