

LAB 7 : Regression Part 2

In this Lab we will look into the shortcomings of Linear Regression and see how those problems can be solved using Logistic Regression. We will also explore Polynomial Regression

1. Polynomial Regression
2. Linear Regression on a specific pattern of data to observe shortcomings
3. Logistic Regression to solve those problems

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from jupyterthemes import jtplot
jtplot.style(theme='gruvboxd', context='notebook')
```

Polynomial Regression

1. Generate data using relation $y = 0.25x^3 + 1.25x^2 - 3x - 3$
2. Corrupt y by adding random noise (uniformly sampled)
3. Fit the generated curve using different polynomial order. (Using matrix inversion and gradient descent)

In [2]:

```
## Use the Regression class defined in the previous lab

class regression:

    # Constructor

    def __init__(self, name='reg'):

        self.name = name # Create an instance variable


    def grad_update(self,w_old,lr,y,x):

        #write code here

        w = w_old + 2*lr*x@(y-x.T@w_old)/y.shape[0]

        return w


    def error(self,w,y,x):

        return np.average((y-x.T@w)**2) # write code here


    def mat_inv(self,y,x_aug):

        return np.linalg.pinv(x_aug@x_aug.T)@x_aug@y# write code
here


    # By Gradien descent

    def Regression_grad_des(self,x,y,lr):

        w = np.reshape(np.zeros(x.shape[0]),(x.shape[0],1))

        err = []

        for i in range(100000):

            # write code here

            err.append(self.error(w,y,x))

            w = self.grad_update(w,lr,y,x)

            dev=np.abs(self.error(w,y,x)-err[-1])# write code
here)

            # print(i)

            if dev <= 0.000001 :

                break

        w_pred = w

        return w_pred,err
```

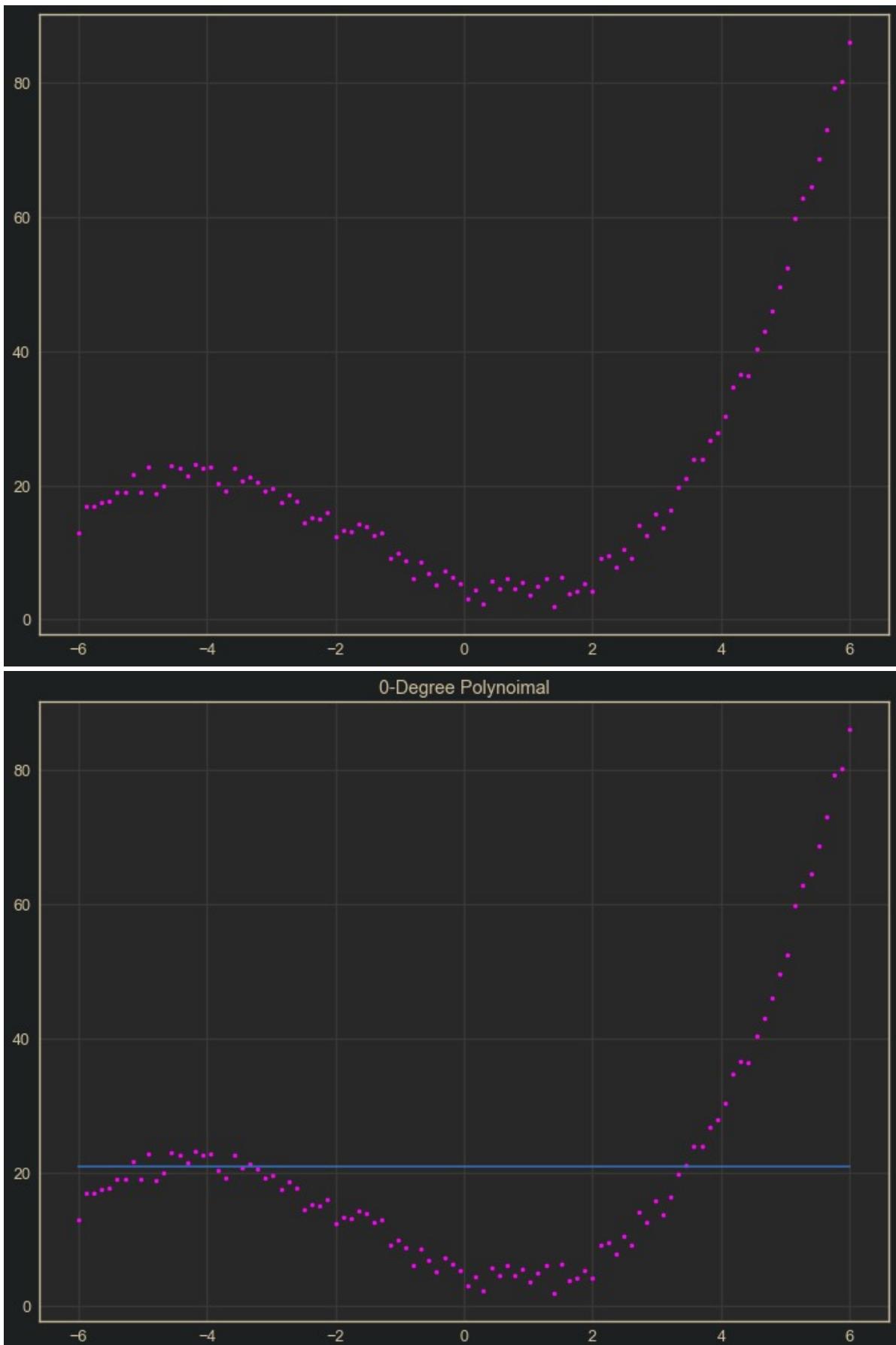
In [3]:

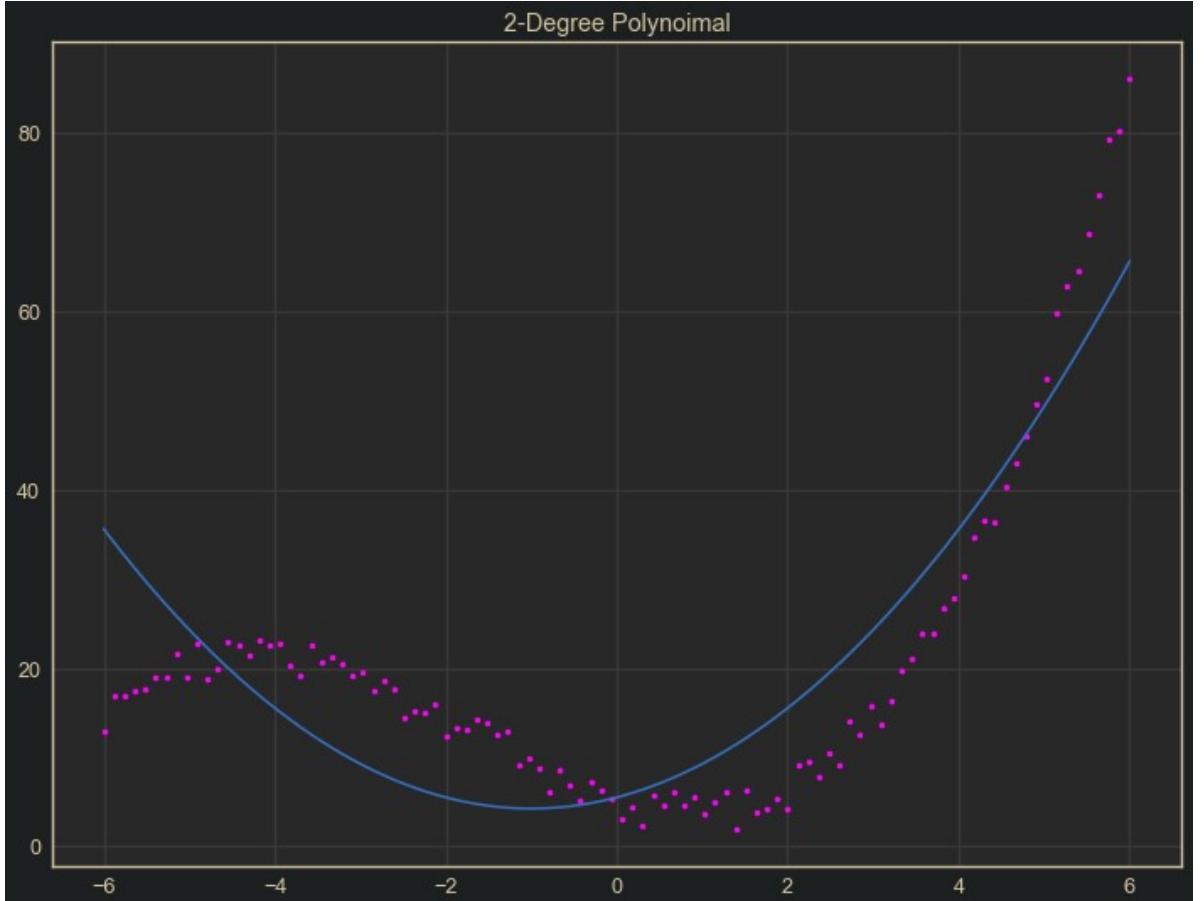
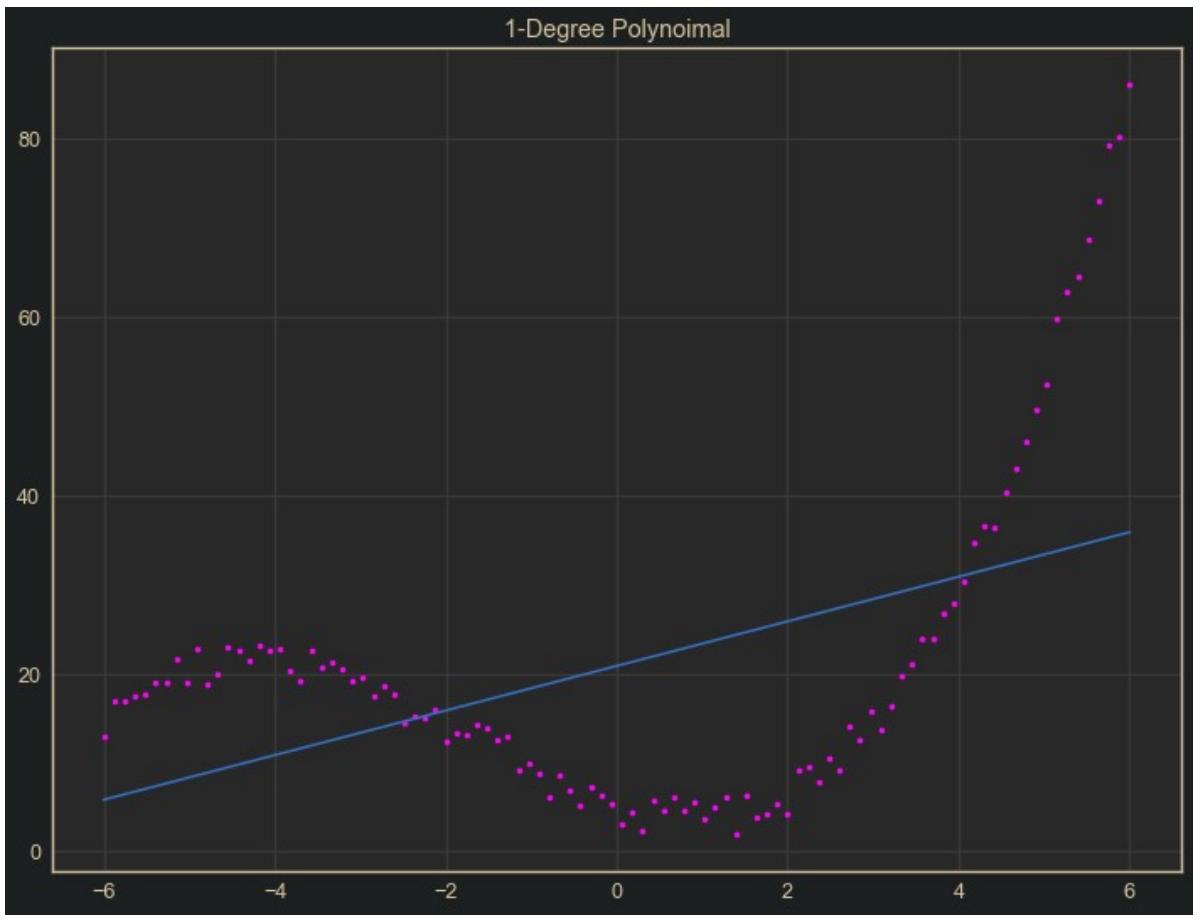
```
## Data generation
x=np.linspace(-6,6,100)
x=x[:,np.newaxis]
# print(x.shape)
w = np.transpose([[3,-3,1.25,0.25]])## Define Weights as per the
given equation
# print(w.shape)
## Function to transform the data into polynomial

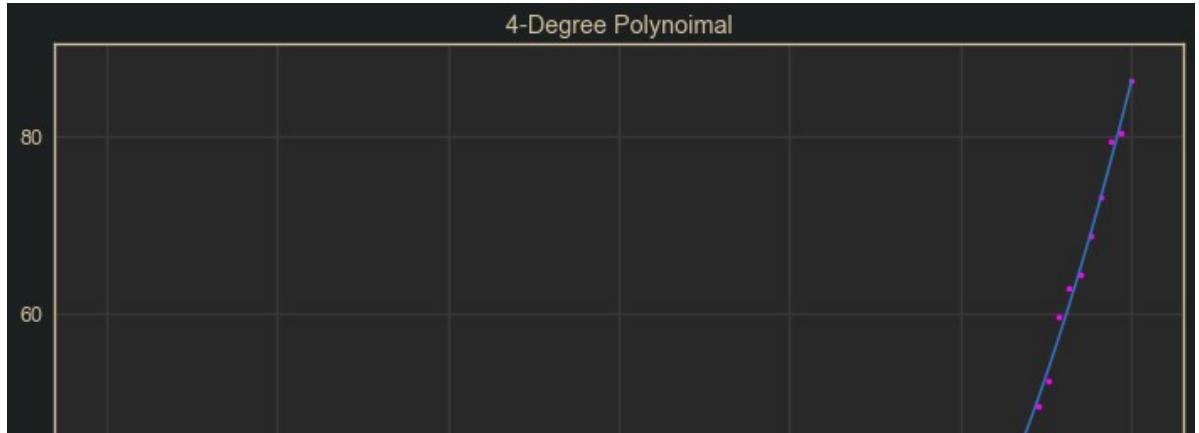
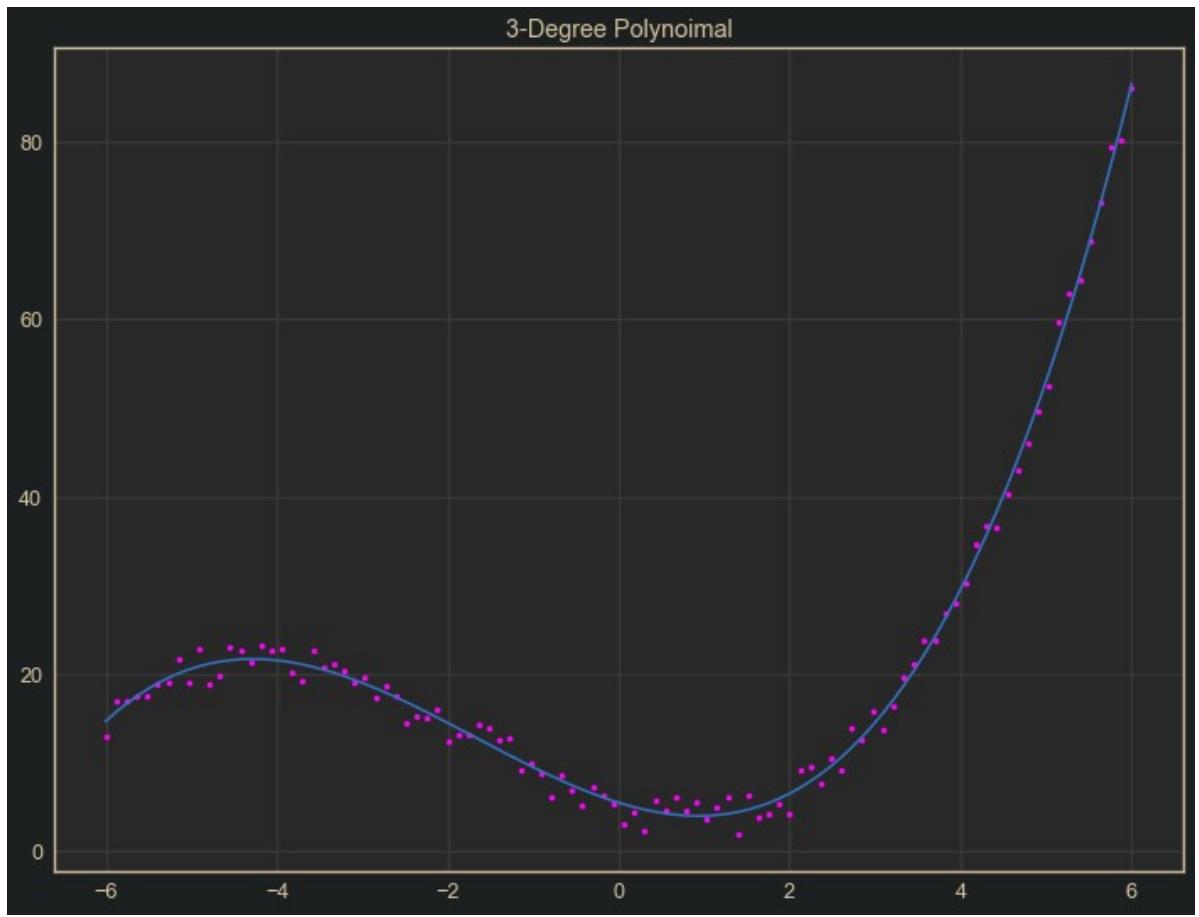
def data_transform(X,degree):
    ## Write your code here
    X_new = np.empty((degree+1,X.shape[1]))
    for i in range(degree+1):
        X_new[i,:] = x**i;
    return X_new

X = data_transform(x,3)
# print(X.shape)
y = X.T @ w
# print(y.shape)
y = y+5*np.random.uniform(0,1,y.shape)
# print(y.shape)
plt.figure(figsize=(12,9))
plt.plot(x.T,y,'.',c='magenta');

reg=regression()
# By computation
# Code for degree 0 polynomial fitting
# # Write the code for degree 1 polynomial fitting
# # Write the code for degree 2 polynomial fitting
# # Write the code for degree 3 polynomial fitting
# # Write the code for degree 4 polynomial fitting
for degree in range(5):
    X_1 = data_transform(x,degree)
    w_mat=reg.mat_inv(y,X_1)
    y_pred=X_1.T @ w_mat
    plt.figure(figsize=(12,9))
    plt.plot(x.T,y,'.',c='magenta');
    plt.plot(x.T,y_pred);
    plt.title(str(degree)+'-Degree Polynoimal');
```



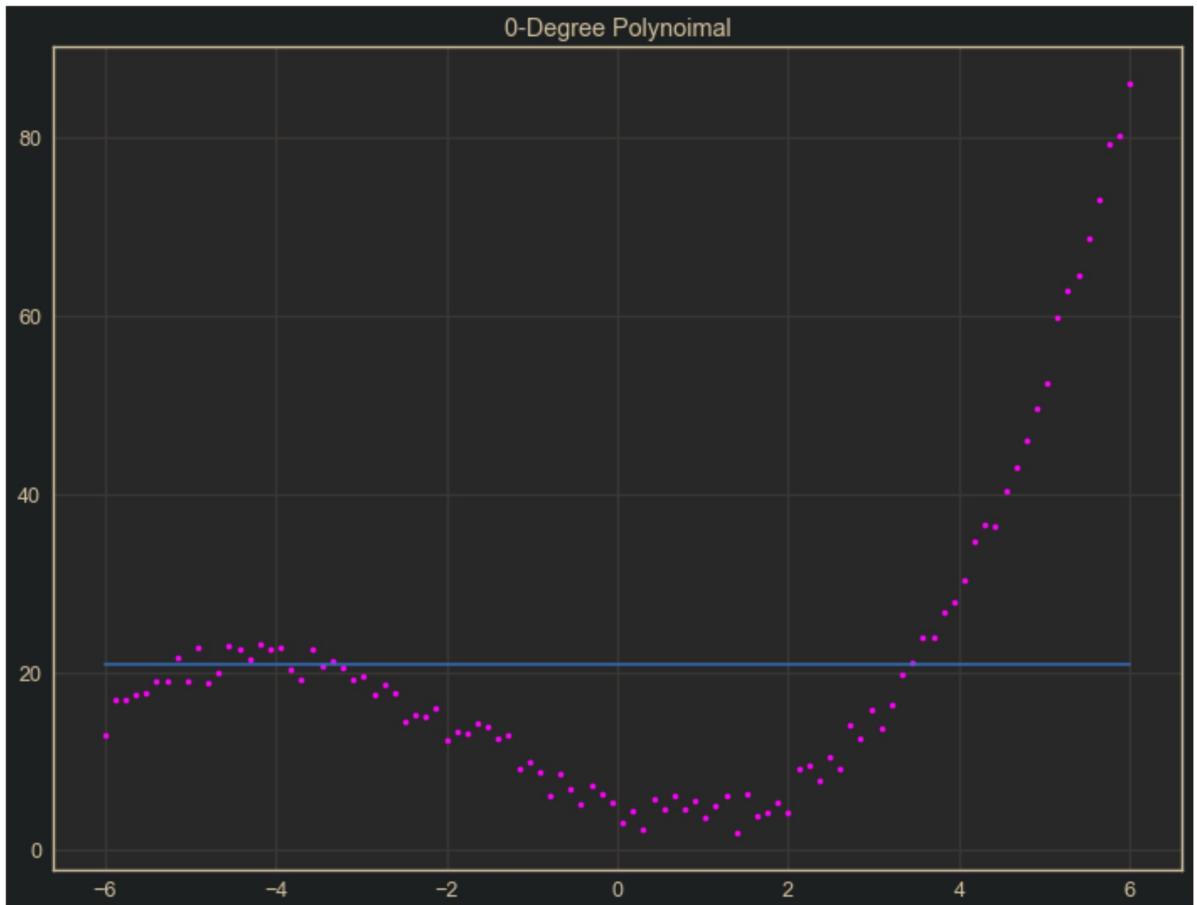


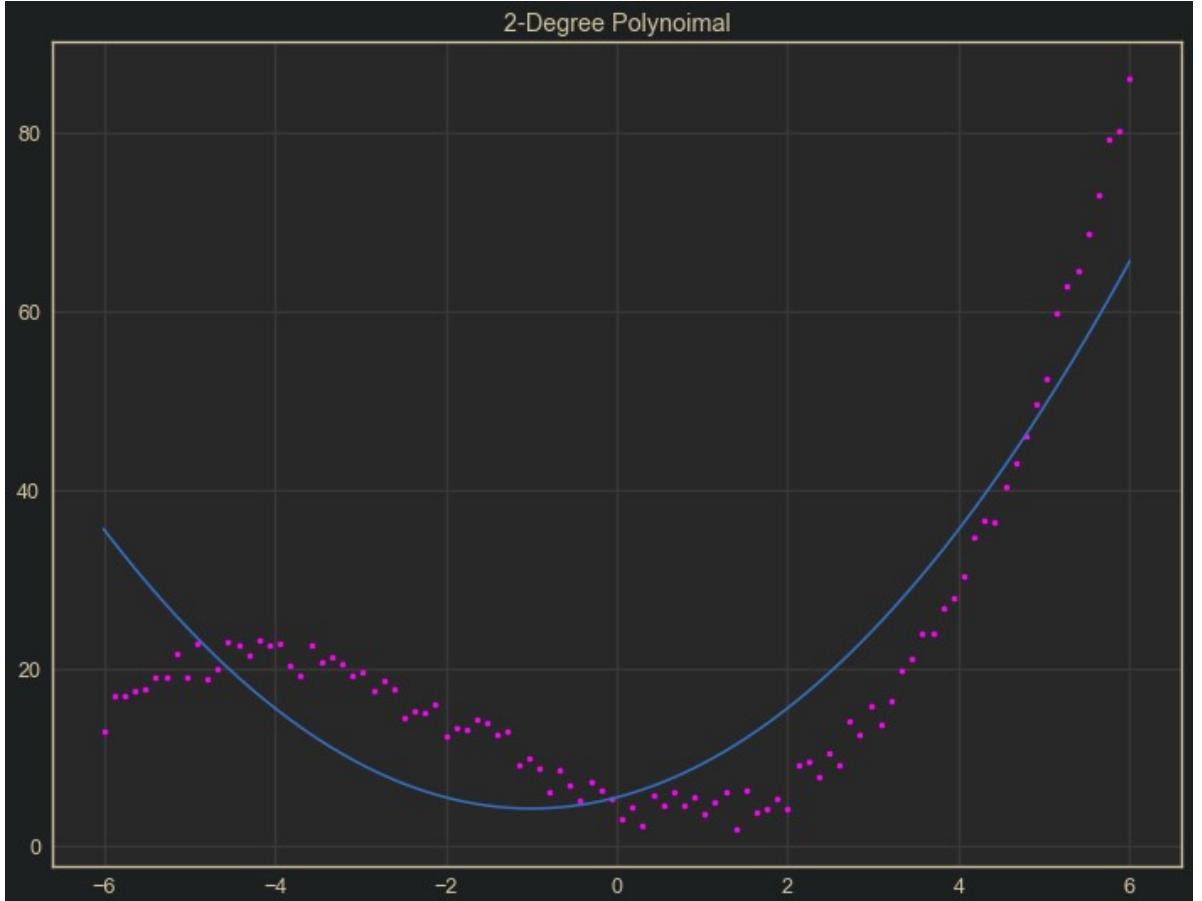
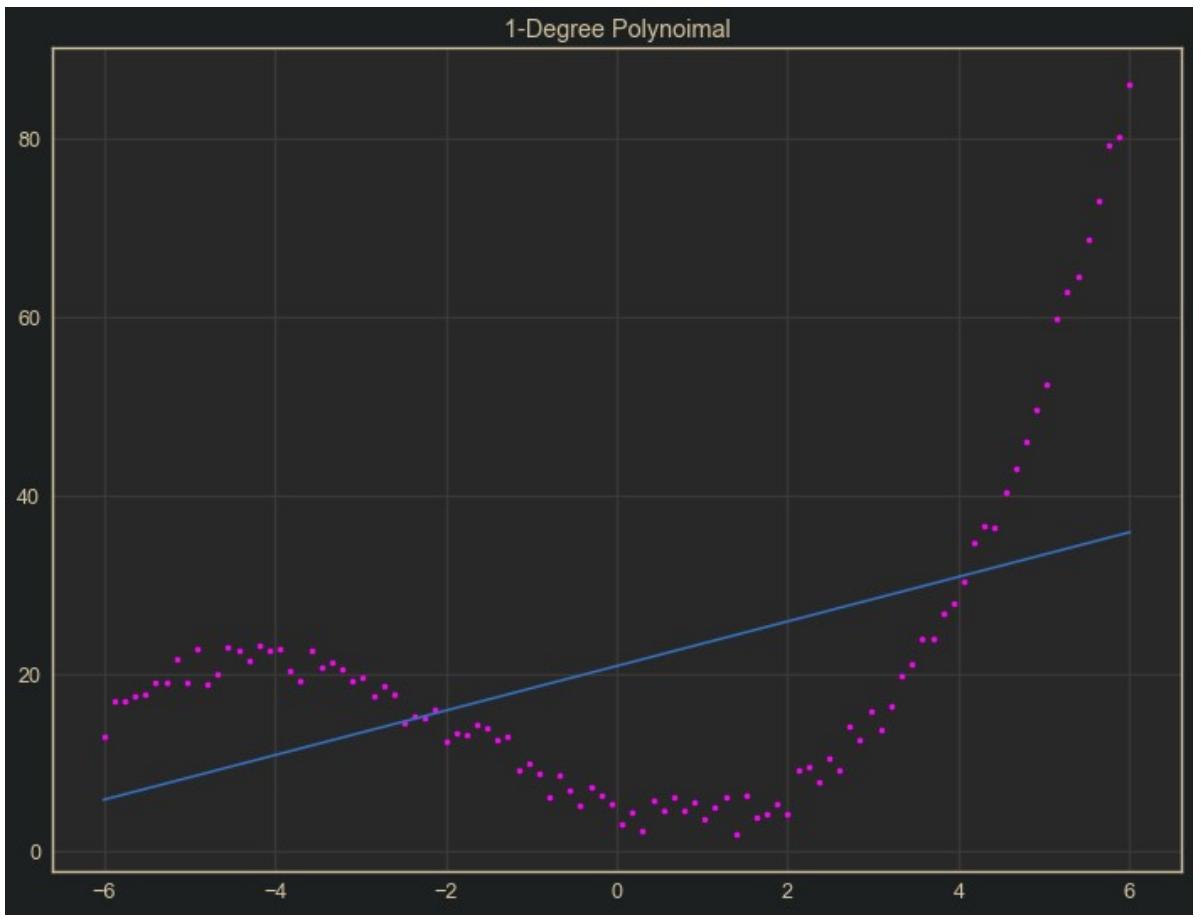


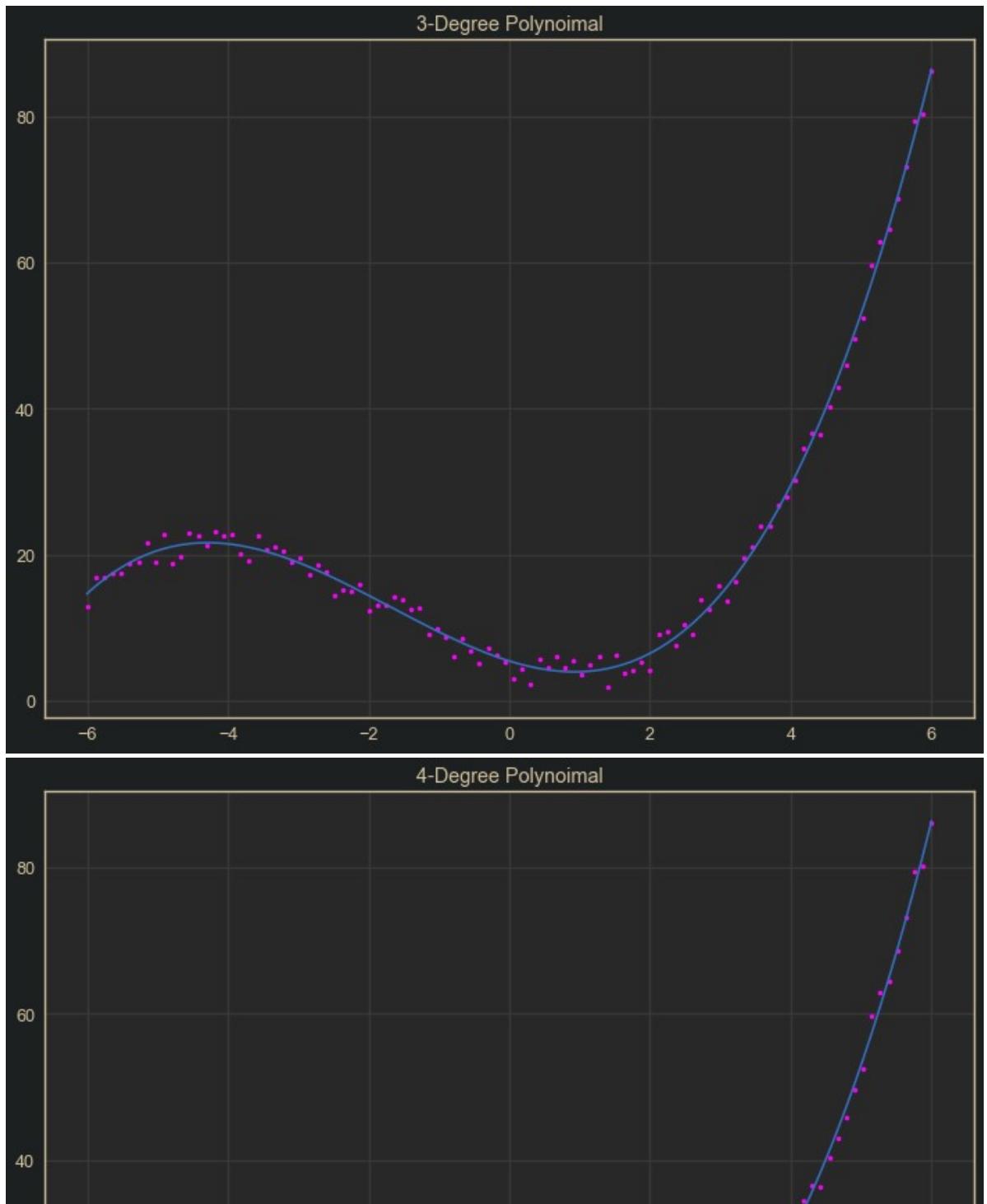
In [4]:

```
# By Gradient Descent
## Write your code here
for degree in range(5):
    x_aug = data_transform(x,degree)
    reg = regression()
    lr=0.01
    x_std = x_aug;
    for i in np.arange(1,x_aug.shape[0]):
        x_std[i,:] = (x_aug[i,:]-
np.mean(x_aug[i,:]))/np.std(x_aug[i,:]);

    w_pred,err=reg.Regression_grad_des(x_std,y,lr);
    #     print(w_pred.shape)
    y_pred = x_std.T @ w_pred
    #     print(y_pred.shape)
    plt.figure(figsize=(12,9))
    plt.plot(x.T,y,'.',c='magenta');
    plt.plot(x.T,y_pred);
    plt.title(str(degree)+'-Degree Polynoimal');
```





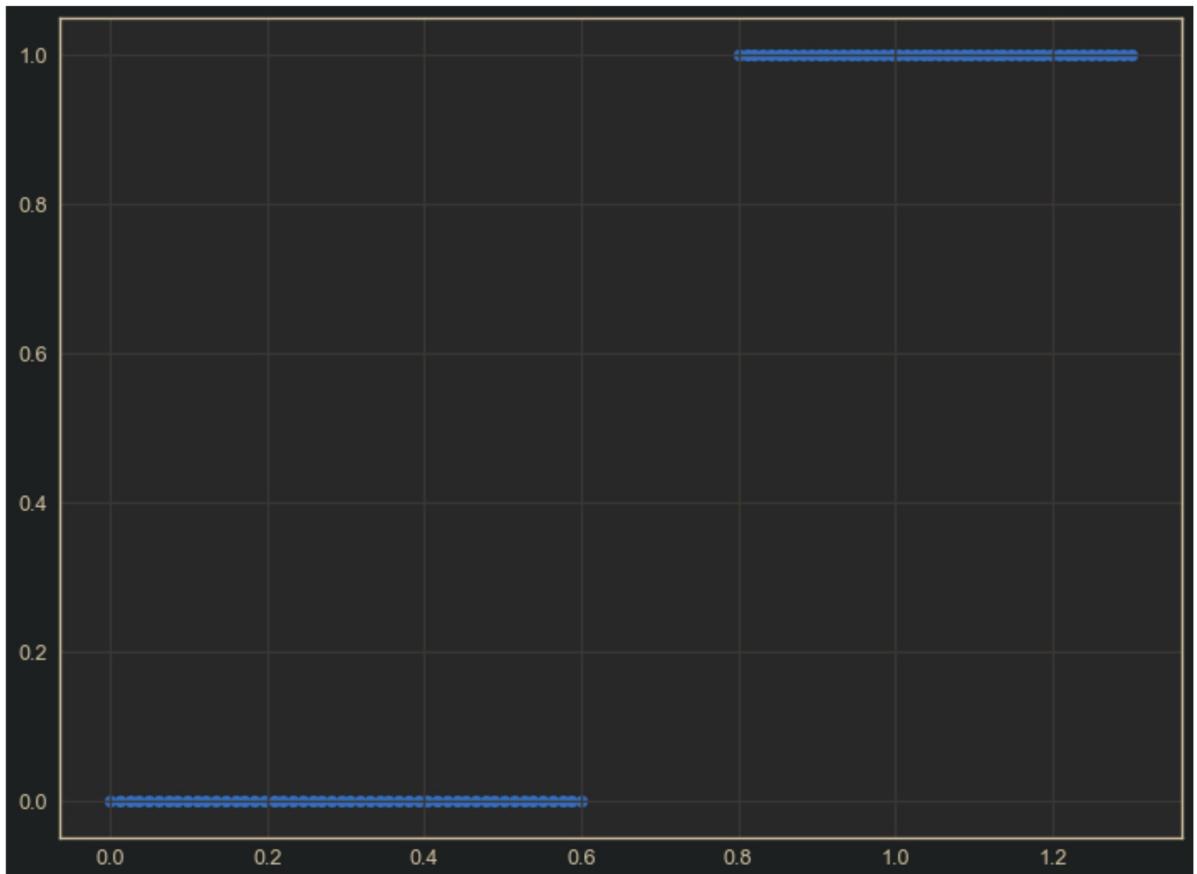


Linear Regression

Generate the data as shown in the figure below

In [5]:

```
## Write your code here  
x = np.append(np.linspace(0,0.6,50),np.linspace(0.8,1.3,50))  
x = x[:,np.newaxis]  
# print(x.shape)  
y = np.zeros(x.shape)  
# print(y.shape)  
y[50:,0] = 1  
  
plt.figure(figsize=(12,9))  
plt.scatter(x,y);
```



Use the Regression class defined in the previous lab to fit the curve

In [6]:

```
## Write your Code here

class regression:

    # Constructor

    def __init__(self, name='reg'):

        self.name = name # Create an instance variable


    def grad_update(self,w_old,lr,y,x):

        #write code here

        w = w_old + 2*lr*x@(y-x.T@w_old)/y.shape[0]

        return w


    def error(self,w,y,x):

        return np.average((y-x.T@w)**2) # write code here


    def mat_inv(self,y,x_aug):

        return np.linalg.pinv(x_aug@x_aug.T)@x_aug@y# write code
here

# By Gradien descent

def Regression_grad_des(self,x,y,lr):

    w = np.reshape(np.zeros(x.shape[0]),(x.shape[0],1))

    err = []

    for i in range(100000):

        # write code here

        err.append(self.error(w,y,x))

        w = self.grad_update(w,lr,y,x)

        dev=np.abs(self.error(w,y,x)-err[-1])# write code
here)

        # print(i)

        if dev <= 0.000001 :

            break

    w_pred = w

    return w_pred,err
```

Augment the Data and generate optimal weights

In [7]:

```
## Write your Code here
x_aug = np.append(np.ones((1,x.shape[0])),x.T, axis=0)
print('Data shape after augmenting:', x_aug.shape)

# By Gradient descent
lr=0.01
w_pred,err=reg.Regression_grad_des(x_aug,y,lr)
print("Optimal weights obatained by Gradient descent:
",w_pred.T)
```

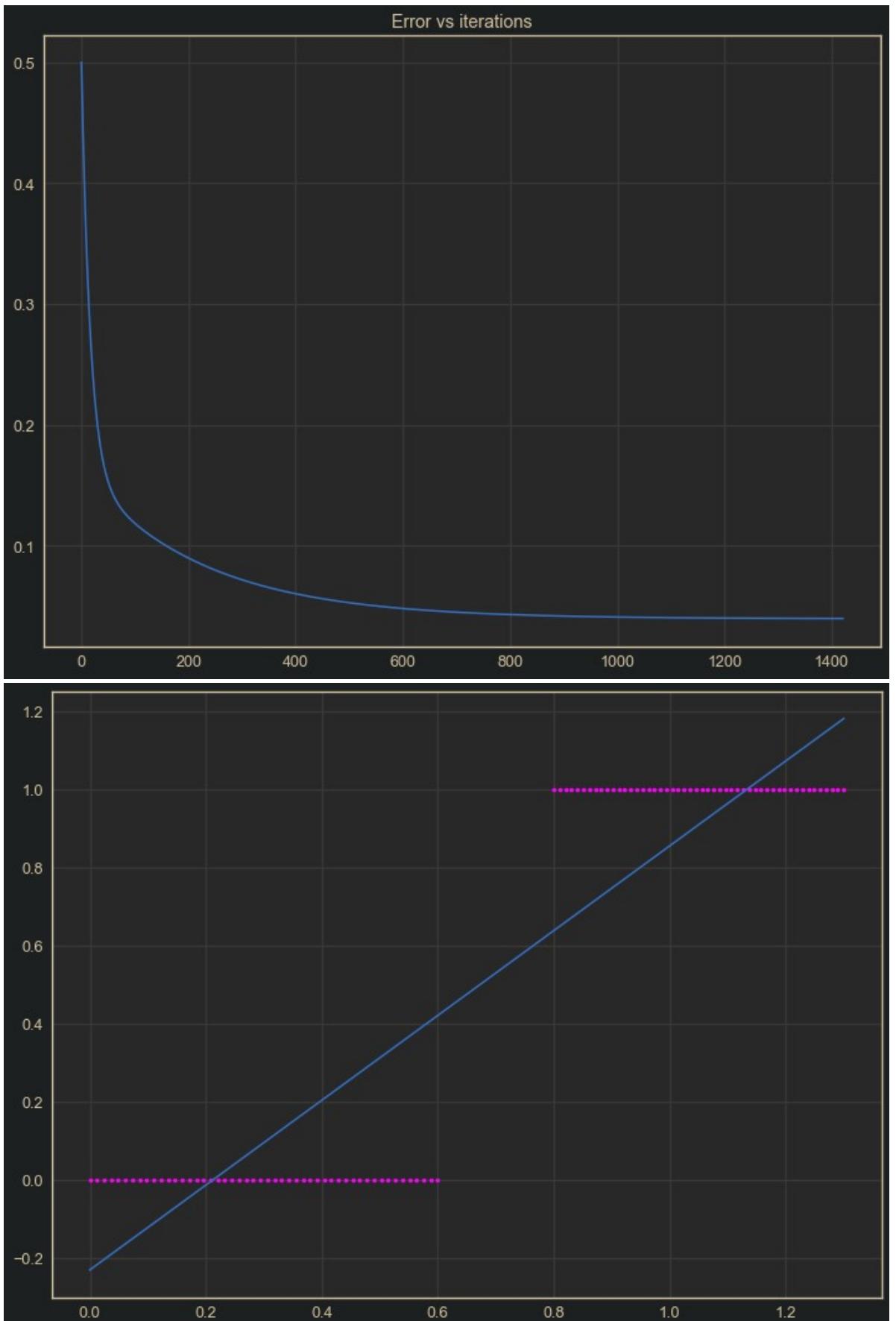
Data shape after augmenting: (2, 100)
Optimal weights obatained by Gradient descent: [-0.23019384 1.0862383]

Using the optimal weights, fit the curve

In [8]:

```
## Write your Code here
plt.figure(figsize=(12, 9))
plt.plot(err);

plt.figure(figsize=(12, 9))
plt.plot(x,y, '.', c='magenta');
x2 = np.linspace(0, np.max(x), 100)
x2 = x2[:,np.newaxis]
x2_aug = np.append(np.ones((1,x2.shape[0])),x2.T, axis=0)
y_pred = x2_aug.T@w_pred
plt.plot(x2,y_pred);
```



Drawback of Linear regression based Classification

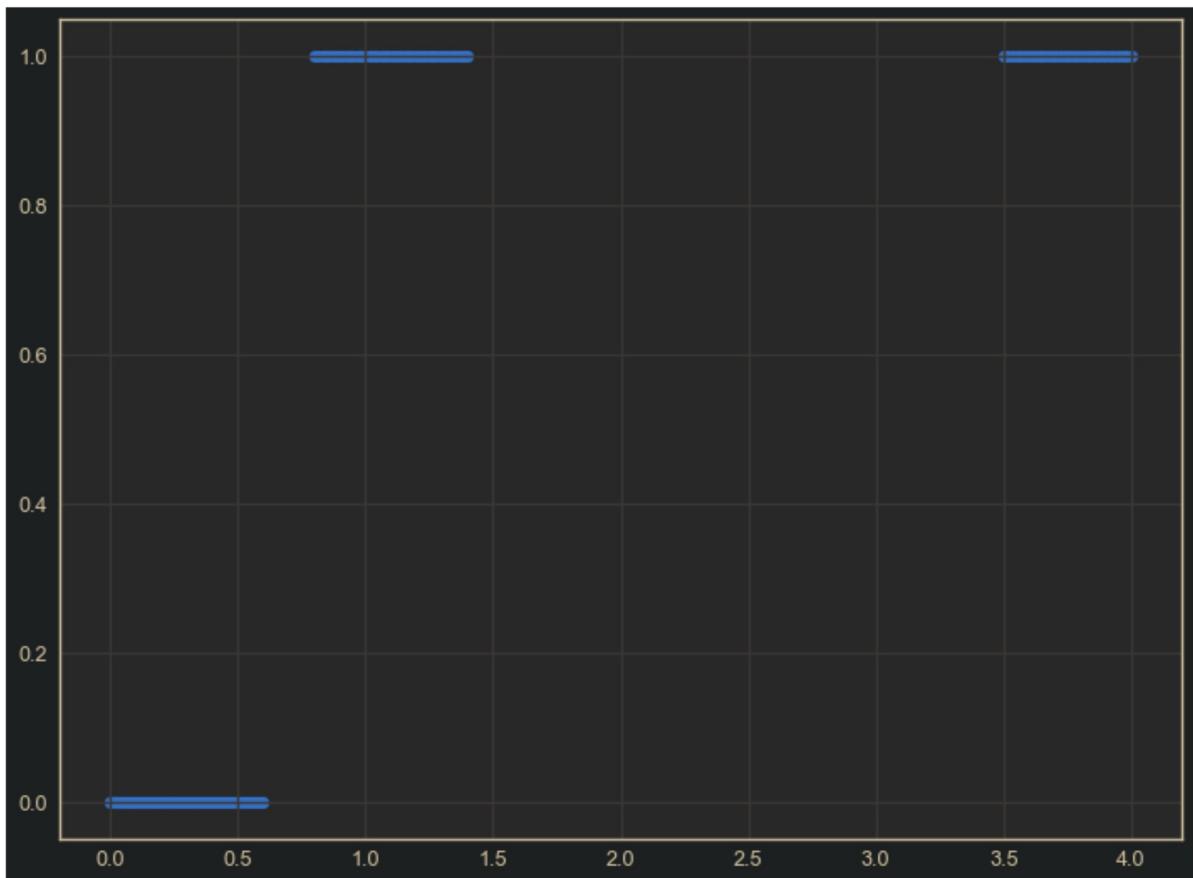
Generate the Data as shown in the figure and follow the same steps as above to fit a curve

using regression class

In [9]:

```
## Write your code here
x = np.append(np.linspace(0,0.6,50),np.linspace(0.8,1.4,40))
x=np.append(x,np.linspace(3.5,4,30))
x = x[:,np.newaxis]
# print(x.shape)
y = np.zeros(x.shape)
# print(y.shape)
y[50:,0] = 1

plt.figure(figsize=(12,9))
plt.scatter(x,y);
```



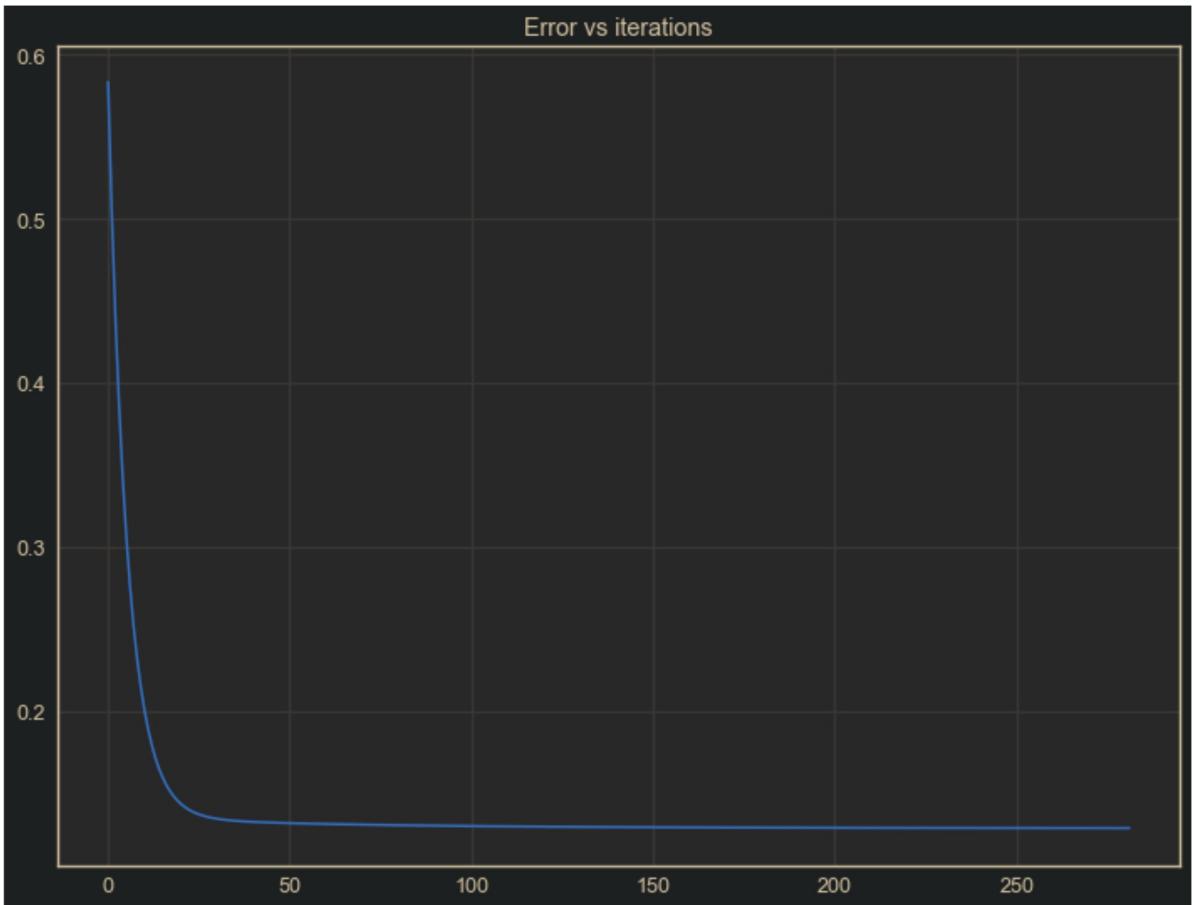
In [10]:

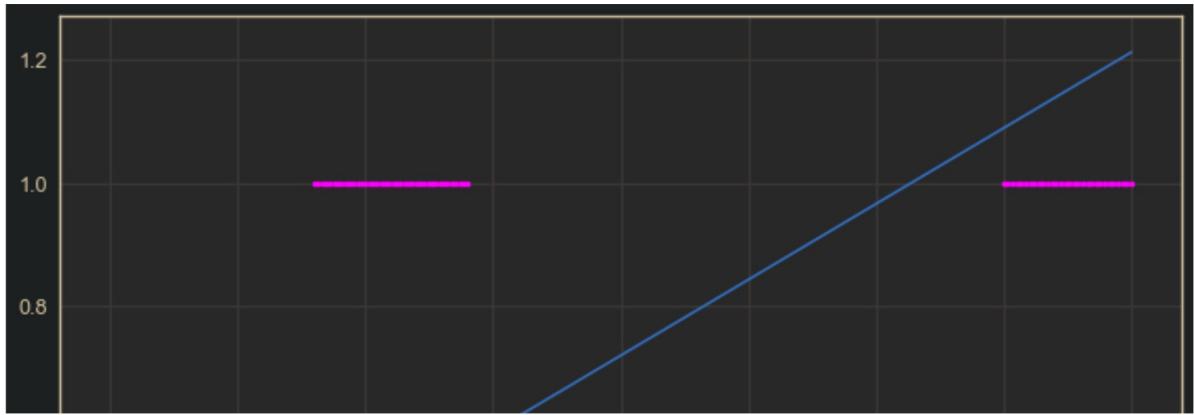
```
## Write your code here
x_aug = np.append(np.ones((1,x.shape[0])),x.T, axis=0)
print('Data shape after augmenting:',x_aug.shape)

# By Gradient descent
lr=0.01
w_pred,err=reg.Regression_grad_des(x_aug,y,lr)
print("Optimal weights obtained by Gradient descent:
",w_pred.T)
```

Data shape after augmenting: (2, 120)
Optimal weights obatained by Gradient descent: [[0.22648388 0.24647292]]

```
In [11]:  
plt.figure(figsize=(12,9))  
plt.plot(err);  
plt.title("Error vs iterations");  
  
plt.figure(figsize=(12,9))  
plt.plot(x,y,'.',c='magenta');  
x2 = np.linspace(0,np.max(x),100)  
x2 = x2[:,np.newaxis]  
x2_aug = np.append(np.ones((1,x2.shape[0])),x2.T, axis=0)  
y_pred = x2_aug.T@w_pred  
plt.plot(x2,y_pred);
```





Logistic regression

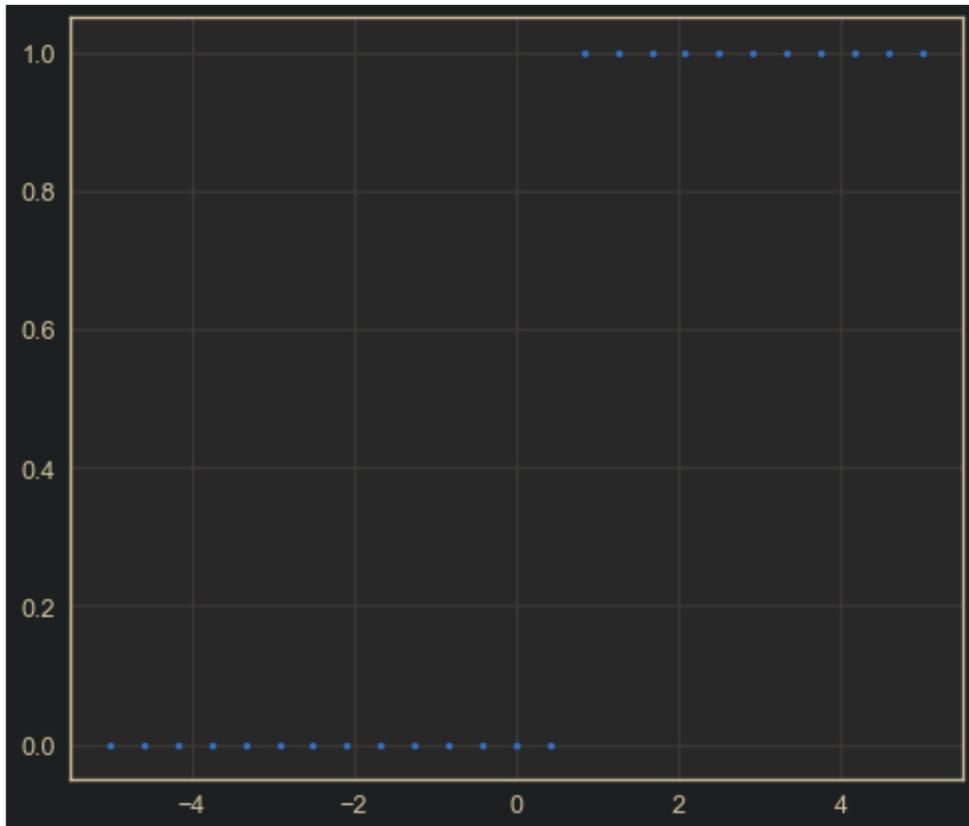
Error Surface (Comparison between Logistic Loss and Mean Squared Error)

In [12]:

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-5,5,25)
y=np.zeros(x.shape)
y[np.where(x>0.7314)]=1

plt.plot(x,y,'.');
```



1. $\text{MSE} = \frac{1}{2N} \sum_{i=1}^N (y_i^p - y_i)^2$, where $y^p = \frac{1}{1+e^{-w^T x}}$
2. $\text{Logistic loss} = -\frac{1}{N} \sum_{i=1}^N y_i \log(y_i^p) + (1 - y_i) \log(1 - y_i^p)$

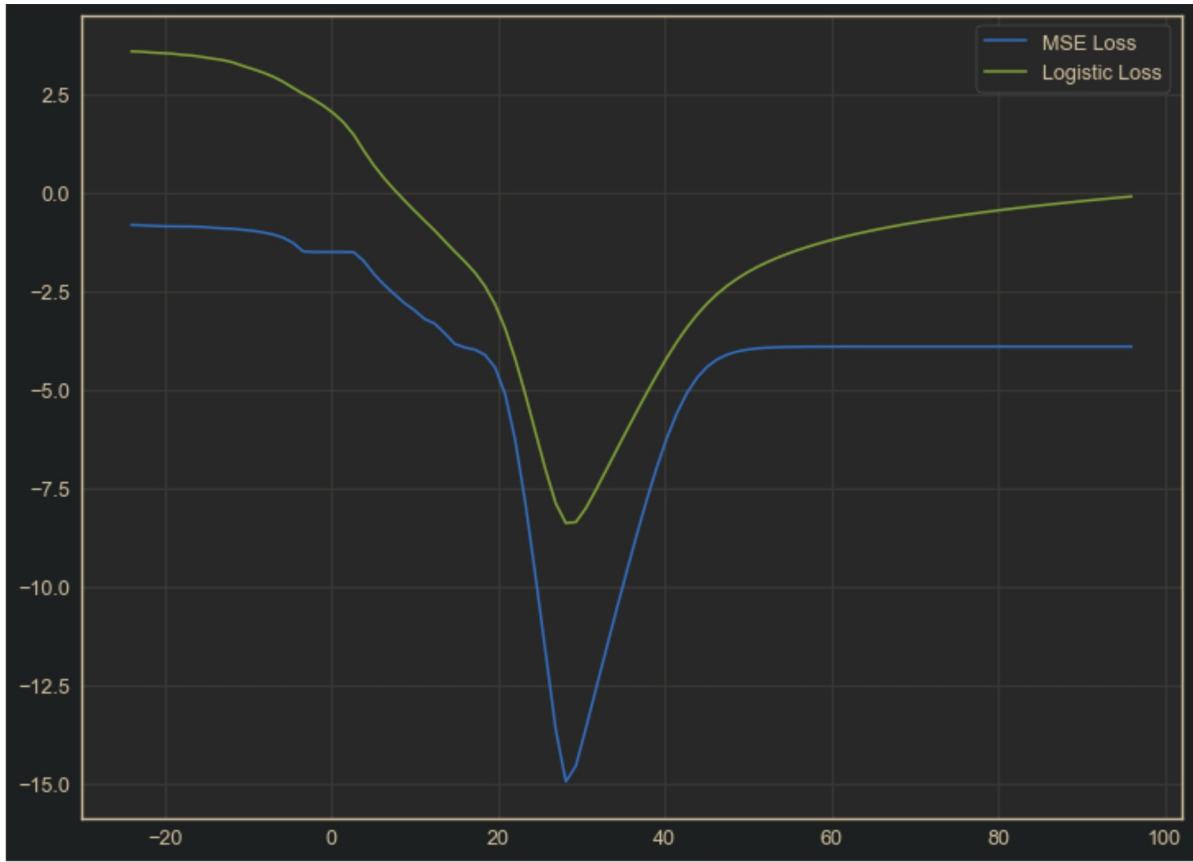
In [13]:

```
# search space (only w1 is searched, where as w0 is fixed)
w1_in=10/(x[1]-x[0])
w0=-w1_in*0.7314
w1=np.linspace(-w1_in,4*w1_in,100)

cost_fn_mse=[]
cost_fn_logis=[]
for i in range(w1.shape[0]):
    # Compute Mean square error and logistic loss using cost
    function
    # Write your code here
    y_pred = 1/(1+np.exp(-w0-w1[i]*x));
    cost_fn_mse = np.append(cost_fn_mse,np.average((y-
    y_pred)**2)/2);
    cost_fn_logis = np.append(cost_fn_logis,-
    1*np.average(y*np.log(y_pred+1e-20)+(1-y)*np.log(1-y_pred+1e-
    20)));
```

In [14]:

```
# Ploting of error surface
plt.figure(figsize=(12,9))
plt.plot(w1,np.log(cost_fn_mse),label='MSE Loss');
plt.plot(w1,np.log(cost_fn_logis),label ='Logistic Loss');
plt.legend();
```



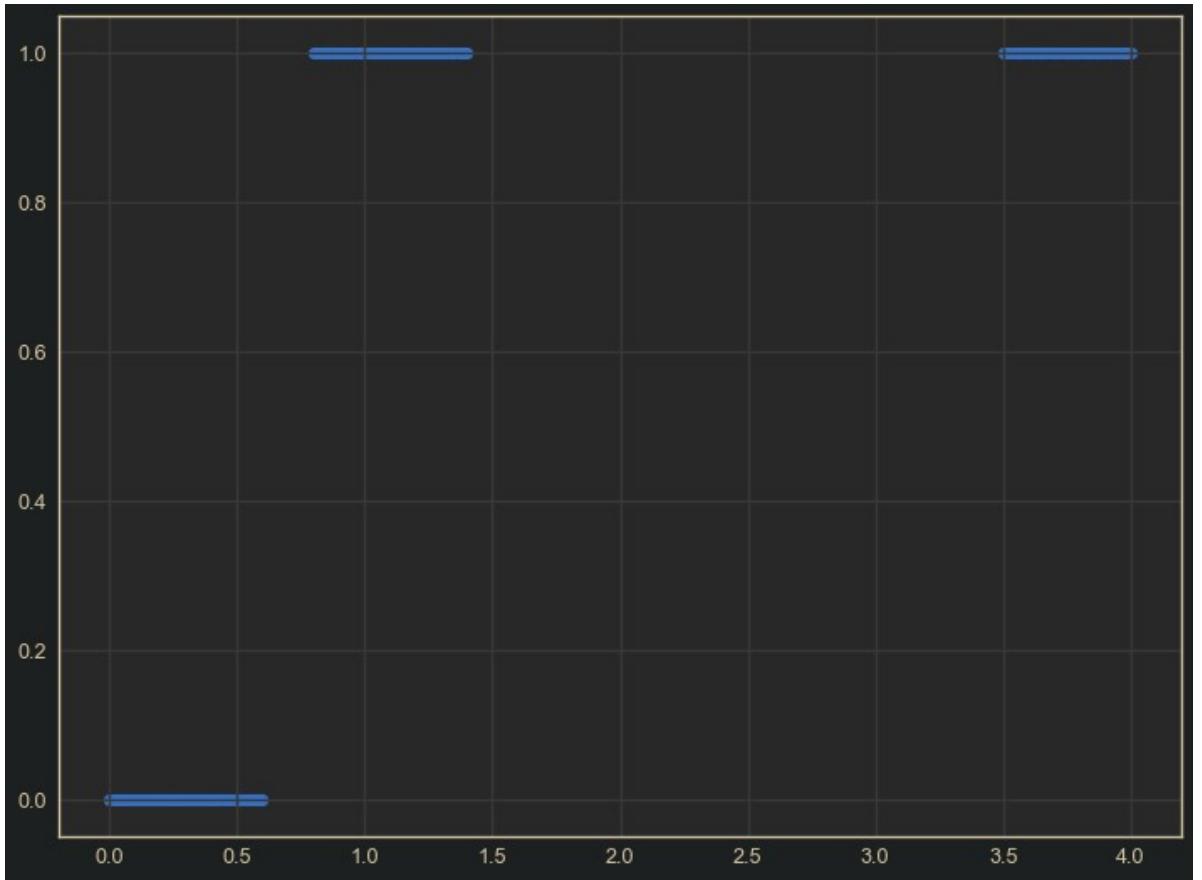
Solving the Outlier Issue

Generate the Data as shown in the figure

In [15]:

```
## Write your Code here
x = np.append(np.linspace(0, 0.6, 50), np.linspace(0.8, 1.4, 40))
x=np.append(x,np.linspace(3.5,4,30))
x = x[:,np.newaxis]
# print(x.shape)
y = np.zeros(x.shape)
# print(y.shape)
y[50:,0] = 1

plt.figure(figsize=(12, 9))
plt.scatter(x,y);
```



Define a Logistic Regression class

In [16]:

```
class logis_regression:  
    # Constructor  
  
    def __init__(self, name='reg'):  
        self.name = name # Create an instance variable  
  
  
    def logis(self,x,w_old):  
        # write code here  
        op = 1/(1+np.exp(-1*x@w_old));  
        return op  
  
  
    def grad_update(self,w_old,lr,y,x):  
        # write code here  
        w = w_old - lr*x.T@(self.logis(x,w_old)-y)/x.shape[0];  
        return w  
  
  
    def error(self,w,y,x):  
        return -1*np.average(y*np.log(self.logis(x,w)+1e-20)+(1-y)*np.log(1-self.logis(x,w)+1e-20)) # write code here  
  
  
    def Regression_grad_des(self,x,y,lr):  
        w=np.zeros((x.shape[1],1));  
        err=[];  
        for i in range(1000):  
            # write code here  
            err.append(self.error(w,y,x));  
            w = self.grad_update(w,lr,y,x);  
            dev=np.abs(self.error(w,y,x)-err[-1]);  
            if dev<=10**(-20):  
                break  
        w_pred = w;  
        return w_pred,err
```

Augment the data and fit the curve by obtaining optimal weights (Using Gradient Descent)

In [17]:

```
## Write your code here

x_aug = np.append(np.ones((x.shape[0], 1)), x, axis=1)
print('Data shape after augmenting:', x_aug.shape)

lreg=logis_regression()
lr=0.1

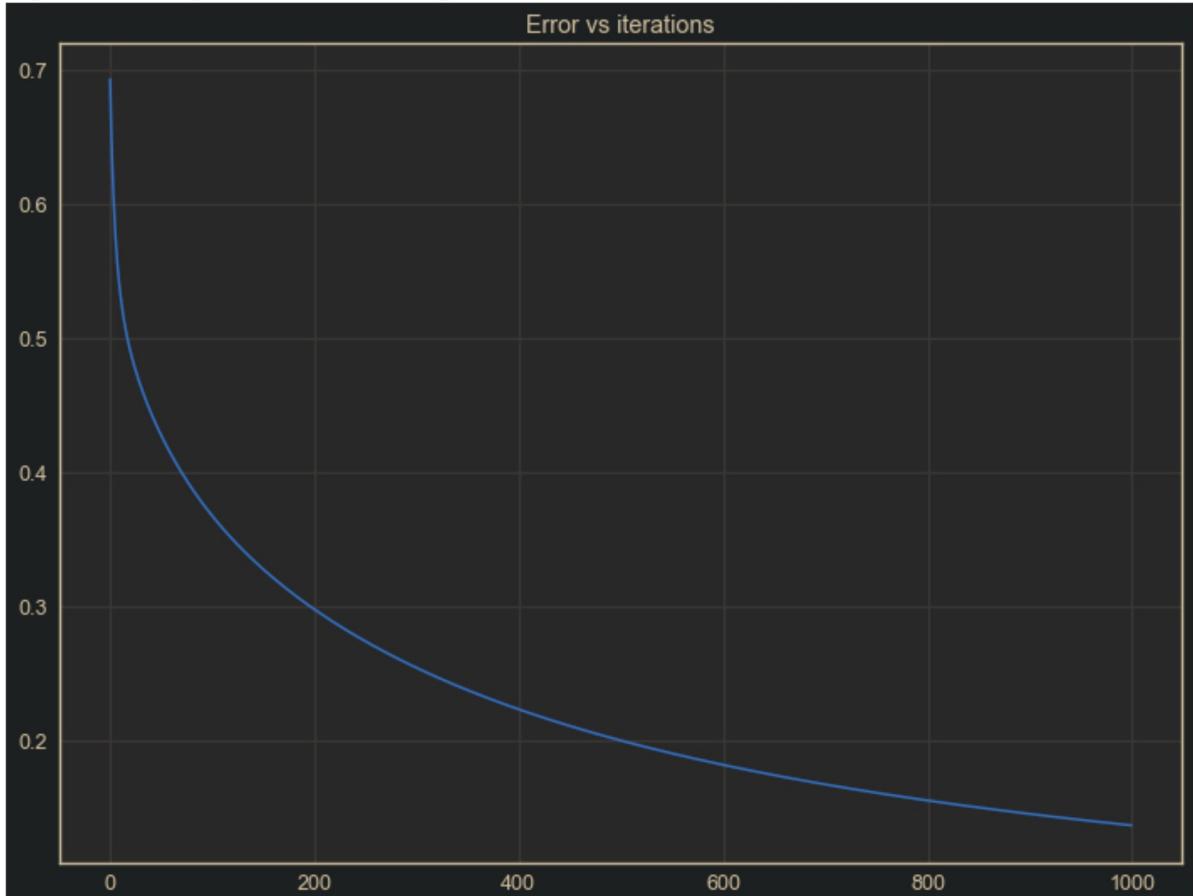
w_pred,err=lreg.Regession_grad_des(x_aug,y,lr)
print("Optimal weights obatained by Gradient descent:
",w_pred.T)

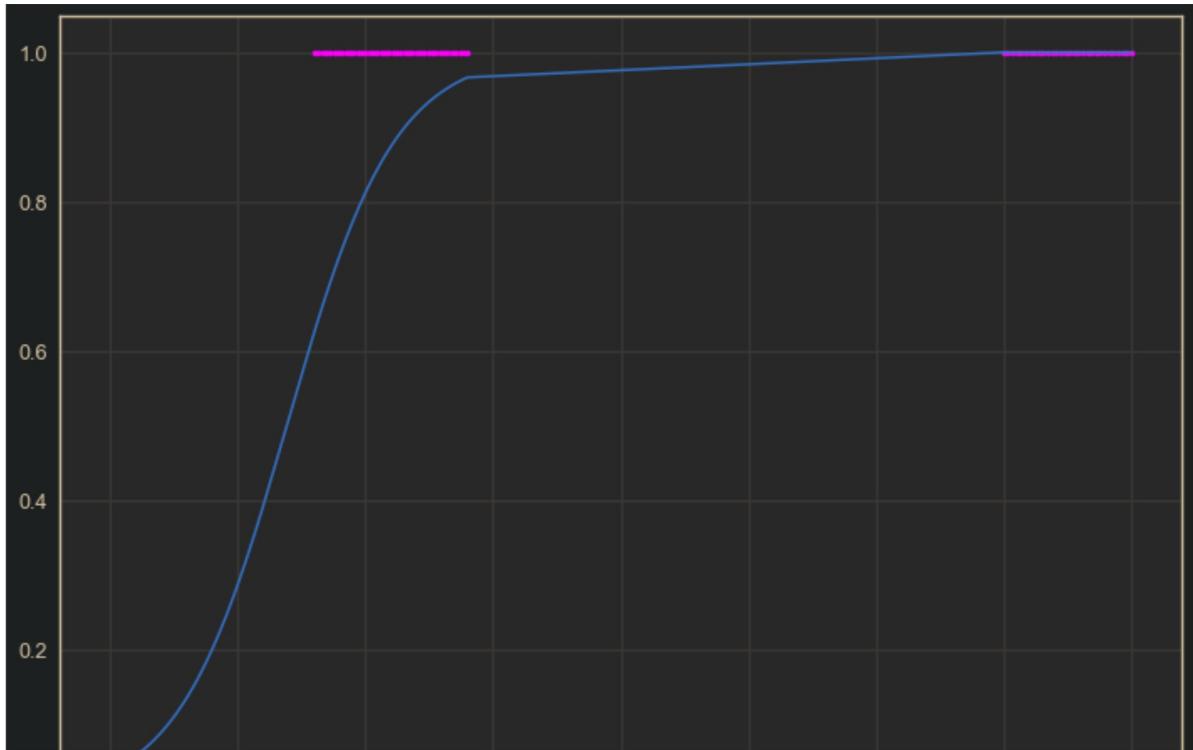
plt.figure(figsize=(12, 9))
plt.plot(err);
plt.title("Error vs iterations");

y_pred = lreg.logis(x_aug,w_pred)
plt.figure(figsize=(12, 9))
plt.plot(x,y,'.',c='magenta');
plt.plot(x,y_pred);
```

Data shape after augmenting: (120, 2)

Optimal weights obatained by Gradient descent: [-3.2882017 4.74809518]



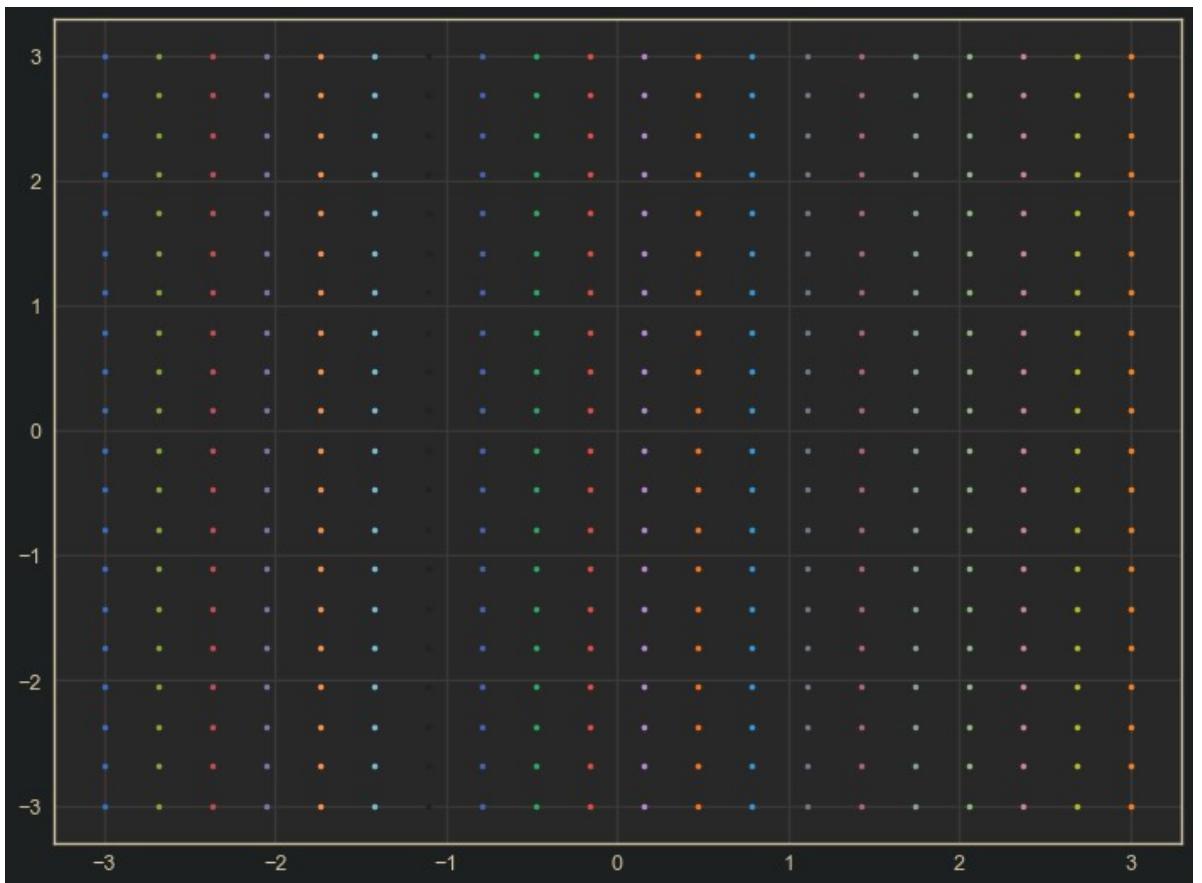


Classification of circularly separated data using logistic regression

In [18]:

```
x1=np.linspace(-3,3,20)
x2=np.linspace(-3,3,20)

x11,x22=np.meshgrid(x1,x2)
plt.figure(figsize=(12,9))
plt.plot(x11,x22,'.');
```



Using the above data generate circular data

In [19]:

```
# Write code here

x11f = x11.flatten()
x22f = x22.flatten()

x1f=[]
x2f=[]
labels=[]

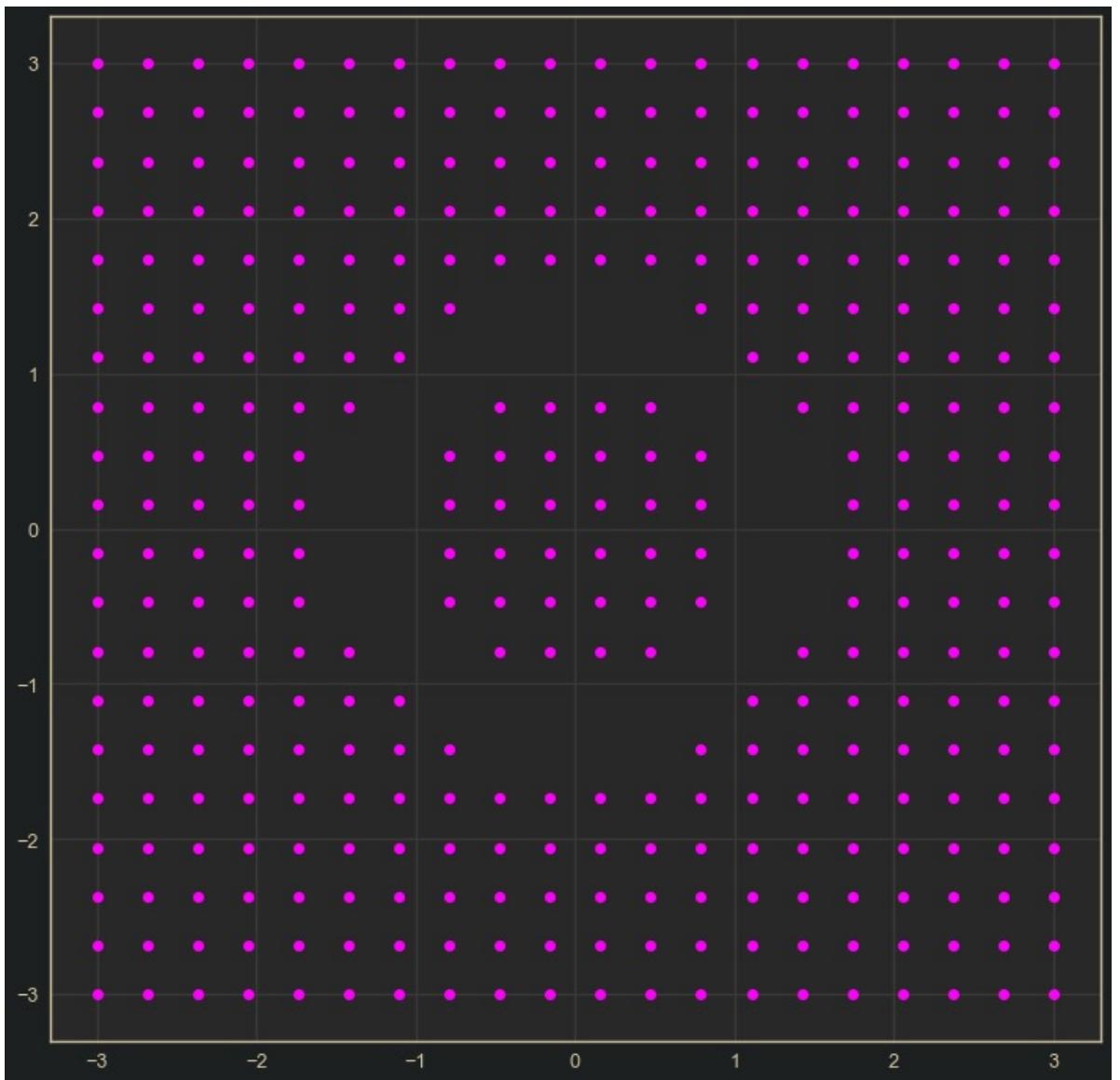
# plt.figure(figsize=(12,9))
# plt.plot(x11f,x22f,'.');
for i in range(len(x11f)):

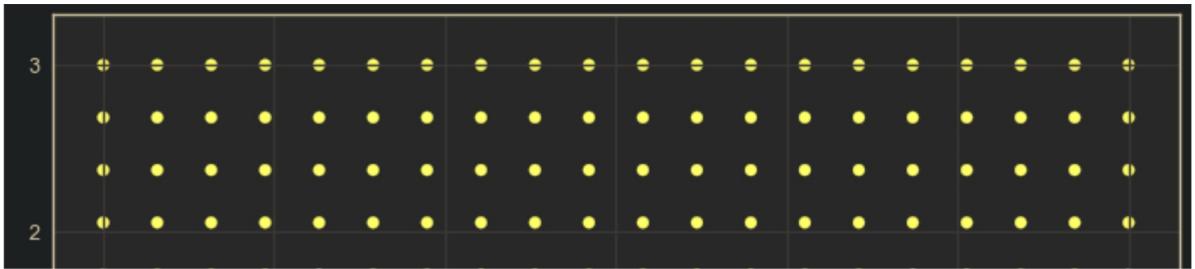
    if x11f[i]**2+x22f[i]**2<=1:
        x1f.append(x11f[i]);
        x2f.append(x22f[i]);
        labels=np.append(labels,0);

    elif x11f[i]**2+x22f[i]**2>=2.25:
        x1f.append(x11f[i]);
        x2f.append(x22f[i]);
        labels=np.append(labels,1);

labels=labels[:,np.newaxis];
# print(labels.shape);
plt.figure(figsize=(12,12))
plt.plot(x1f,x2f,'o',c='magenta');

plt.figure(figsize=(12,12))
plt.scatter(x1f,x2f,c=labels,cmap='summer');
```





As in case of circularly separated data, the boundary is nonlinear, so squared feature is taken.

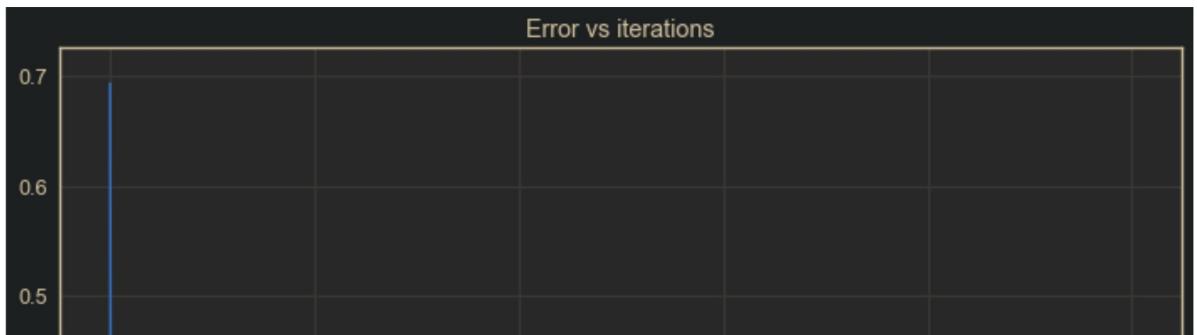
In [20]:

```
# perform logistic regression
x = np.append([x1f], [x2f], axis=0).T;
# print(x.shape)
x_aug = np.empty((x.shape[0], 0))
for i in np.arange(0, 3):
    for j in np.arange(0, 3-i):
        x_aug=np.append(x_aug, ((x[:, 0]**i)*(x[:, 1]**j))[:, np.newaxis], axis=1);
# print(x_aug.shape);
# print(x_aug[0:5,:]);

lreg=logis_regression();
lr=0.3;
w_pred,err=lreg.Regession_grad_des(x_aug,labels,lr);
print("Optimal weights obatained by Gradient descent:
",w_pred.T);

plt.figure(figsize=(12,9))
plt.plot(err);
plt.title("Error vs iterations");
```

Optimal weights obatained by Gradient descent: [[-3.59338135e+00 1.068734
94e-15 2.40122485e+00 1.05907251e-15
1.01771017e-16 2.40122485e+00]]

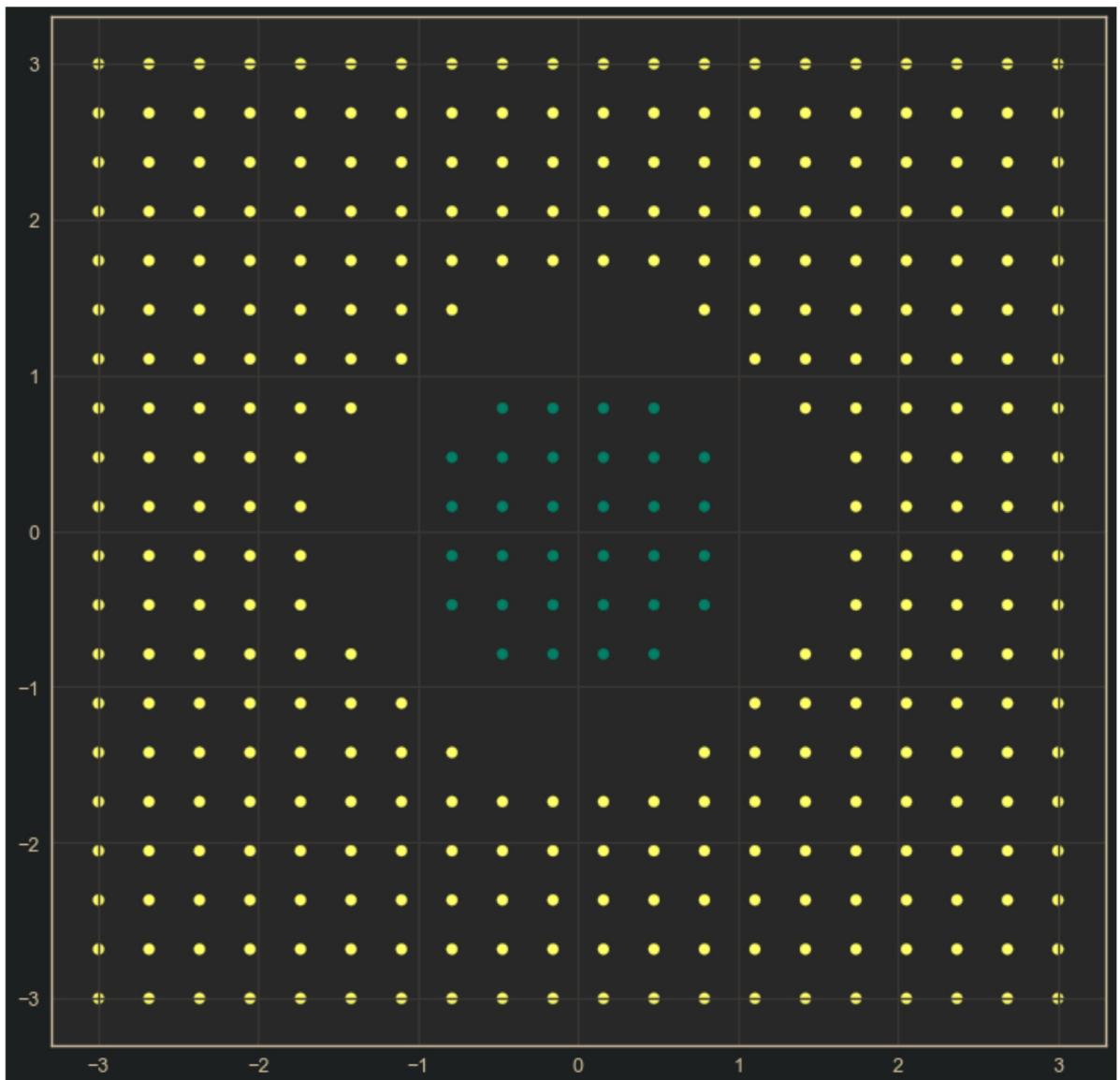


Plot classification using 0.5 as threshold

In [21]:

```
#write code here
y_pred = lreg.logis(x_aug,w_pred)
label_pred = y_pred>0.5;

plt.figure(figsize=(12,12))
plt.scatter(x1f,x2f,c=label_pred,cmap='summer');
```



Multiclass logistic regression

1. Generate 1D data with 3 classes

One vs rest classification

1. Lets take a polynomial of order 2 (by seeing the data distribution)

In [22]:

```
## Write your code here

import numpy as np
import matplotlib.pyplot as plt

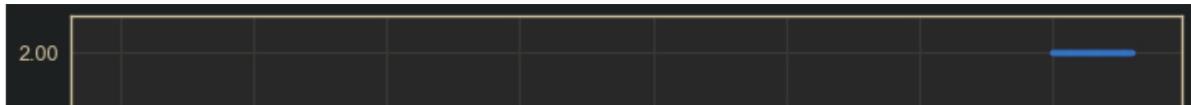
x1=np.linspace(0,0.6,100)
x2=np.linspace(1.1,2.7,100)
x3=np.linspace(3.5,3.8,100)

x=np.concatenate((x1,x2,x3))[:,np.newaxis]
print(x.shape)

y1=np.zeros(x1.shape)
y2=np.ones(x2.shape)
y3=np.tile([2],x3.shape)

y=np.concatenate((y1,y2,y3))[:,np.newaxis]
print(y.shape)
plt.figure(figsize=(12,9))
plt.plot(x,y,'.') ;
```

```
(300, 1)
(300, 1)
```



In [23]:

```
# write code here to generate a polynomial

def data_transform(X,degree):
    X_new=np.empty((X.shape[0],0));
    for i in range(degree +1):
        X_new=np.append(X_new,X**i, axis=1);
    return X_new

x_aug=data_transform(x, 2)
```

In [24]:

```
# plot for classification

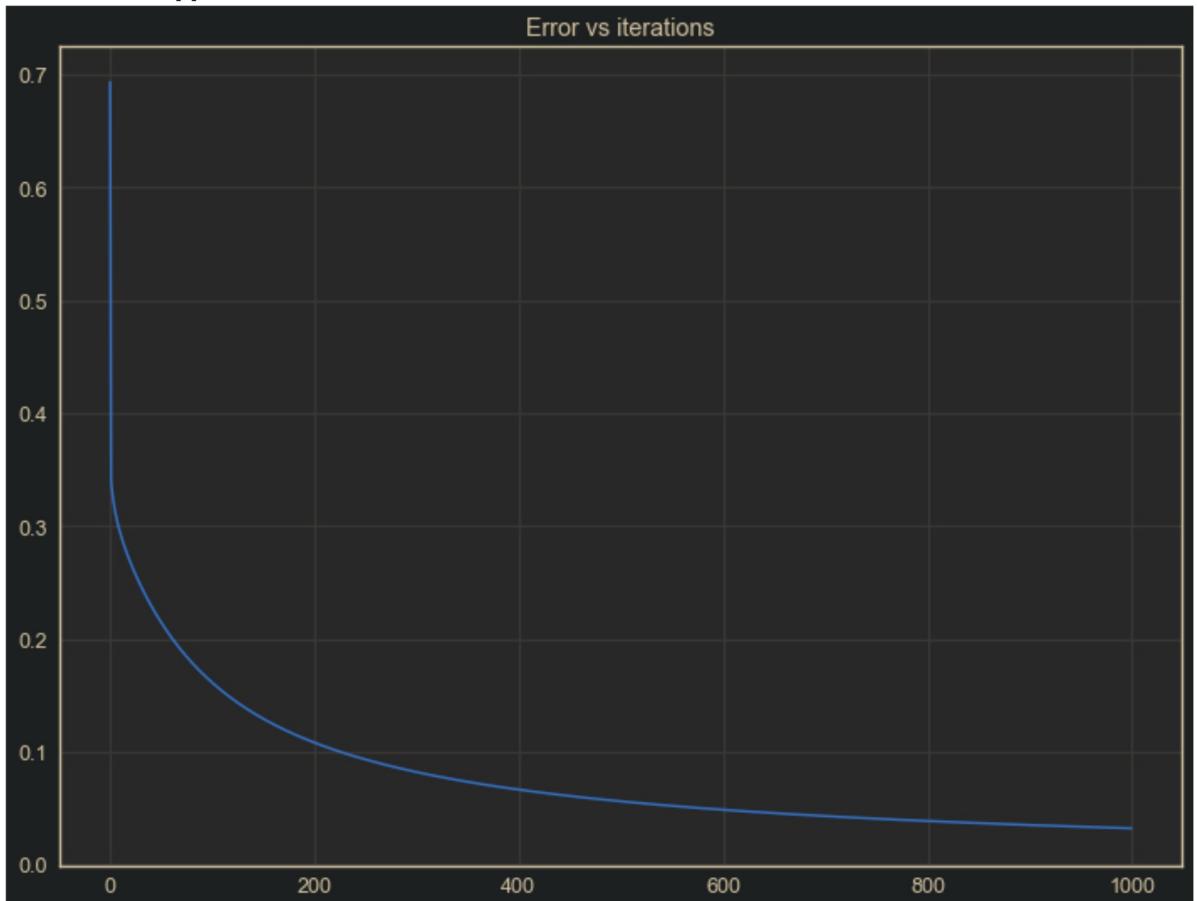
def plot_op(x,y_pred):
    ind0,_=np.where(y_pred<0.5);
    ind1,_=np.where(y_pred>=0.5);
    x0=x[ind0,:];
    x1=x[ind1,:];
    plt.figure(figsize=(12,9));
    plt.plot(x0,np.zeros(x0.shape), 'o', color='y');
    plt.plot(x1,np.ones(x1.shape), '.', color='r');
```

Using the above function for plotting, plot the curve using different configurations

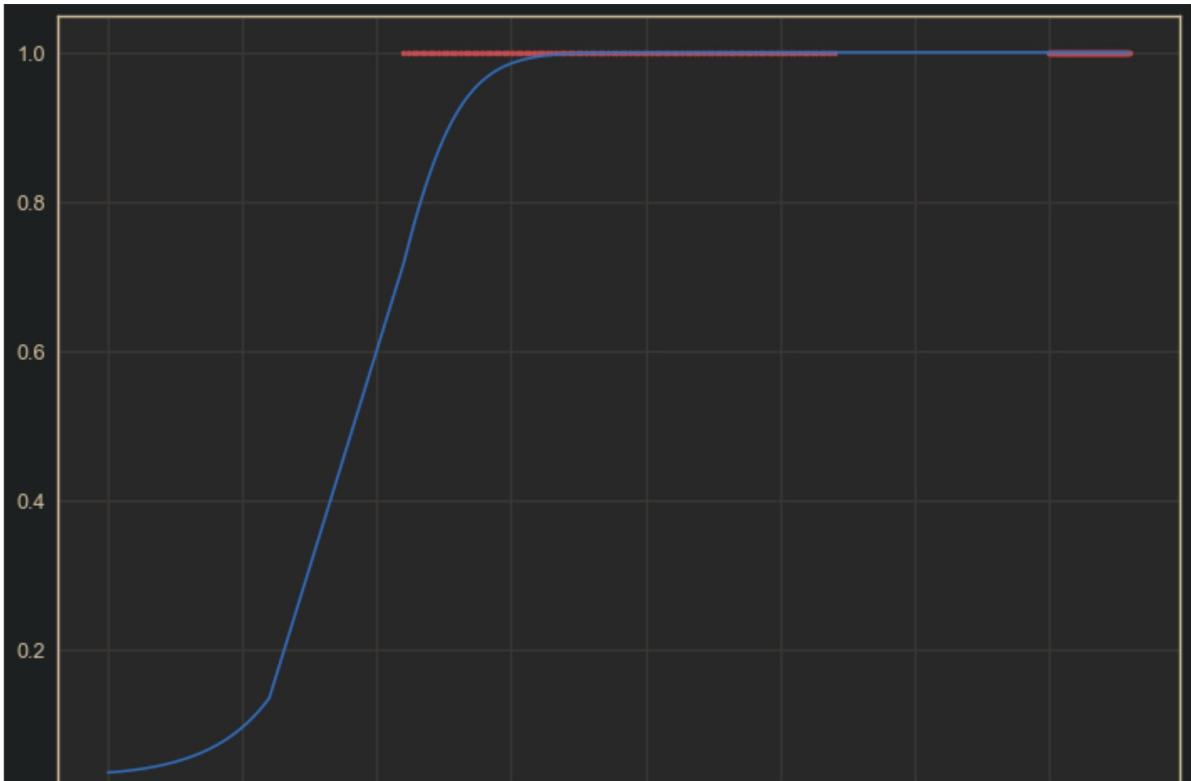
In [25]:

```
# take class 0 as '0' and other to '1'  
## Write your code here  
x0 = x;  
y0 = y >0.1;  
  
lreg=logis_regression()  
lr=0.1  
w0,err=lreg.Regression_grad_des(x_aug,y0,lr);  
print("Optimal weights obatained by Gradient descent: ",w0.T)  
  
plt.figure(figsize=(12, 9));  
plt.plot(err);  
plt.title("Error vs iterations");  
  
y_pred = lreg.logis(x_aug,w0);  
plt.figure(figsize=(12, 9));  
plot_op(x0,y0);  
plt.plot(x0,y_pred);
```

Optimal weights obatained by Gradient descent: [[-3.33922596 0.74779595
2.84819221]]



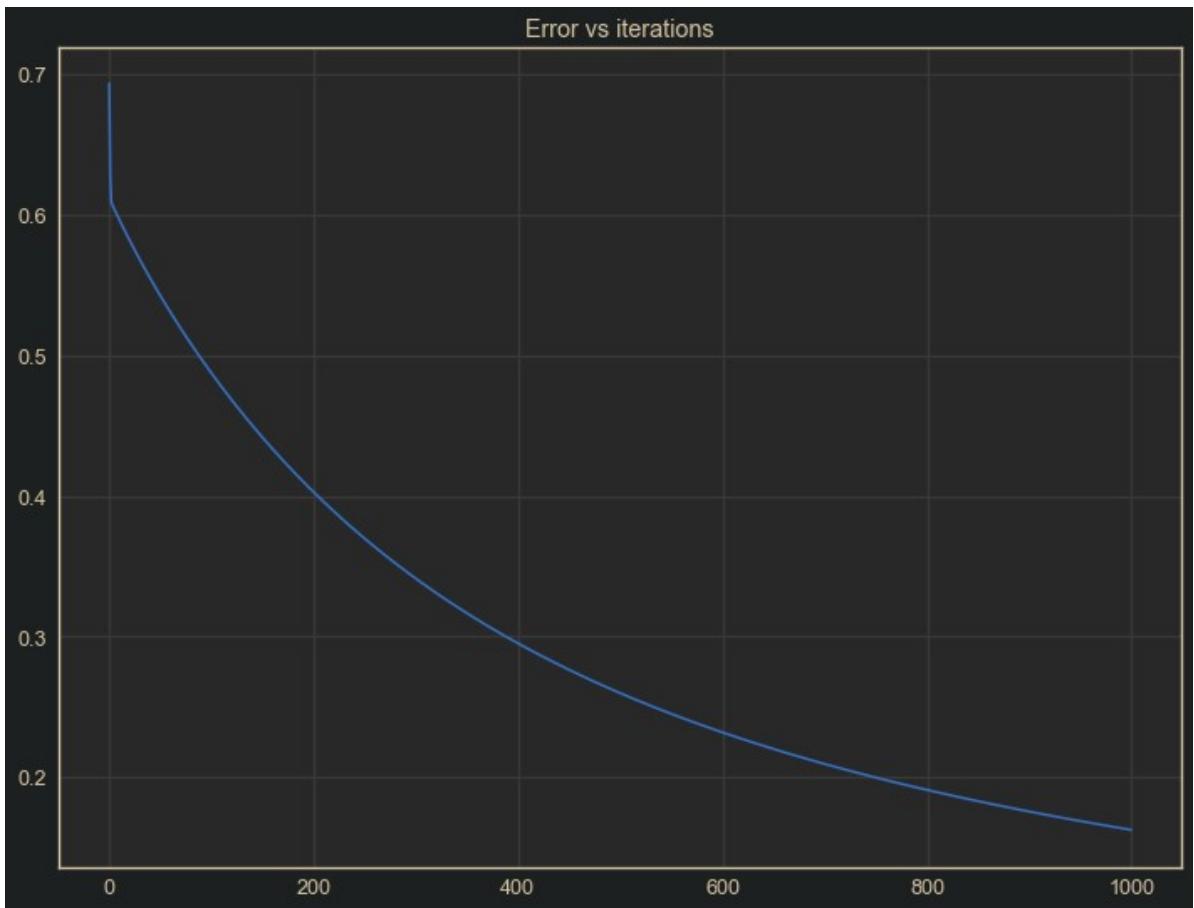
<Figure size 864x648 with 0 Axes>



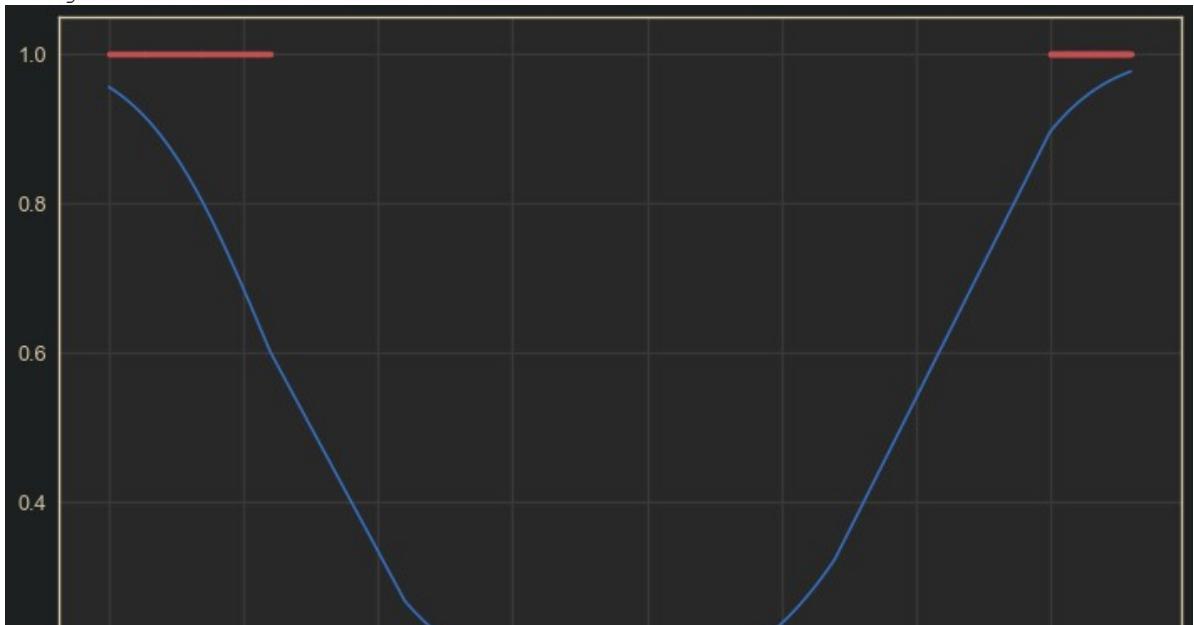
In [26]:

```
# take class 1 as '0' and other to '1'  
## Write your code here  
x1 = x;  
y1 = (y>1.2)+(y<0.8);  
  
lreg=logis_regression()  
lr=0.1  
w1,err=lreg.Regession_grad_des(x_aug,y1,lr);  
print("Optimal weights obatained by Gradient descent: ",w1.T)  
  
plt.figure(figsize=(12, 9));  
plt.plot(err);  
plt.title("Error vs iterations");  
  
y_pred = lreg.logis(x_aug,w1);  
plt.figure(figsize=(12, 9));  
plot_op(x1,y1);  
plt.plot(x1,y_pred);
```

```
Optimal weights obatained by Gradient descent: [[ 3.04807476 -5.26692489  
1.43081929]]
```



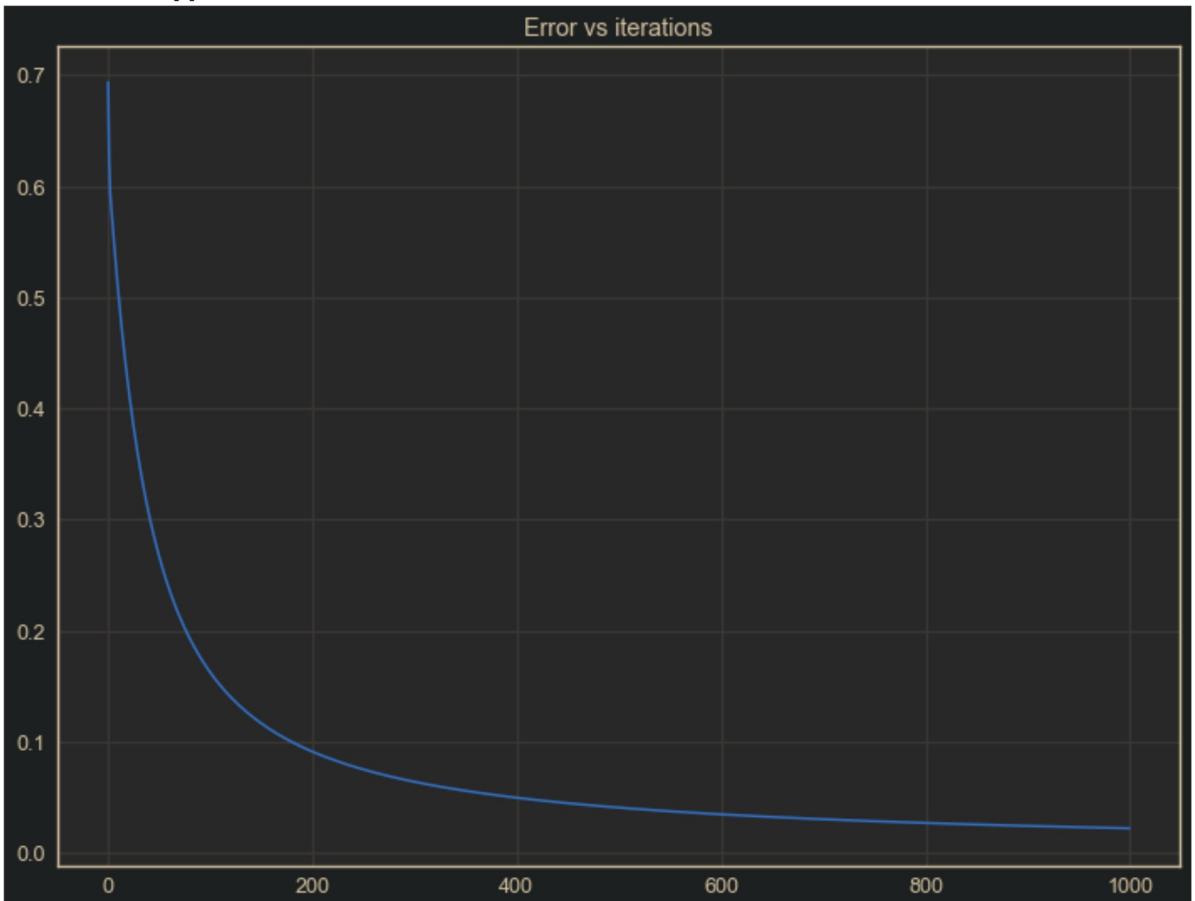
<Figure size 864x648 with 0 Axes>



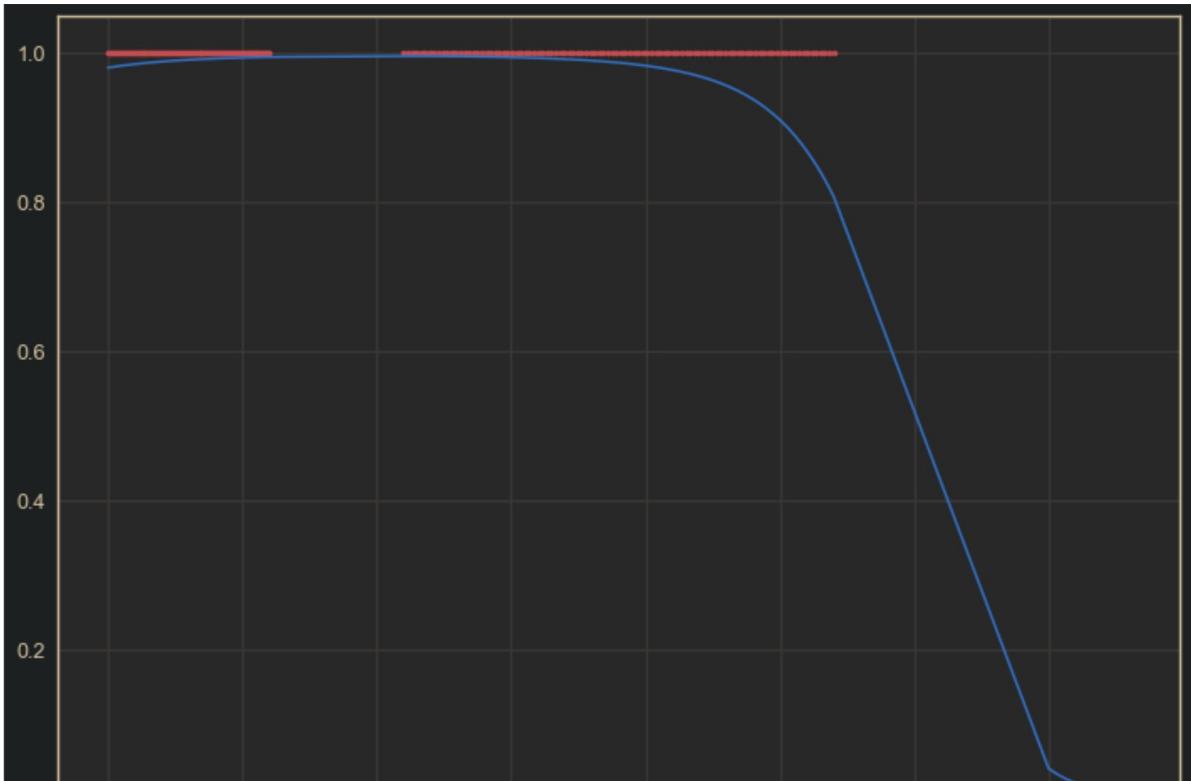
In [27]:

```
# Take class 2 as '0' and other to '1'  
## Write your code here  
x2 = x;  
y2 = y <1.9;  
  
lreg=logis_regression()  
lr=0.1  
w2,err=lreg.Regession_grad_des(x_aug,y2,lr);  
print("Optimal weights obatained by Gradient descent: ",w2.T)  
  
plt.figure(figsize=(12, 9));  
plt.plot(err);  
plt.title("Error vs iterations");  
  
y_pred = lreg.logis(x_aug,w2);  
plt.figure(figsize=(12, 9));  
plot_op(x2,y2);  
plt.plot(x2,y_pred);
```

Optimal weights obatained by Gradient descent: [[3.86508203 2.85058119 -1.38961688]]



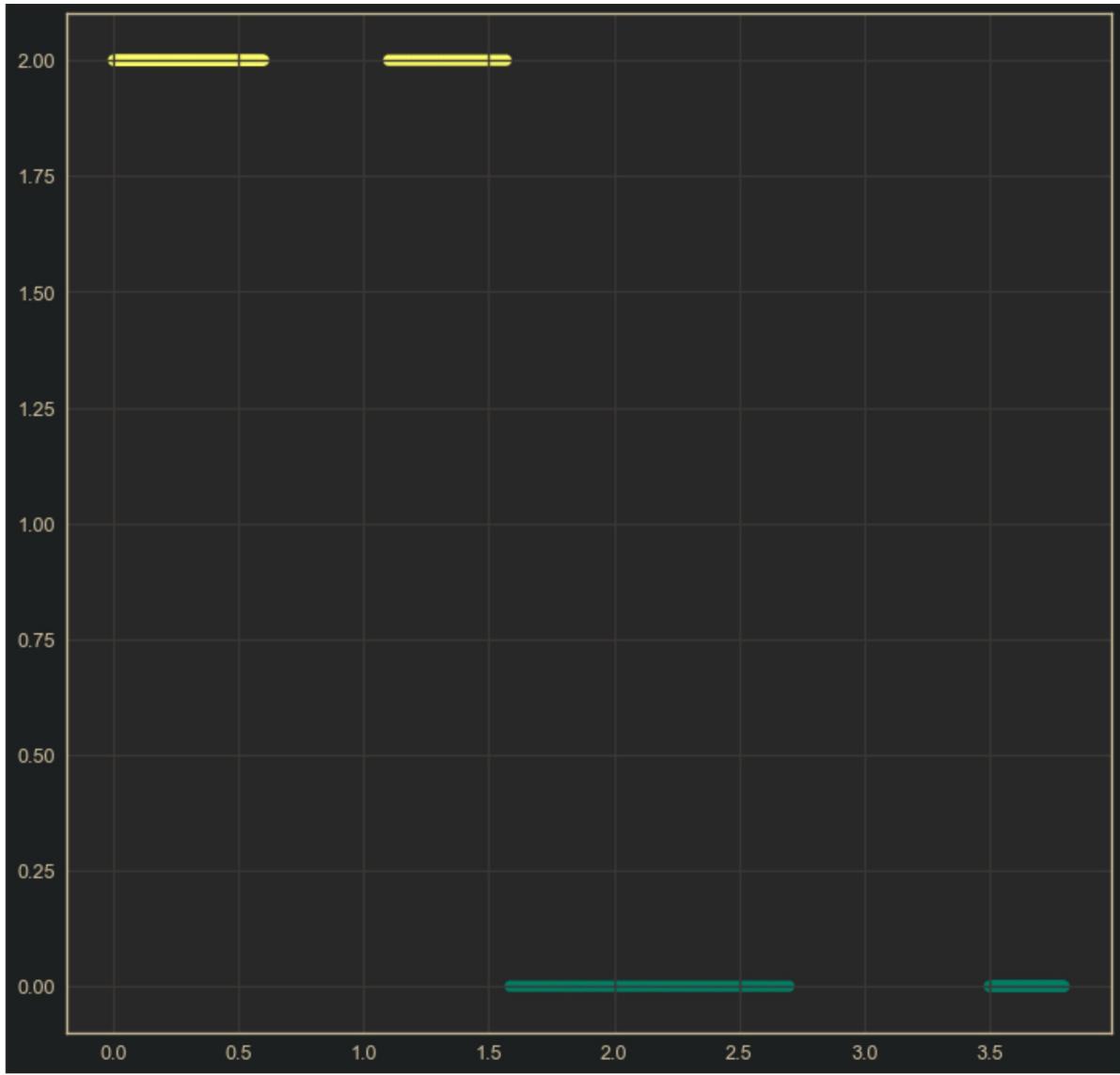
<Figure size 864x648 with 0 Axes>



In [29]:

```
# final classification
## Write your code here
y0 = lreg.logis(x_aug,w0)
y1 = lreg.logis(x_aug,w1)
y2 = lreg.logis(x_aug,w2)
y=np.append(y0,y1, axis=1)
y=np.append(y, y2, axis=1)

y_pred=np.max(y, axis=1)
y_lab=np.argmax(y, axis=1)
# print(y_pred.shape)
# print(y_lab.shape)
plt.figure(figsize=(12,12))
plt.scatter(x,y_lab,c=y_lab,cmap='summer');
```



In []: