
# Lab Sessional Report

*Course Name: Fundamentals of Machine Learning*

*Course Code: CSE313*

Name: **Abhimanyu Bhowmik**

Enrollment No.: **A910119819008**

**B. Tech (AI) || 5th Sem**

Course Faculty:

**Dr Sudarshan Nandy**

Department of Computer Science and Engineering,

Amity School of Engineering and Technology,

Amity University Kolkata,

Kolkata, West Bengal, India.

# Content

# Experiment 1:

---

Write a program to extract the min, max, mean, and standard deviation of an ECG signal by considering a window over each pattern of the IRIS dataset.

### FML LAB 1

In [28]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import axes3d, Axes3D
```

### Read Data

In [29]:
```
iris = pd.read_csv('iris.data')
iris = iris.to_numpy()
iris = np.delete(iris,-1,1)
```

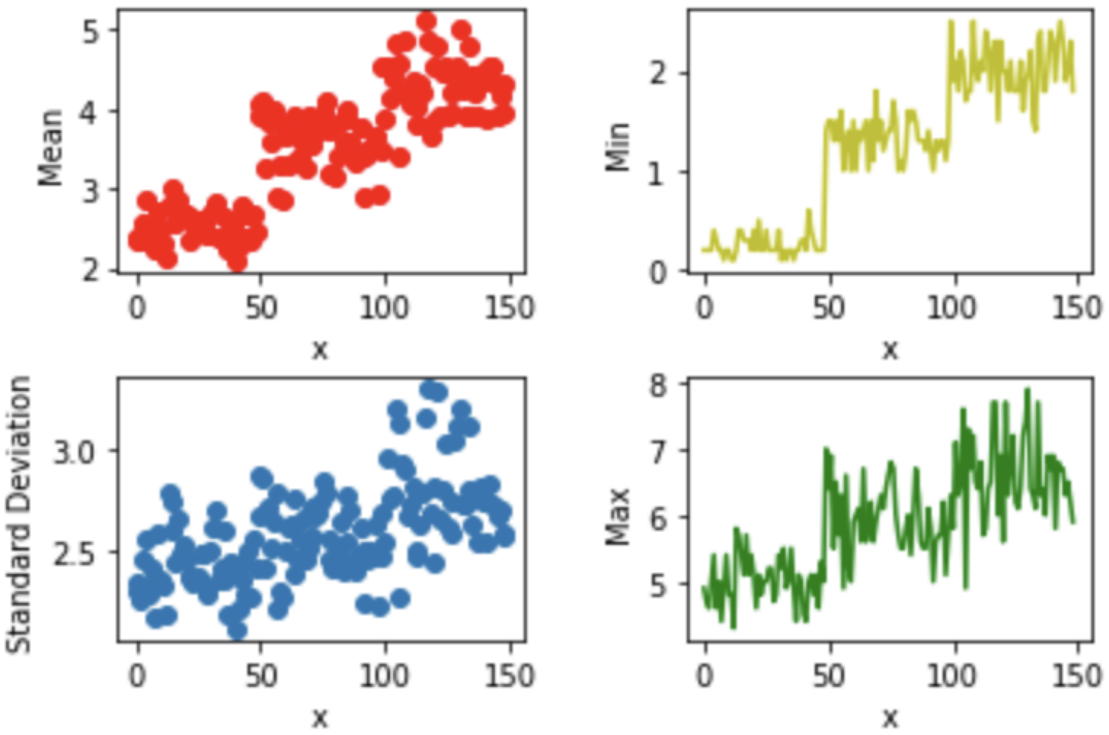
### Compute Min, Max, Mean & Std

In [30]:
```
min = []
max = []
mean = []
std = []

def minimum_function(row):
    min = row[0]
    for i in row:
        if i<min:
            min = i
    return min

def maximum_function(row):
    max = row[0]
    for i in row:
        if i>max:
            max = i
    return max

def mean_function(row):
    sum = 0
    for i in row:
        sum = sum + i
    mean = sum/len(row)
    return mean
```

```
def std_function(row):
    sum = 0
    for i in row:
        sum = sum + i
    xbar = sum/len(row)

    for x in row:
        sum = sum + np.square(x - xbar)

    std = np.sqrt(sum/len(row))
    return std


for i in iris:
    min.append(minimum_function(i))

for i in iris:
    max.append(maximum_function(i))

for i in iris:
    mean.append(mean_function(i))

for i in iris:
    std.append(std_function(i))
```

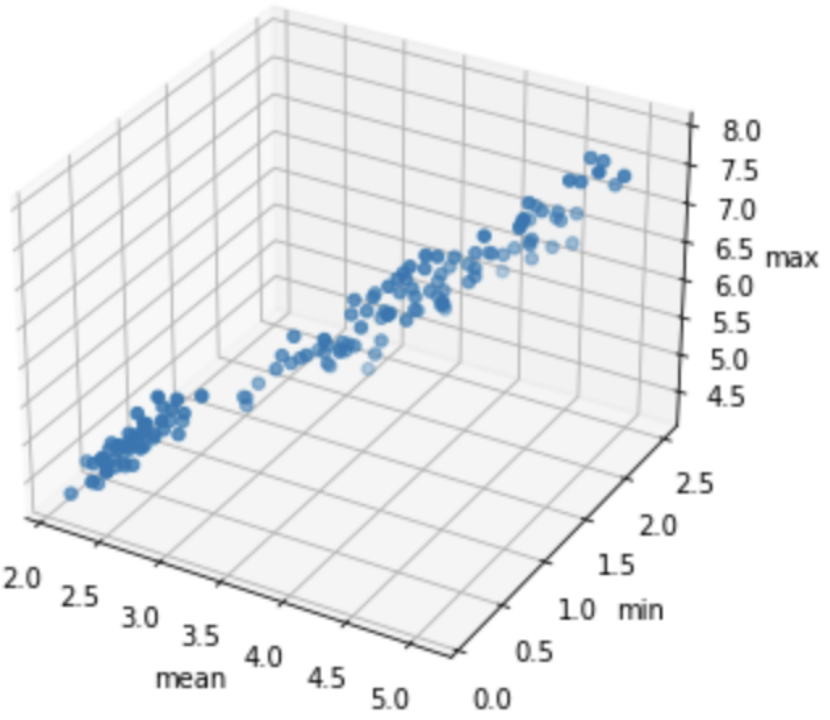
### 2D Plot

In [31]:
```
plt.subplot(2,2,1)
x = np.arange(0,len(mean))
plt.scatter(x,mean,c = 'r')
plt.xlabel('x')
plt.ylabel('Mean')
plt.subplot(2,2,2)
plt.plot(x,min, c = 'y')
plt.xlabel('x')
plt.ylabel('Min')
plt.subplot(2,2,3)
x = np.arange(0,len(std))
plt.scatter(x,std)
plt.xlabel('x')
plt.ylabel('Standard Deviation')
plt.subplot(2,2,4)
plt.plot(x,max, c ='g')
plt.xlabel('x')
plt.ylabel('Max')
plt.subplots_adjust(wspace=0.4,
                    hspace=0.4)
plt.show()
```

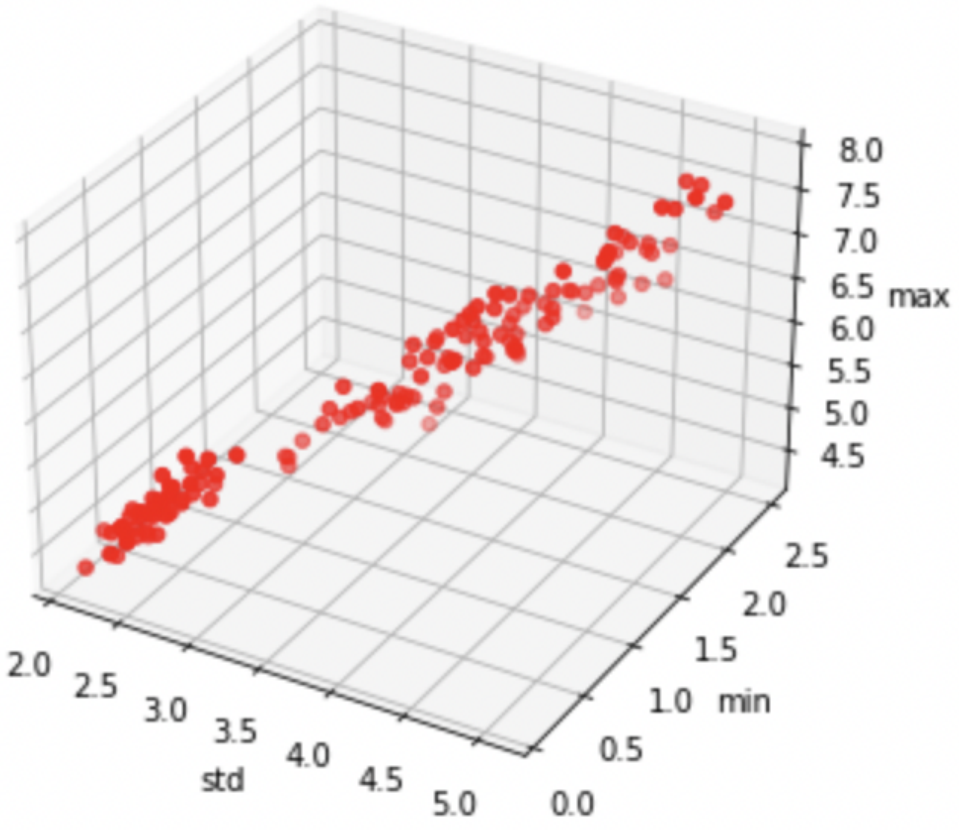



### 3D Plot

In [32]:
```
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(mean, min, max)
ax.set_xlabel('mean')
ax.set_ylabel('min')
ax.set_zlabel('max')
plt.show()
```




In [33]:
```
fig2 = plt.figure()
ax = Axes3D(fig2)
ax.scatter(mean, min, max, c = 'r')
ax.set_xlabel('std')
ax.set_ylabel('min')
ax.set_zlabel('max')
plt.show()
```



# Experiment 2: HISTOGRAM

---

Write a program to extract the min, max, mean, and standard deviation of an ECG signal by considering a window over each pattern of the ECG dataset. Then plot a histogram of standard deviation, maximum value and mean value.

In [2]:
```
from numpy import *
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
ecg = pd.read_csv('ECG Data.csv')
```

In [4]:
```
def minimum(row):
    m = row[0]
    for x in row:
        if (x < m):
            m = x

    return m
```

In [5]:
```
def maximum(row):
    m = row[0]
    for x in row:
        if (x > m):
            m = x

    return m
```

In [6]:
```
def mean(row):
    sum = 0

    for x in row:
        sum = sum + x

    m = sum / len(row)

    return m
```

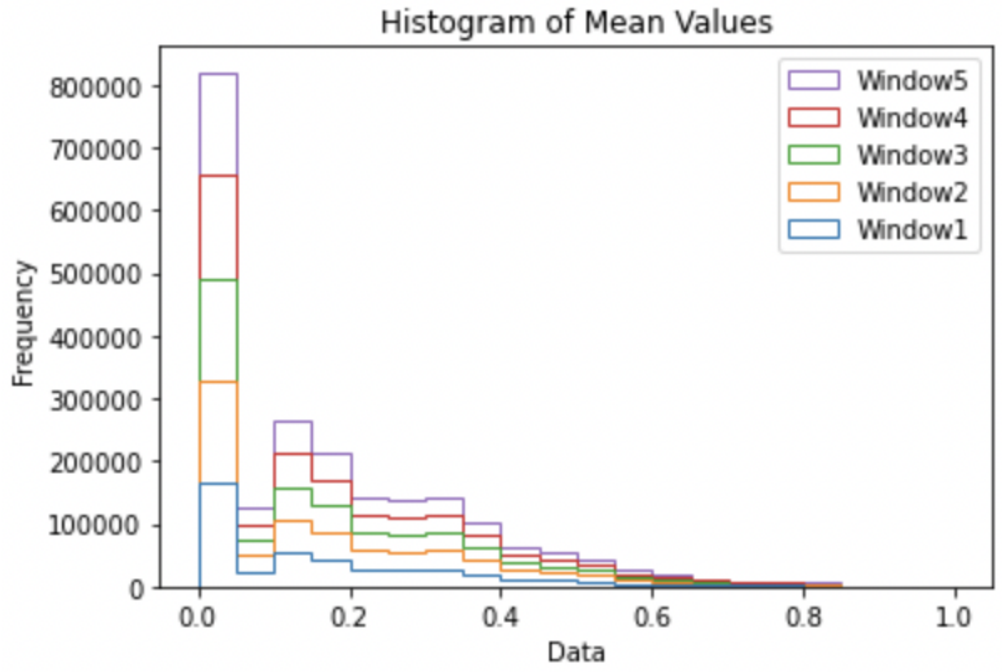
In [7]:
```
def standard_dev(row):
    sum = 0

    for x in row:
        sum = sum + x

    mean = sum / len(row)

    for x in row:
       sum = sum + square(x - mean)

    std = sqrt(sum/len(row))
    return std
```

In [8]:
```
win_dict = {}
min_dict = {}
max_dict = {}
std_dict = {}
mean_dict = {}

min_list = []
max_list = []
std_list = []
mean_list = []
```

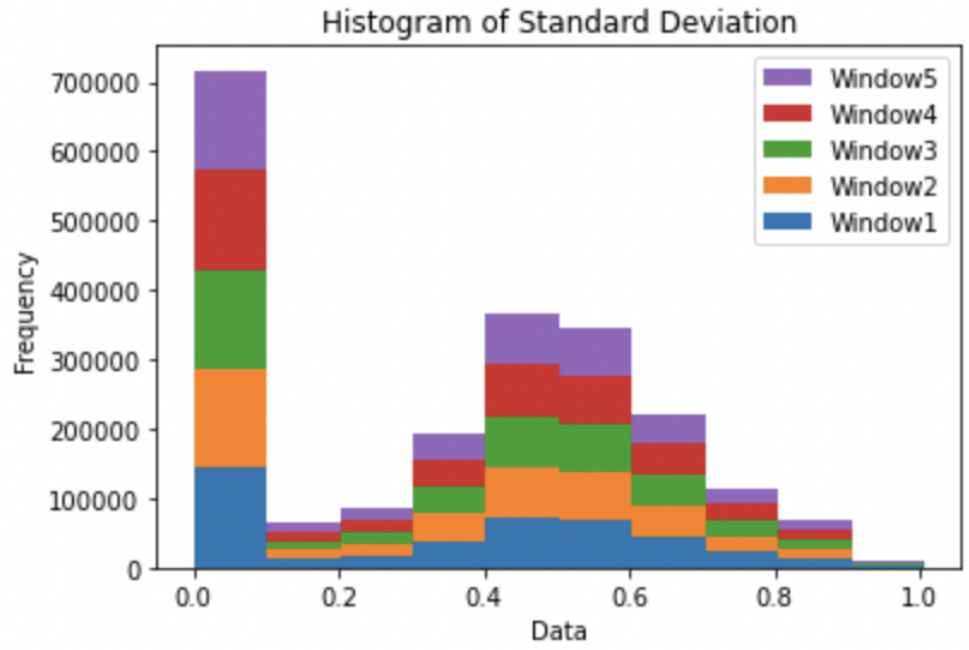
In [9]:
```
n=0
for x in range(0,len(ecg.columns),40):
    n=n+1
    win = ecg.iloc[:,x:x+40]
    win = win.to_numpy()
    win_dict['Window'+ str(n)] = win

    for y in win:

        min_list.append(minimum(y))
        max_list.append(maximum(y))
        mean_list.append(mean(y))
        std_list.append(standard_dev(y))


    min_dict['Window' + str(n)] = min_list
    max_dict['Window' + str(n)] = max_list
    mean_dict['Window' + str(n)] = mean_list
    std_dict['Window' + str(n)] = std_list
```

In [10]:
```
L = ['Window1','Window2','Window3','Window4','Window5']
```

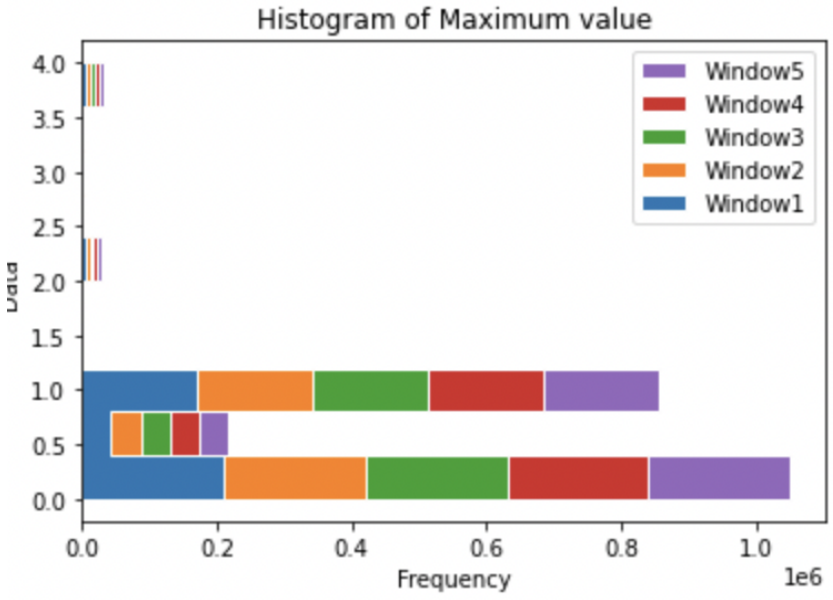
In [11]:
```
n_bins = 20
testing = [mean_dict[i] for i in L]
plt.hist(testing,n_bins, histtype = 'step', stacked=True, label = [i for i in L])
plt.legend(loc='upper right')
plt.xlabel("Data")
plt.ylabel("Frequency")
plt.title("Histogram of Mean Values")
plt.show()
```



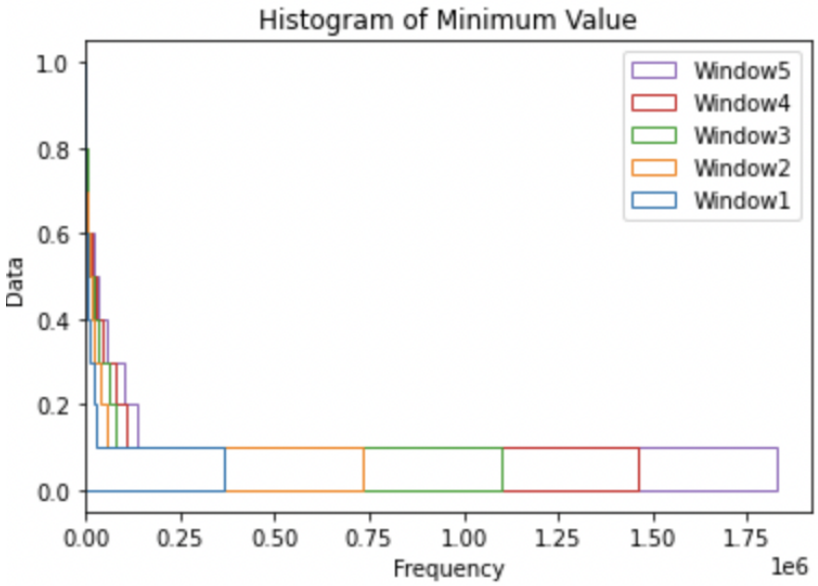

In [12]:
```
n_bins = 10
testing = [std_dict[i] for i in L]
plt.hist(testing,n_bins, histtype = 'stepfilled', stacked=True, label = [i for i in L])
plt.legend(loc='upper right')
plt.xlabel("Data")
plt.ylabel("Frequency")
plt.title("Histogram of Standard Deviation")
plt.show()
```




In [13]:
```
n_bins = 10
testing = [max_dict[i] for i in L]

plt.hist(testing,n_bins, histtype = 'stepfilled', stacked=True,
         edgecolor='w', orientation='horizontal',
         label = [i for i in L])

plt.legend(loc='upper right')
plt.xlabel("Frequency")
plt.ylabel("Data")
plt.title("Histogram of Maximum value")
plt.show()
```



In [14]:
```
n_bins = 10
testing = [min_dict[i] for i in L]

plt.hist(testing,n_bins, histtype = 'step', stacked=True,
         orientation='horizontal',
         label = [i for i in L])

plt.legend(loc='upper right')
plt.xlabel("Frequency")
plt.ylabel("Data")
plt.title("Histogram of Minimum Value")
plt.show()
```



# Experiment 3: **HEBBIAN LEARNING ALGORITHM**

---

CODE:

In [1]:
```
import pandas as pd
from numpy import *
import matplotlib.pyplot as plt
```

In [2]:
```
def logsig(x):
    y = 1/(1+ exp(-x))
    return y
```

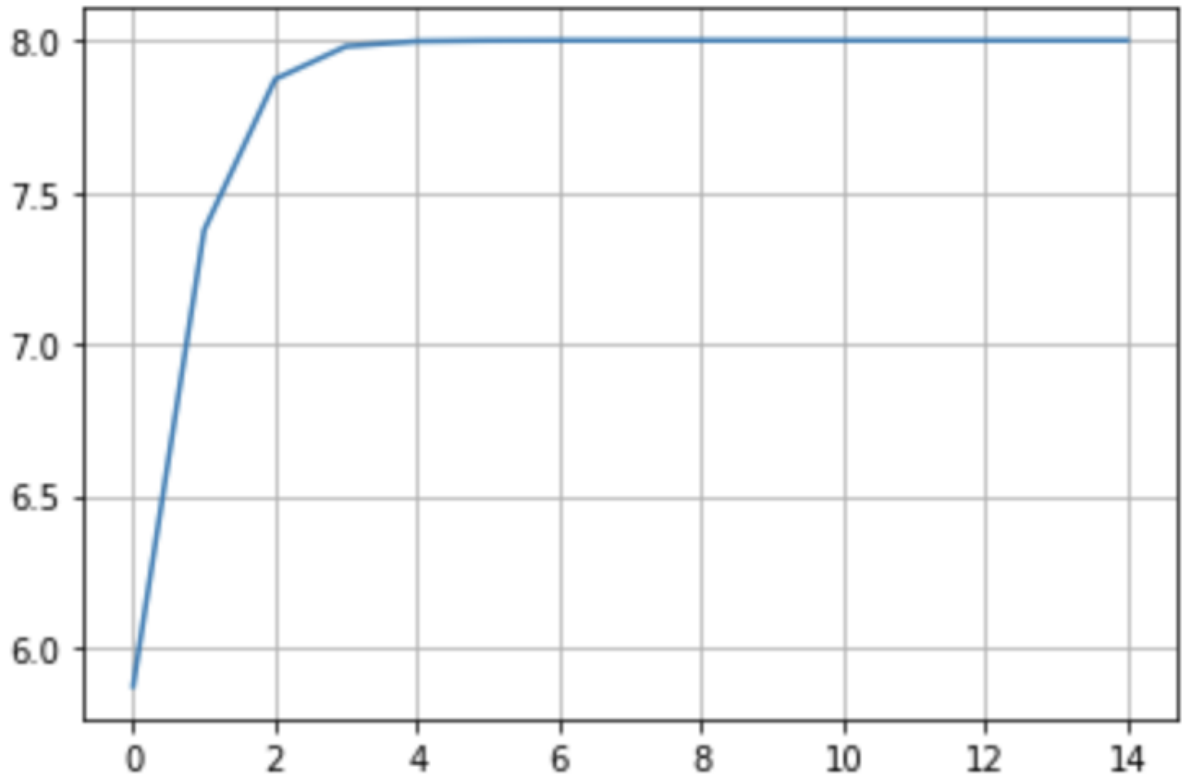
In [3]:
```
def singleNeuron(xx, yy, weight):

    #1. net = wp


    b = matrix([0])
    weight = matrix(weight)
    temp_err = []
    for pat, tar in zip(xx, yy):

        pat = matrix(pat)
        tar = matrix(tar)

        net = weight*pat.T + b

        ao = logsig(net)

        error = (tar - ao).tolist()[0][0]
        temp_err.append(error)

        #modify the weight
        alpha = 1.0

        weight = weight + alpha * ao * pat
        b = b + ao


        print("\nWeight=", weight)
        print("\nError=", error)
    return temp_err, weight
```

In [4]:
```
def main():
    ###DATSET###
    plt.ion()
    plt.figure(1)
    #X_train, X_test, y_train, y_test = dataprocessing()
    xx = [[1,1,1,1],[-1,1,-1,1],[1,1,1,-1],[1,-1,-1,1]] #pattern
    yy = [1,1,-1,-1]
    sse_list = []
    ww= matrix([0,0,0,0])

    for ii in range(0,15):
        #pdb.set_trace()
        error_list, ww = singleNeuron(xx, yy, ww)
        print("\n############ Iteration = ", ii)
        sse_list.append((matrix(error_list)*matrix(error_list).T).tolist()[0][0])

        plt.subplot(111)
        plt.plot(sse_list)
        plt.grid(True)
        plt.pause(0.01)
        plt.draw()
```

In [5]:
```
main()
```

OUTPUT:

```
Weight= [[28.09544174 28.18056084  1.22231562 27.71367937]]

Err click to unscroll output; double click to hide

Weight= [[27.09544174 29.18056084  0.22231562 28.71367937]]

Error= 1.056044141023449e-12

Weight= [[28.09544174 30.18056084  1.22231562 27.71367937]]

Error= -1.9999999999998839

Weight= [[29.09544174 29.18056084  0.22231562 28.71367937]]

Error= -1.999999999998748

############ Iteration =  14
```



# Experiment 4: SINGLE LAYER PERCEPTRON NEURAL NETWORK

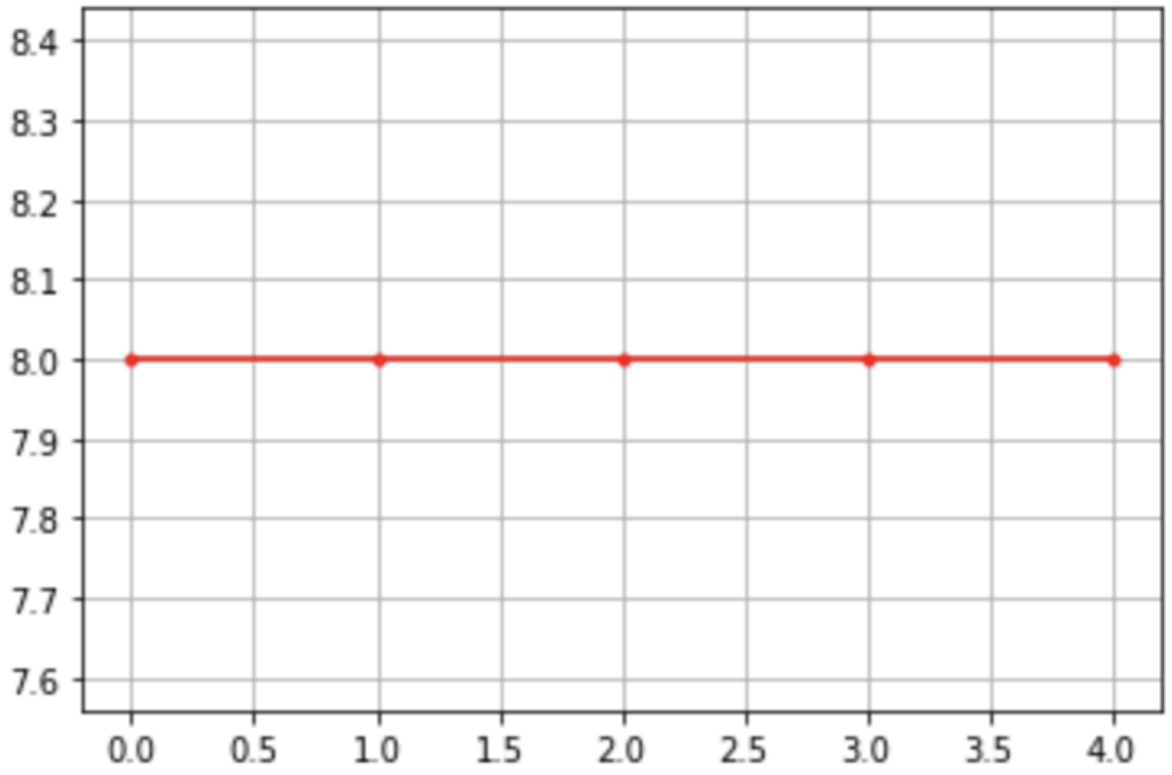
CODE:

In [1]:
```
import pandas as pd
from numpy import *
import matplotlib.pyplot as plt

def tf(n):

    if n<0:
        return -1
    elif n >=0:
        return 1
    else:
        return 0

def singleNeuron(xx, yy):
    ww= [0,0,0,0]
    weight = matrix(ww)
    b = matrix([0])
    temp_err = []


    for pat, tar in zip(xx, yy):
        pat = matrix(pat)
        tar = matrix(tar)
        net = weight*pat.T + b
        ao = tf(net)
```

```
        error = (tar - ao).tolist()[0][0]
        temp_err.append(error)
        alpha = 1

        if error <0:
            weight=weight - alpha*pat
            b = b + alpha*error
        elif error > 0:
            weight=weight + alpha*pat
            b = b + alpha*error
        elif error == 0:
            weight = weight
            b = b
        print("\nWeight=", weight)
        print("\nError=", error)
    return temp_err


def main():
    #DATASET#

    plt.ion()
    plt.figure(1)
    xx = [[1,1,1,1],[-1,1,-1,1],[1,1,1,-1],[1,-1,-1,1]] #pattern
    yy = [1,1,-1,-1]
    sse_list = []
    for i in range(0,5):


        error_list = singleNeuron(xx, yy)

        sse_list.append((matrix(error_list)*matrix(error_list).T).tolist()[0][0])

        plt.subplot(111)
        plt.plot(sse_list, 'r.-')
        plt.grid(True)
        plt.pause(0.01)
        plt.draw()
```

OUTPUT:

```
Weight= [[0 0 0 0]]
Error= 0
Weight= [[0 0 0 0]]
Error= 0
Weight= [[-1 -1 -1  1]]
Error= -2
Weight= [[-2  0  0  0]]
Error= -2
```



# Experiment 5: ADALINE NEURAL NETWORK

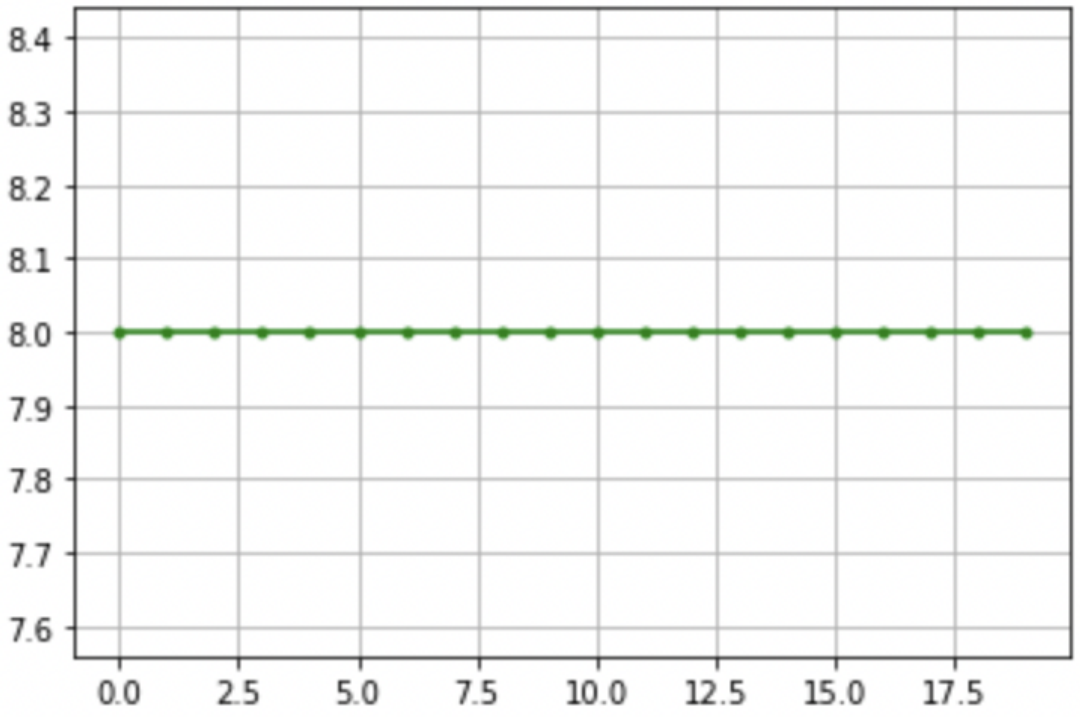
CODE:

In [2]:

```
import pandas as pd
from numpy import *
import matplotlib.pyplot as plt

def tf(n):

    if n<0:
        return -1
    elif n >=0:
        return 1
    else:
        return 0

def singleNeuron(xx, yy):
    ww= [0,0,0,0]
    weight = matrix(ww)
    b = matrix([0])
    temp_err = []


    for pat, tar in zip(xx, yy):
        pat = matrix(pat)
        tar = matrix(tar)
        net = weight*pat.T + b
        ao = tf(net)


        error = (tar - ao).tolist()[0][0]
        temp_err.append(error)
        alpha = 1


        weight= weight + alpha*(tar-net)*pat
        b = b + alpha*(tar-net)
        print("\nWeight=", weight)
        print("\nError=", error)
    return temp_err


def main():
    #DATASET#

    plt.ion()
    plt.figure(1)
    xx = [[1,1,1,1],[-1,1,-1,1],[1,1,1,-1],[1,-1,-1,1]] #pattern
    yy = [1,1,-1,-1]
    sse_list = []
    for i in range(0,20):

        error_list = singleNeuron(xx, yy)

        sse_list.append((matrix(error_list)*matrix(error_list).T).tolist()[0][0])

        plt.subplot(111)
        plt.plot(sse_list, 'g.-')
        plt.grid(True)
        plt.pause(0.01)
        plt.draw()


main()
```

OUTPUT:

```
Weight= [[1 1 1 1]]
Error= 0
Weight= [[1 1 1 1]]
Error= 0
Weight= [[-3 -3 -3  5]]
Error= -2
Weight= [[-9  3  3 -1]]
Error= -2
```



# Experiment 6: LMS ALGORITHM

---

CODE:

In [1]:
```
import pandas as pd
from numpy import *
import matplotlib.pyplot as plt
```

In [2]:
```
def logsig(x):
    y = 1/(1+ exp(-x))
    return y
```

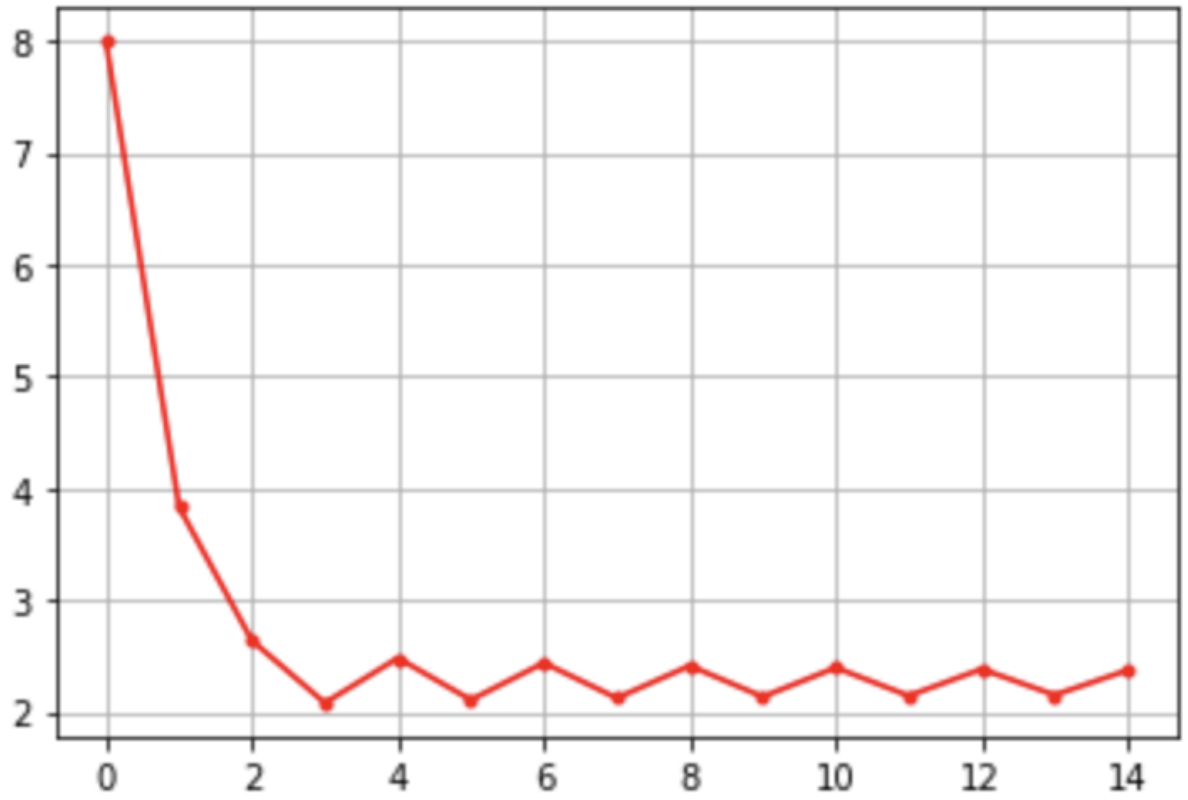
In [3]:
```
def singleNeuron(xx, yy, weight):

    #1. net = wp


    b = matrix([0])
    weight = matrix(weight)
    temp_err = []
    for pat, tar in zip(xx, yy):

        pat = matrix(pat)
        tar = matrix(tar)

        net = weight*pat.T + b

        ao = logsig(net)

        error = (tar - ao).tolist()[0][0]
        temp_err.append(error)

        #modify the weight
        alpha = 1.0

        weight = weight + 2 * alpha * pat *error
        b = b + (alpha*error)*2


        print("\nWeight=", weight)
        print("\nError=", error)
    return temp_err, weight
```

In [4]:
```
def main():
    ###DATSET###
    plt.ion()
    plt.figure(1)
    #X_train, X_test, y_train, y_test = dataprocessing()
    xx = [[1,1,1,1],[-1,1,-1,1],[1,1,1,-1],[1,-1,-1,1]] #pattern
    yy = [1,1,-1,-1]
    sse_list = []
    ww= matrix([0,0,0,0])

    for ii in range(0,15):
        #pdb.set_trace()
        error_list, ww = singleNeuron(xx, yy, ww)
        print("\n############ Iteration = ", ii)
        sse_list.append((matrix(error_list)*matrix(error_list).T).tolist()[0][0])

        plt.subplot(111)
        plt.plot(sse_list, 'r.-')
        plt.grid(True)
        plt.pause(0.01)
        plt.draw()
```

In [5]:
```
main()
```

OUTPUT:

```
Weight= [[-44.7575775   16.40611582  15.33031956  17.45237641]]
Error= 0.6096172299312359
Weight= [[-44.7575775   16.40611582  15.33031956  17.45237641]]
Error= 0.0
Weight= [[-46.7575775   14.40611582  13.33031956  19.45237641]]
Error= -1.0000000000001972
Weight= [[-48.7575775   16.40611582  15.33031956  17.45237641]]
Error= -1.0
############ Iteration =  14
```



# Experiment 7: NAIVE BAYES ALGORITHM

---

CODE:

In [210]:
```
import matplotlib.pylab as plt
from scipy.stats import norm
import pandas as pd
import numpy as np
from numpy import mean
from numpy import std
```

In [211]:
```
df = pd.read_csv('iris.data')
X = df.iloc[:,0:4]
Y = df.iloc[:,4]
```

In [212]:
```
def target_converter(Lable):
    A = []
    class_list = []
    output = []
    x = 0

    for i in Lable:
        if (i not in A):
            A.append(i)
            class_list.append(x)
            x += 1

    for i in Lable:
        x = A.index(i)
        output.append(x)

    return(class_list,np.array(output))
```

In [213]:
```
X = np.array(X)
class_list, Y = target_converter(np.array(Y))
```

In [214]:
```
def prior_list_generator(class_list,X):
    prior_list = []
    for i in class_list:
        xyi = X[Y == i]
        prior = len(xyi)/len(X)
        prior_list.append(prior)
    return prior_list
```

In [215]:
```
Prior = prior_list_generator(class_list,X)
```

In [216]:
```
def fit_dist(data):
    mu = mean(data)
    sigma = std(data)
    out_dist = norm(mu,sigma)
    return out_dist
```

In [217]:
```
def likelihood(X,Y,class_list):
    likelihood_list = []
    for i in class_list:
        xyi = X[Y == i]
        likelihood = []
        for j in range(len(X[0])):
            like_xj_yi = fit_dist(xyi[:,j])
            likelihood.append(like_xj_yi)
        likelihood_list.append(likelihood)
    return likelihood_list
```

In [218]:
```
Likelihood = likelihood(X,Y,class_list)
```

In [219]:
```
def probability(sample, prior, likelihood):
    Likelihood = 1
    for i in range(len(likelihood)):
        Likelihood = Likelihood * likelihood[i].pdf(sample[i])
    posterior = Likelihood * prior
    return posterior
```

In [220]:
```
def calc_prob(likelihood_list,prior_list,xsample,ysample):
    pred_list = []
    for likelihood,prior in zip(likelihood_list,prior_list):
        y_pred = probability( xsample, prior, likelihood)
        pred_list.append(y_pred)
    for i in range(len(pred_list)):
        if pred_list[i] == max(pred_list):
            prediction = i
    if prediction == ysample:
        return 1
    else:
        return 0
```

In [221]:
```
def accuracy_pred(X_sample,Y_sample,Likelihood,Prior):
    accuracy_list = []
    accurate_pred = 0
    for i, j in zip(X_sample, Y_sample):
        accuracy = calc_prob(Likelihood,Prior,i,j)
        accuracy_list.append(accuracy)
    for i in accuracy_list:
        if i == 1:
            accurate_pred += 1
    return accurate_pred/len(accuracy_list)
```

In [222]:
```
accuracy_list = []
sampling_size = []
def naive_bayes_classification(sample_percent):
    sample_size = int(sample_percent* len(X))
    acc = accuracy_pred(X[0:sample_size],Y[0:sample_size],Likelihood,Prior)
    sampling_size.append(sample_percent)
    accuracy_list.append(acc)
```

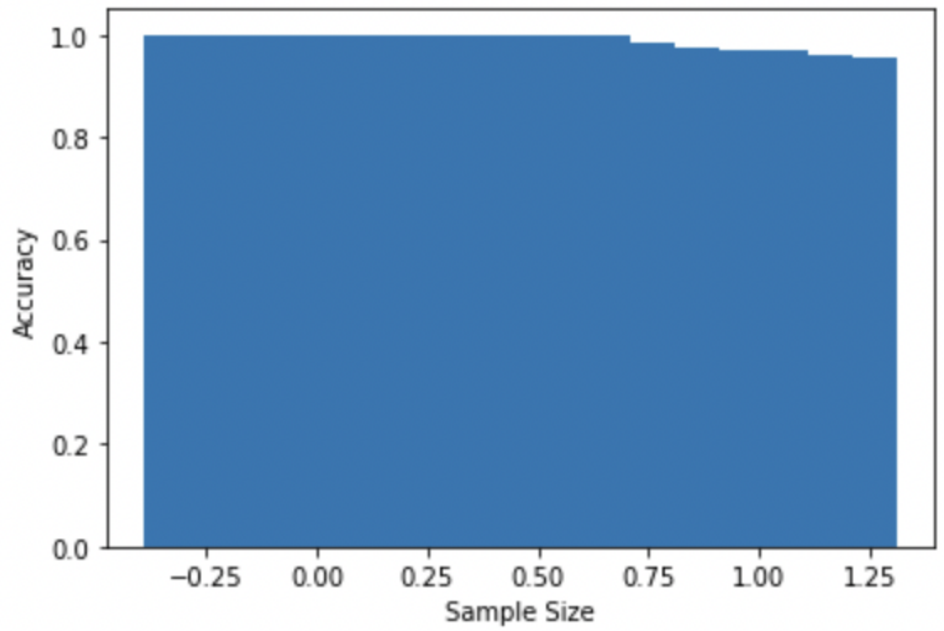
In [223]:
```
for i in range(1,100,10):
    naive_bayes_classification(i/100)
print(len(sampling_size),len(accuracy_list))
plt.bar(sampling_size,accuracy_list)
plt.xlabel("Sample Size")
plt.ylabel("Accuracy")
plt.show()
```

In [224]:
```
sampling_size
```

Out[224]:
```
[0.01, 0.11, 0.21, 0.31, 0.41, 0.51, 0.61, 0.71, 0.81, 0.91]
```

OUTPUT:

```
10 10
```

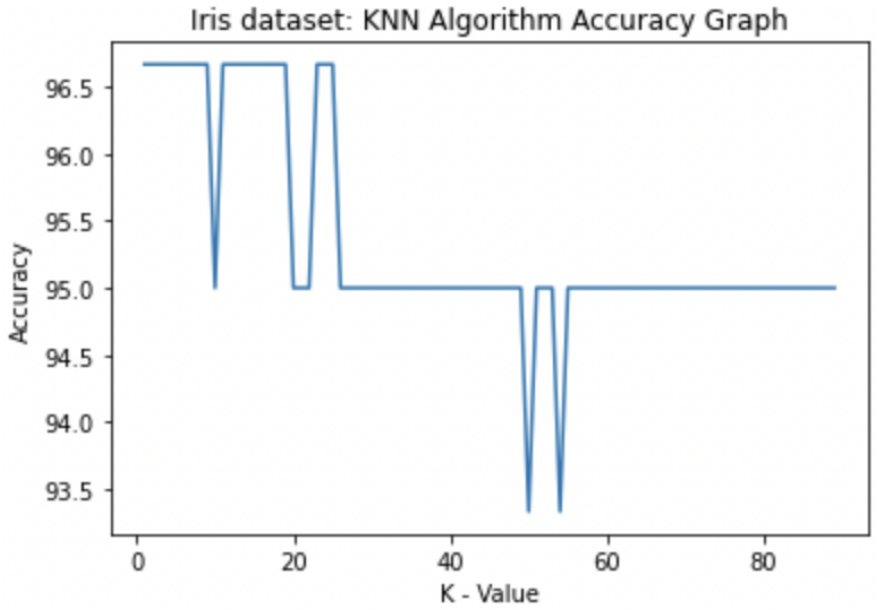


# Experiment 8: K-NEAREST NEIGHBOR ALGORITHM

CODE:

In [11]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score

df = load_iris()

accuracy_list = []
X = df.data[:,: 4]
Y = df.target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.4,random_state= 2)
k = []
for i  in range(1,90):
    knn = KNeighborsClassifier (n_neighbors = i, weights = "distance")
    knn.fit(X_train, Y_train)
    pred = knn.predict(X_test)
    k.append(i)
    accuracy_list.append (accuracy_score (Y_test, pred)*100)
plt.plot(k,accuracy_list)
#plt.ylim(0.7,0.8)
plt.title ("Iris dataset: KNN Algorithm Accuracy Graph")
plt.xlabel('K - Value')
plt.ylabel('Accuracy')
plt.show()
```

OUTPUT:



# Experiment 9: **MULTI LAYER PERCEPTRON (MLP) NEURAL NETWORK**

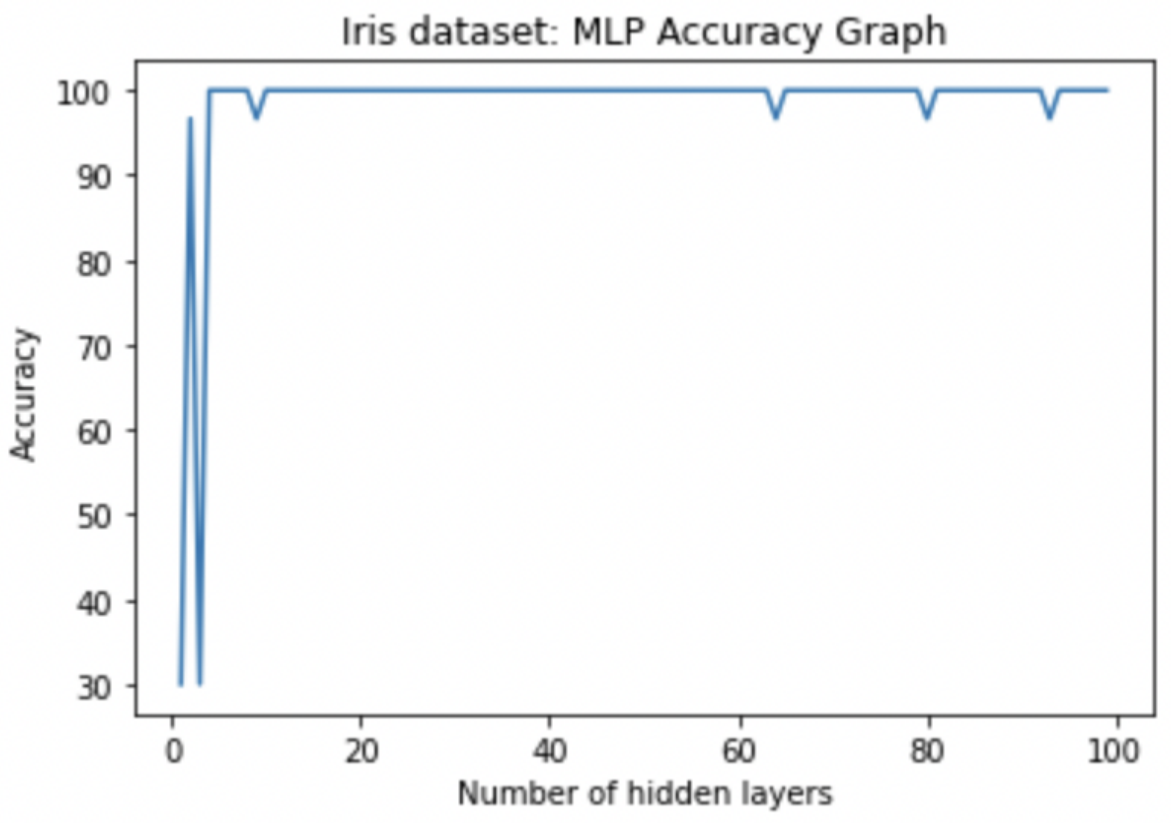
---

CODE:

In [5]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score,roc_curve,classification_report
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris



df = load_iris()

X = df.data
y = df.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
accuracy_list = []
layer_list = []

def Multi_layer_Perceptron(hidden_layers):
    cl1 = MLPClassifier(solver="lbfgs", alpha=0.05, hidden_layer_sizes=(hidden_layers,), random_state=1)
    cl1.fit(X_train, y_train)
    y_pred = cl1.predict(X_test)
    layer_list.append(hidden_layers)
    accuracy_list.append (accuracy_score (y_test, y_pred)*100)
```

In [6]:
```
for i in range(1,100):
    Multi_layer_Perceptron(i)
plt.plot(layer_list,accuracy_list)
plt.xlabel('Number of hidden layers')
plt.ylabel('Accuracy')
plt.title ("Iris dataset: MLP Accuracy Graph")
plt.show()
```

OUTPUT:



# Experiment 10: LINEAR REGRESSION ALGORITHM

CODE:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.axis import Axis
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('iris.csv')
X = df.iloc[:,2]
Y = df.iloc[:,0]
df
```

In [3]:
```
def target_converter(Lable):
    A = []
    output = []
    x = 0

    for i in Lable:
        if (i not in A):
            A.append(i)
            x += 1

    for i in Lable:
        x = A.index(i)
        output.append(x)

    return(np.array(output))
```

In [4]:
```
def linear_regression(X,Y):
    x_bar = X.mean()
    y_bar = Y.mean()
    sigma_x = X.var()
    cov = 0
    for i,j in zip(X,Y):
        cov = cov + (i - x_bar)*(j - y_bar)
    b_xy = cov/sigma_x
    y = b_xy*(X - x_bar) + y_bar
    fig, ax = plt.subplots()
    plt.plot(X,y)
    plt.scatter(X,Y)
    ax.grid()
    plt.xlabel('petal_length')
    plt.ylabel('Species')
    plt.show()
```

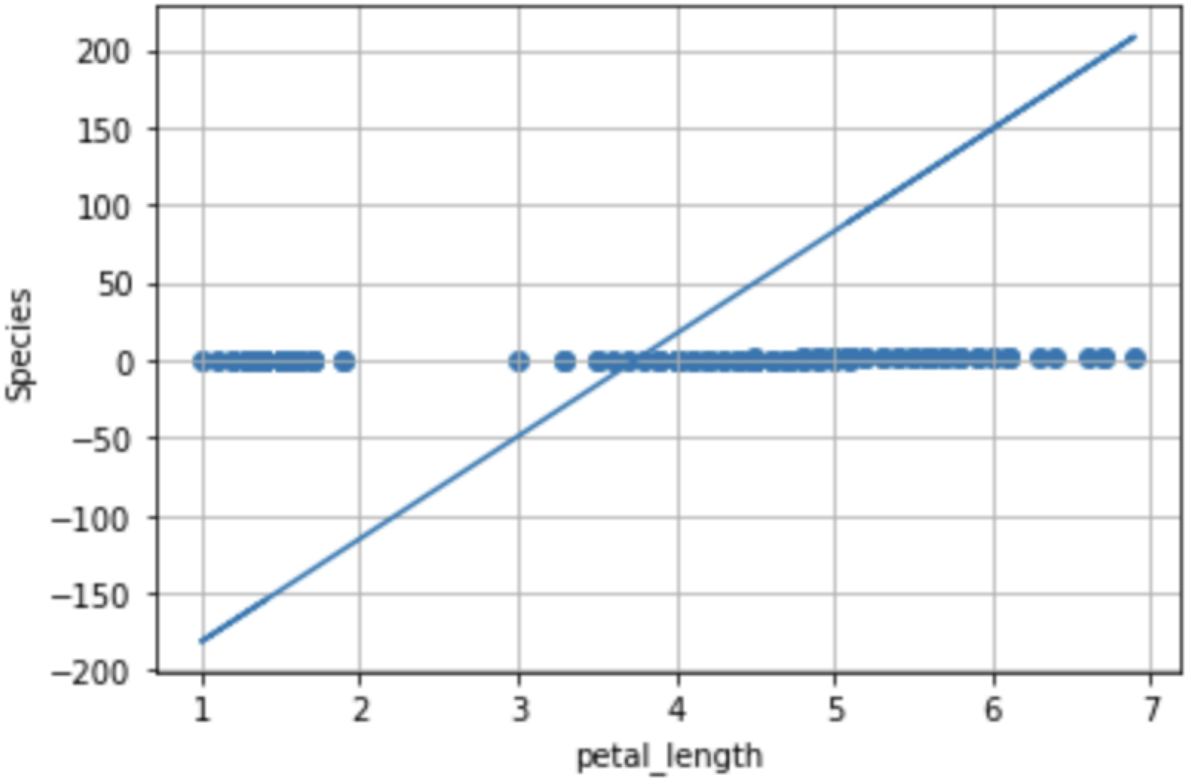
In [5]:
```
X = np.array(X)
Y = target_converter(np.array(Y))
linear_regression(X,Y)
```

OUTPUT:

Out[2]:

| | Target | sepal_width | petal_length | petal_width | sepal_length |
|---|---|---|---|---|---|
| **0** | Iris-setosa | 3.5 | 1.4 | 0.2 | 5.1 |
| **1** | Iris-setosa | 3.0 | 1.4 | 0.2 | 4.9 |
| **2** | Iris-setosa | 3.2 | 1.3 | 0.2 | 4.7 |
| **3** | Iris-setosa | 3.1 | 1.5 | 0.2 | 4.6 |
| **4** | Iris-setosa | 3.6 | 1.4 | 0.2 | 5.0 |
| **...** | ... | ... | ... | ... | ... |
| **145** | Iris-virginica | 3.0 | 5.2 | 2.3 | 6.7 |
| **146** | Iris-virginica | 2.5 | 5.0 | 1.9 | 6.3 |
| **147** | Iris-virginica | 3.0 | 5.2 | 2.0 | 6.5 |
| **148** | Iris-virginica | 3.4 | 5.4 | 2.3 | 6.2 |
| **149** | Iris-virginica | 3.0 | 5.1 | 1.8 | 5.9 |

150 rows × 5 columns


200
150
100
50
0
−50
−100
−150
−200
Species
1
2
3
4
5
6
7
petal_length

# Experiment 11: BACKPROPAGATION ALGORITHM

---

CODE:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sympy import *
```

In [2]:
```
x = Symbol('x')
f1_exp = 1/(1+exp(-x))
f2_exp = x
f1_prime_exp = diff(f1_exp)
f2_prime_exp = diff(f2_exp)
f1 = lambdify(x,f1_exp)
f2 = lambdify(x,f2_exp)
f1_prime = lambdify(x,f1_prime_exp)
f2_prime = lambdify(x,f2_prime_exp)
Tf_list = [f1,f2]
F_prime = [f1_prime,f2_prime]
```

In [3]:
```
def Z_op(X,W,B):
    Z = np.dot(W,X.T) + B
    return Z
```

In [4]:
```
def Y_op(Tf,*Z):
    Yout = []
    for z in Z:
        y = Tf(z)
        Yout.append(y)
    return np.array(Yout)
```

In [5]:
```
def Cost(Y,T):
    errors = T - Y
    SSE = 0
    for error in errors:
        SSE = SSE + error*error
    MSE = SSE/len(errors)
    return MSE
```

In [6]:
```
def feed_forward(W,B,X,Tf_list):
    Z_list = []
    X_list = []
    X_list.append(X)
    for i,j,k in zip(range(len(W)),range(len(B)),range(len(Tf_list))):
        z = Z_op(X,W[i],B[j])
        Z_list.append(z)
        y = Y_op(Tf_list[k],z)
        X = y
        X_list.append(X)
    return Z_list,X_list
```

In [7]:
```
w1 = np.array([[0.2,0.4 , -0.5],
    [-0.3, 0.1, 0.2]])
w2 = np.array([-0.3,-0.2])
W = [w1,w2]
X = np.array([1,0,1])
b1 = np.array([-0.4,0.2])
b2 = np.array([0.1])
B = [b1,b2]
alpha = 0.9
Target = np.array([1])
print(W)
print(B)
print(Target)
print(X)
```

In [8]:
```
def sensitivity(Z_list,X_list,F_prime,Target,Weights):
    F_list = []
    Sn_list = []
    for i in range(len(Z_list)):
        D = []
        for k in Z_list[i]:
            d = F_prime[i](k)
            D.append(d)
        Dag = np.array(D)
        F_dot = np.diag(Dag)
        F_list.append(F_dot)
    F_n = F_list.pop(len(F_list)-1)
    A_n = X_list.pop(len(X_list)-1)
    error = Target - A_n
    S_n = np.array(-2*np.dot(F_n,error.T))
    Sn_list.append(S_n)
    for i in range(len(F_list),0,-1):
        F_dot_W = np.array(np.dot(F_list[i-1],Weights[i].T))
        S_n = F_dot_W*S_n
        Sn_list.append(S_n.T)
    return Sn_list
```

In [9]:
```
def weight_bias_updation(Weight,Bias,Alpha,Sensitivity,X_List):
    new_weights = []
    new_bias = []
    def reshaping(vector):
        if vector.shape == (len(vector),1):
            return vector.reshape(len(vector),)
        else :
            return vector
    def Reverse(lst):
        return [ele for ele in reversed(lst)]
    sens = Reverse(Sensitivity)

    for i in range(len(Weight)):
        delta_w = reshaping(Alpha*sens[i]*X_List[i].T)
        w = Weight[i] - delta_w #As a(1) = X[0], The loop will run for w[0],w[1]...w[n-1], which we will use as w1,w
        new_weights.append(w)
    for i in range(len(Bias)):
        b = Bias[i] - reshaping(Alpha*sens[i])
        new_bias.append(b)
    return new_weights,new_bias
```

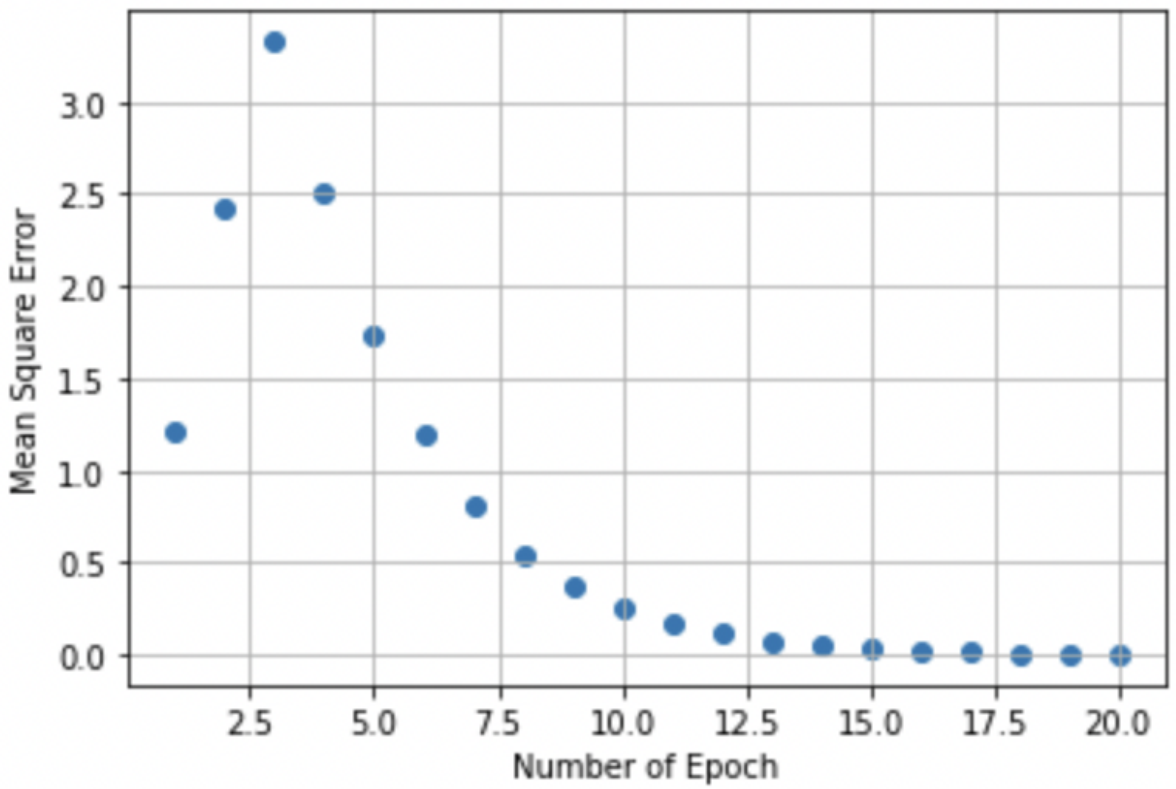
In [21]:
```
def Error_Collection(Target,Output,All_Errors):
  Error = Cost(Output,Target)
  All_Errors.append(Error)
```

In [34]:
```
def backpropagation(Weight,Bias,Pattern,Target,Alpha,Transfer_Function,F_Prime,Epoch):
    all_errors = []
    epoch_list = []
    for i in range(Epoch):
        zl,xl = feed_forward(Weight,Bias,Pattern,Transfer_Function)
        actual_output = xl[len(xl)-1]
        Error_Collection(Target,actual_output,all_errors)
        sen = sensitivity(zl,xl,F_prime,Target,Weight)
        Weight,Bias = weight_bias_updation(Weight,Bias,alpha,sen,xl)
        epoch_list.append(i+1)
    plt.scatter(epoch_list,all_errors)
    plt.xlabel('Number of Epoch')
    plt.ylabel('Mean Square Error')
    plt.grid()
```

In [36]:
```
backpropagation(W,B,X,Target,alpha,Tf_list,F_prime,20)
```

OUTPUT:

```
[array([[ 0.2,  0.4, -0.5],
        [-0.3,  0.1,  0.2]]), array([-0.3, -0.2])]
[array([-0.4,  0.2]), array([0.1])]
[1]
[1 0 1]
```


Mean Square Error
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of Epoch