**The LNM Institute of Information Technology, Jaipur**
**CSE4121: Deep Learning**
**Assignment 1**

Team Members:

1. Abhey Raheja (22UCS003)
2. Akhil Murarka (22UCS009)
3. Lakshya Rawat (22UCS113)

Q1. Implement a deep neural network for a binary classification problem.

- Implement the code in python without using Sequential/Dense libraries. The code should have functions for forward propagation, cost computation and back propagation.
- You can take any suitable dataset. -normalize the data before using it.
- Show scatter plot of data.
- Plot the cost vs no. of epochs.

1) Import the necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.model_selection import train_test_split
```

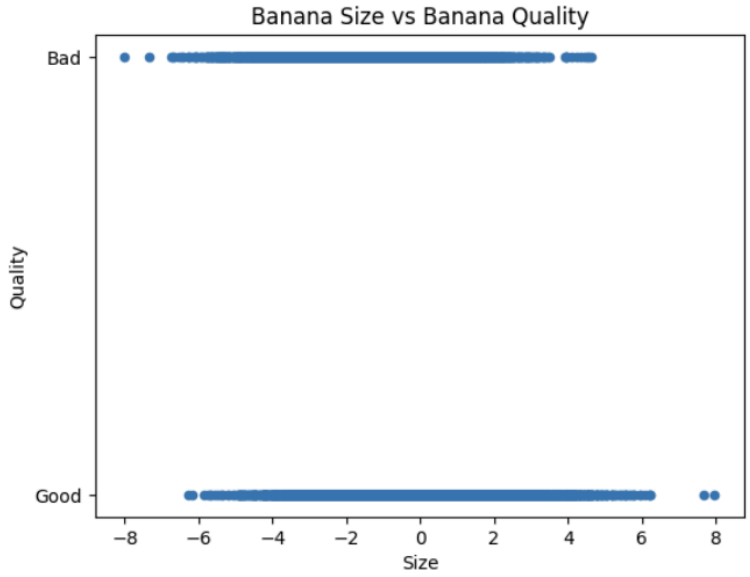
2) Load the dataset (banana quality)

```
csv_file_path='/content/banana_quality.csv'
df=pd.read_csv(csv_file_path)
print(df)
```

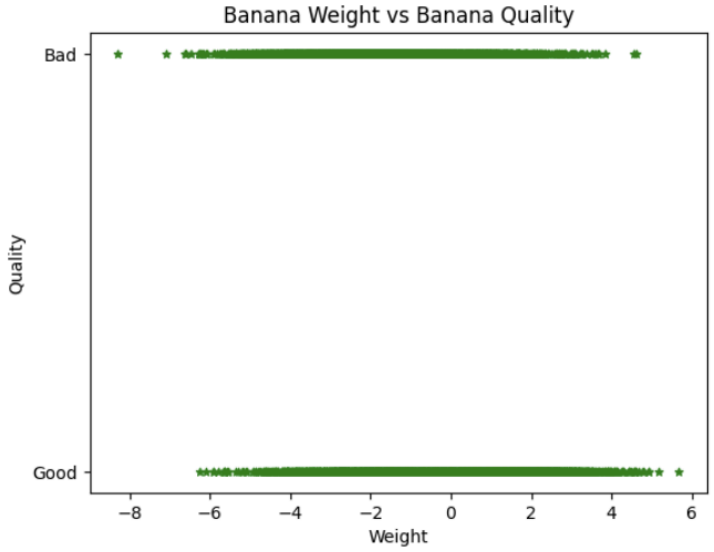
```
          Size    Weight  Sweetness  Softness  HarvestTime  Ripeness  \
0    -1.924968  0.468078   3.077832 -1.472177     0.294799  2.435570
1    -2.409751  0.486870   0.346921 -2.495099    -0.892213  2.067549
2    -0.357607  1.483176   1.568452 -2.645145    -0.647267  3.090643
3    -0.868524  1.566201   1.889605 -1.273761    -1.006278  1.873001
4     0.651825  1.319199  -0.022459 -1.209709    -1.430692  1.078345
...        ...       ...        ...       ...          ...       ...
7995 -6.414403  0.723565   1.134953  2.952763     0.297928 -0.156946
7996  0.851143 -2.217875  -2.812175  0.489249    -1.323410 -2.316883
7997  1.422722 -1.907665  -2.532364  0.964976    -0.562375 -1.834765
7998 -2.131904 -2.742600  -1.008029  2.126946    -0.802632 -3.580266
7999 -2.660879 -2.044666   0.159026  1.499706    -1.581856 -1.605859

       Acidity Quality
0     0.271290    Good
1     0.307325    Good
2     1.427322    Good
3     0.477862    Good
4     2.812442    Good
...        ...     ...
7995  2.398091     Bad
7996  2.113136     Bad
7997  0.697361     Bad
7998  0.423569     Bad
7999  1.435644     Bad

[8000 rows x 8 columns]
```

3) Scatter plot of features with the target variable

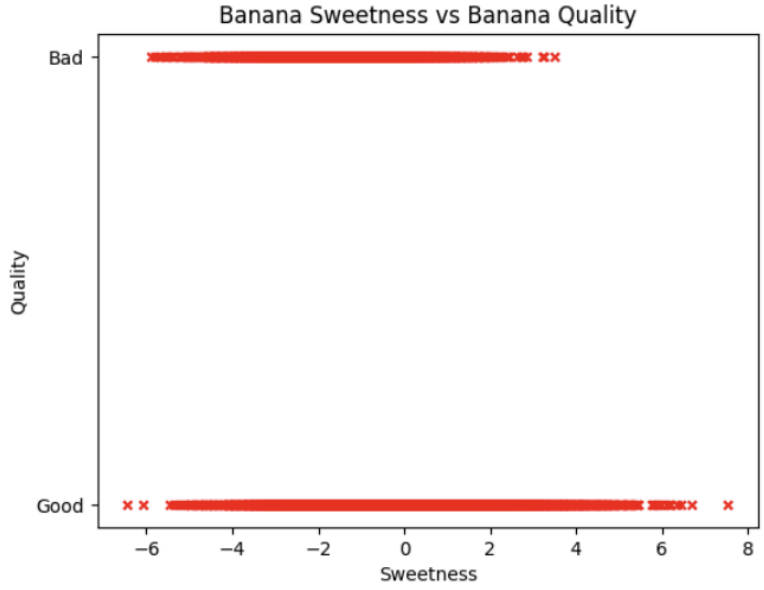
```
plt.scatter(df.Size,df.Quality,s=20,marker='o')
plt.xlabel('Size')
plt.ylabel('Quality')
plt.title('Banana Size vs Banana Quality')
plt.show()
```



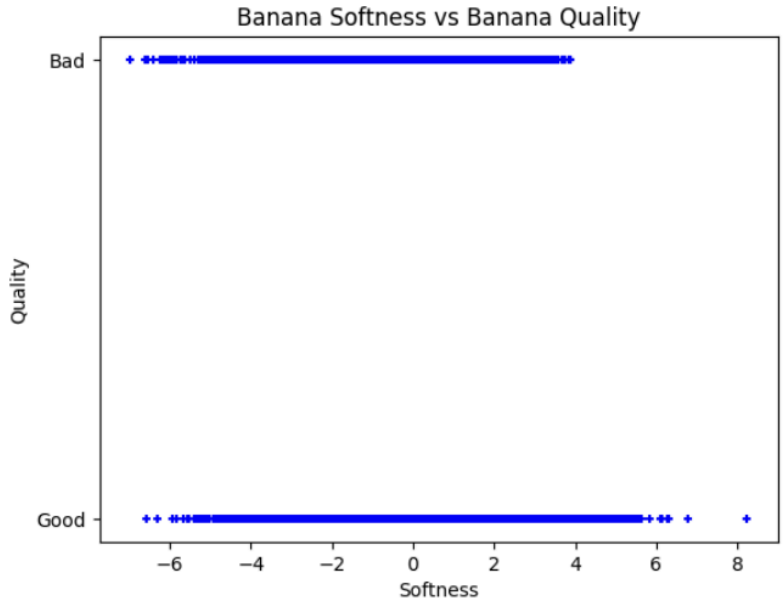
```
plt.scatter(df.Weight,df.Quality,s=20,marker='*',c='green')
plt.xlabel('Weight')
plt.ylabel('Quality')
plt.title('Banana Weight vs Banana Quality')
plt.show()
```



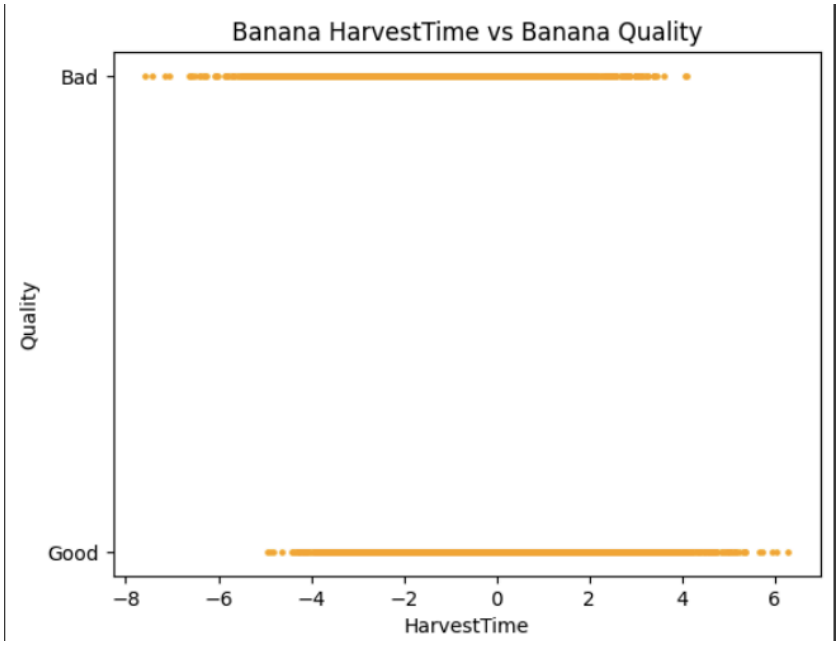

```
plt.scatter(df.Sweetness,df.Quality,s=20,marker='x',c='red')
plt.xlabel('Sweetness')
plt.ylabel('Quality')
plt.title('Banana Sweetness vs Banana Quality')
plt.show()
```



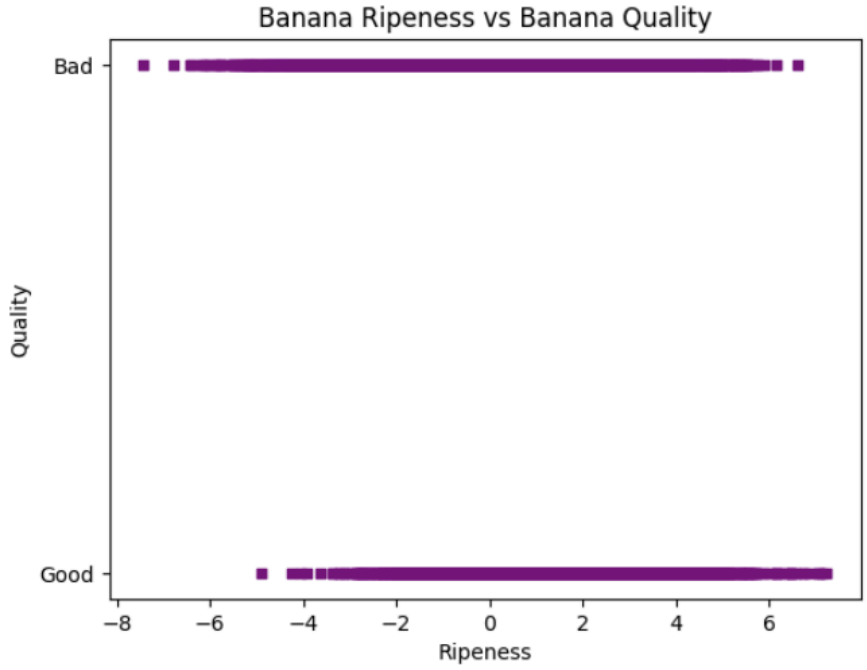
```
plt.scatter(df.Softness,df.Quality,s=20,marker='+',c='blue')
plt.xlabel('Softness')
plt.ylabel('Quality')
plt.title('Banana Softness vs Banana Quality')
plt.show()
```



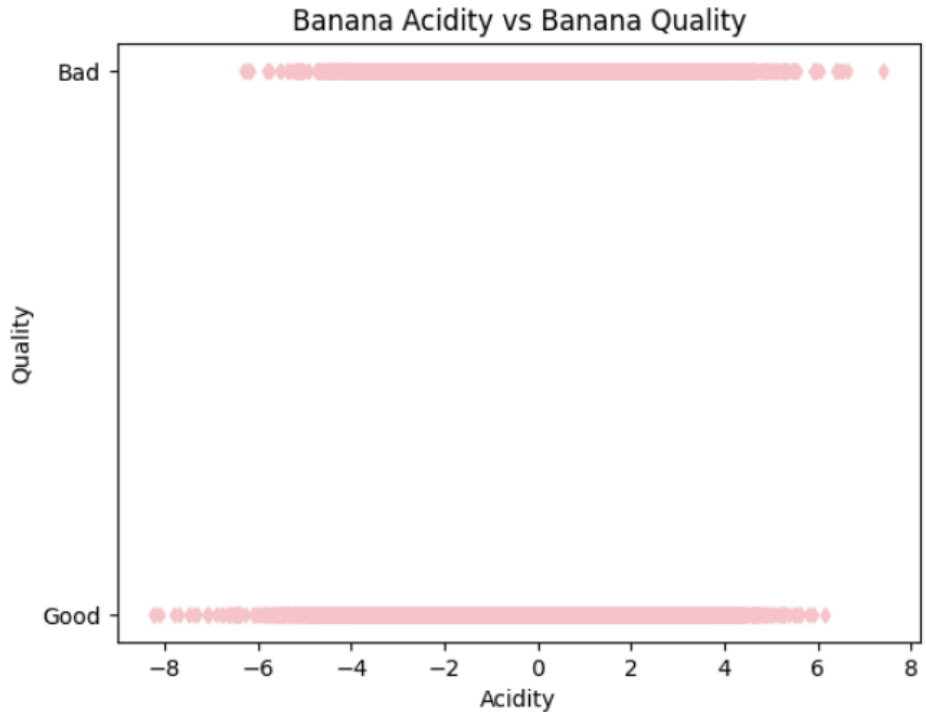

```
plt.scatter(df.HarvestTime,df.Quality,s=20,marker='.',c='orange')
plt.xlabel('HarvestTime')
plt.ylabel('Quality')
plt.title('Banana HarvestTime vs Banana Quality')
plt.show()
```



```
plt.scatter(df.Ripeness,df.Quality,s=20,marker='s',c='purple')
plt.xlabel('Ripeness')
plt.ylabel('Quality')
plt.title('Banana Ripeness vs Banana Quality')
plt.show()
```



```
plt.scatter(df.Acidity,df.Quality,s=20,marker='d',c='pink')
plt.xlabel('Acidity')
plt.ylabel('Quality')
plt.title('Banana Acidity vs Banana Quality')
plt.show()
```




4) Divide the columns into features and target

```
# Assume the last column is the target (label) and all other columns are features
X = df.iloc[:, :-1].values  # Features (all columns except the last)
y = df.iloc[:, -1].values  # Target (last column)
print(X)
print('-------------------------')
print(y)
```

```
[[-1.9249682   0.46807805  3.0778325  ...  0.2947986   2.4355695
   0.27129033]
 [-2.4097514   0.48686993  0.34692144 ... -0.8922133   2.0675488
   0.30732512]
 [-0.3576066   1.4831762   1.5684522  ... -0.64726734  3.0906434
   1.427322  ]
 ...
 [ 1.4227225  -1.9076649  -2.532364   ... -0.5623754  -1.8347653
   0.6973611 ]
 [-2.131904   -2.7425997  -1.0080286  ... -0.80263203 -3.5802662
   0.4235689 ]
 [-2.6608794  -2.0446665   0.15902641 ... -1.5818563  -1.6058589
   1.4356443 ]]
-------------------------
['Good' 'Good' 'Good' ... 'Bad' 'Bad' 'Bad']
```

5) Convert categorical values to numerical values

```
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

y = y.reshape(y.shape[0], 1)
print(y)
```

```
[[1]
 [1]
 [1]
 ...
 [0]
 [0]
 [0]]
```

LabelEncoder() converts the categorical data to a specific numerical value. In this case, Good is replaced by 1 and Bad replaced by 0.

y.shape[0] gives the total number of rows.

y.reshape(y.shape[0],1) changes the dimensions of the array to the number of rows and each row will contain an array of 1 element.

6) Standardise / Normalise the feature values

```
# Normalize the data (only features)
scaler = StandardScaler()
X = scaler.fit_transform(X)
print(X)
```

```
[[-0.55113643  0.60972933  1.97505067 ...  0.52395094  0.78256769
   0.11449116]
 [-0.77810662  0.61905159  0.57338529 ... -0.07058475  0.60849327
   0.13020408]
 [ 0.18268513  1.11329814  1.20034726 ...  0.05210072  1.09241877
   0.61857695]
 ...
 [ 1.01621574 -0.56882668 -0.90443461 ...  0.09462033 -1.237308
   0.30027865]
 [-0.64802151 -0.98302025 -0.12205536 ... -0.0257164  -2.06293293
   0.18089199]
 [-0.895682   -0.63679029  0.47694643 ... -0.41600452 -1.12903488
   0.62220587]]
```

StandardScaler() normalizes the values to a common scale

7) Split the dataset into training and testing

```
# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Ensure y_train and y_test are floats (important for cost computation)
y_train = y_train.astype(float)
y_test = y_test.astype(float)
```

train_test_split divides the dataset into 4 parts. First 2 parts are the training feature and testing feature. Remaining 2 parts are training output and testing output.

Test_size defines the percentage of the dataset to allocate to testing and remaining to training. In this, 20% dataset is testing and remaining 80% is training dataset.

By defining random_state, we can ensure that same split will happen in different machines

.astype(float) converts the current data type into float data type

8) Activation functions and their derivatives

```
def Sigmoid(z):
    return 1 / (1 + np.exp(-z))

def Sigmoid_Derivative(z):
    return Sigmoid(z) * (1 - Sigmoid(z))

def ReLU(z):
    return np.maximum(z,0)

def ReLU_Derivative(z):
    return np.where(z > 0, 1, 0)
```

Using ReLU for hidden layers and sigmoid for output layer

9) Forward Propagation

```
def Forward_Propagation(X, weights, biases):
    Z1 = np.dot(X, weights['W1']) + biases['b1']
    A1 = ReLU(Z1)  # Use ReLU for hidden layer
    Z2 = np.dot(A1, weights['W2']) + biases['b2']
    A2 = Sigmoid(Z2)  # Use Sigmoid for output layer
    return A1, A2
```

It takes input parameters as Input features, weights and biases.
Z1 represents the aggregation for layer 1 and A1 represents the activation applied for layer 1 and same for Z2 and A2.

10) Backpropagation

```python
def Back_Propagation(X, Y, A1, A2, weights):
    m = X.shape[0] # number of rows

    dZ2 = A2 - Y
    dW2 = (1/m) * np.dot(A1.T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)

    dZ1 = np.dot(dZ2, weights['W2'].T) * ReLU_Derivative(A1)  # Use ReLU derivative in hidden layer
    dW1 = (1/m) * np.dot(X.T, dZ1)
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)

    return dW1, db1, dW2, db2
```

It takes the inputs as training dataset for features(X) and outputs(Y), activation values for layer 1(A1) and layer 2(A2) and weights.
While traversing back in the neural network from output layer to input layer, it calculates the change in weights and biases by doing the derivatives(applying chain rule) at each layer.

11) Cost Computation

```python
def compute_cost(A2, Y):
    m = Y.shape[0] # number of rows
    cost = (-1/m) * np.sum(Y * np.log(A2) + (1 - Y) * np.log(1 - A2))
    return np.squeeze(cost)
```

It takes the input parameters as predicted value(A2) and true value(Y)
As it is a binary classification problem, we are using a binary cross entropy loss function.
np.squeeze(cost) removes the dimension equal to 1.

12) Parameter Initialisation

```python
def initialize_parameters(input_size, hidden_size, output_size):
    np.random.seed(42) # setting the starting point for pseudo random numbers
    weights = {
        'W1': np.random.randn(input_size, hidden_size) * 0.01,
        'W2': np.random.randn(hidden_size, output_size) * 0.01
    }
    biases = {
        'b1': np.zeros((1, hidden_size)),
        'b2': np.zeros((1, output_size))
    }
    return weights, biases
```

It takes the input parameters as number of neurons in input layer, hidden layer and output layer
np.random.seed() sets the starting value for pseudo random numbers
Weights are initialized randomly and biases are set to zero.

13) Training the neural network

```
def Train_Network(X, Y, hidden_size, epochs, learning_rate):
    input_size = X.shape[1] # number of columns
    output_size = 1  # output contains only 1 neuron as it is a Binary classification

    weights, biases = initialize_parameters(input_size, hidden_size, output_size)
    costs = []

    for epoch in range(epochs):
        # Forward propagation
        A1, A2 = Forward_Propagation(X, weights, biases)

        # Cost computation
        cost = compute_cost(A2, Y)
        costs.append(cost)

        # Backpropagation
        dW1, db1, dW2, db2 = Back_Propagation(X, Y, A1, A2, weights)

        # Update weights and biases
        weights['W1'] -= learning_rate * dW1
        biases['b1'] -= learning_rate * db1
        weights['W2'] -= learning_rate * dW2
        biases['b2'] -= learning_rate * db2

        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Cost: {cost}")

    return weights, biases, costs
```

It takes inputs as training dataset for features(X) and outputs(Y), neurons in hidden layer, number of epochs, and the learning rate.
First, the weights and biases are initialized and then the model will run for the whole dataset for the specified number of epochs
Output layer contains only 1 neuron as it is a binary classification problem. If the output is 1/True then it is a good quality otherwise bad.
In a single epoch, the model will first forward propagates which includes aggregation and activation and then cost is computed and then backpropagation happens.
After the backpropagation, weights and biases are updated by subtracting the product of learning rate and change in weight/bias.

14) Prediction for testing dataset

```
def Predict_Output(X, weights, biases):
    _, A2 = Forward_Propagation(X, weights, biases)
    return A2 > 0.5
```

It takes the inputs as testing dataset(X), weights and biases.
After the model is trained, the testing dataset undergoes forward propagation.
Based on the threshold, it outputs the boolean values. If the value is greater than 0.5, then it is a good quality banana or else bad quality.

15) Defining hyper parameters and training the network and plotting a graph between cost and number of epochs

```
# Training parameters
epochs = 10000
hidden_size = 10
learning_rate = 0.01

# Train the model
weights, biases, costs = Train_Network(X_train, y_train, hidden_size, epochs, learning_rate)

# Plot cost vs. number of epochs
plt.plot(range(epochs), costs)
plt.title('Cost vs. Number of Epochs')
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.show()
```

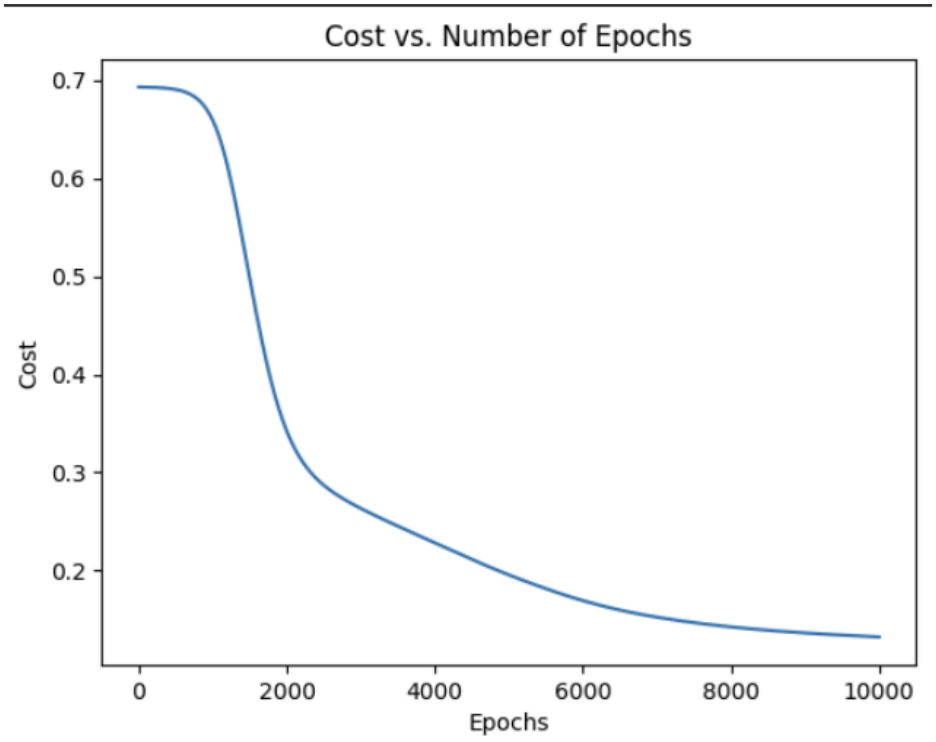
Number of epochs, neurons in hidden layer and learning rate

After the training of the network, plotting will be done in which the x axis represents Number of Epochs and the y-axis represents Cost Associated with each Epoch.

```
Epoch 0, Cost: 0.6930307077637659
Epoch 100, Cost: 0.6928646559620514
Epoch 200, Cost: 0.6926191903864194
Epoch 300, Cost: 0.6922255955585823
Epoch 400, Cost: 0.6915715435062458
Epoch 500, Cost: 0.6904743781661796
Epoch 600, Cost: 0.6886274313756109
Epoch 700, Cost: 0.6855338708857203
Epoch 800, Cost: 0.6804121332141173
Epoch 900, Cost: 0.6721039414770708
Epoch 1000, Cost: 0.6590402732943459
Epoch 1100, Cost: 0.6395000603471105
Epoch 1200, Cost: 0.6123232402048221
Epoch 1300, Cost: 0.5779221886234743
Epoch 1400, Cost: 0.5386274183073364
Epoch 1500, Cost: 0.4976721690066389
Epoch 1600, Cost: 0.4579200931986235
Epoch 1700, Cost: 0.4214771140048461
Epoch 1800, Cost: 0.38975149095084166
Epoch 1900, Cost: 0.36337195358380536
Epoch 2000, Cost: 0.3421971028992257
```

.....

```
Epoch 8000, Cost: 0.14259874578676507
Epoch 8100, Cost: 0.14184297125685663
Epoch 8200, Cost: 0.14112041542217418
Epoch 8300, Cost: 0.14043263975833897
Epoch 8400, Cost: 0.1397803889739324
Epoch 8500, Cost: 0.13915540108437827
Epoch 8600, Cost: 0.13856165103564355
Epoch 8700, Cost: 0.13799353405037912
Epoch 8800, Cost: 0.1374502450732951
Epoch 8900, Cost: 0.13692932045363637
Epoch 9000, Cost: 0.13643219441516297
Epoch 9100, Cost: 0.1359559337612735
Epoch 9200, Cost: 0.1355011079766428
Epoch 9300, Cost: 0.13506549665794548
Epoch 9400, Cost: 0.134646540757249
Epoch 9500, Cost: 0.1342378757790018
Epoch 9600, Cost: 0.13384177013974138
Epoch 9700, Cost: 0.1334608857701261
Epoch 9800, Cost: 0.13309420389536306
Epoch 9900, Cost: 0.13273927560864432
```




16) Predicting the accuracy of the model based on the testing dataset

```
predictions = Predict_Output(X_test, weights, biases)
predicted_classes = (predictions > 0.5).astype(int)  # Binary classification threshold
accuracy = accuracy_score(y_test, predicted_classes)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.96
```

It compares the Actual value and the true values and gives the accuracy of the neural network.