Scratch

In [4]:
```python
import pandas as pd
#These imports are just for the visualizations
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

data = pd.read_csv('/Users/lukehenry/Documents/Jupyter/ANN/data.csv') # Reac
# Understanding the data
print(data.shape)
print(data.describe().T)
print(data.info())
data.head()
```

```
(1032, 3)
    count       mean      std  min  25%   50%   75%   max
x  1032.0  11.024322  6.67162  0.0  5.1  10.8  16.0  25.9
y  1032.0   7.827326  4.40012  0.0  4.2   7.5  11.1  18.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1032 entries, 0 to 1031
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   x       1032 non-null   float64
 1   y       1032 non-null   float64
 2   Class   1032 non-null   object 
dtypes: float64(2), object(1)
memory usage: 24.3+ KB
None
```

Out[4]:

	x	y	Class
0	24.9	15.9	C3
1	25.2	15.0	C1
2	24.6	14.8	C3
3	25.1	13.6	C2
4	25.6	12.5	C1

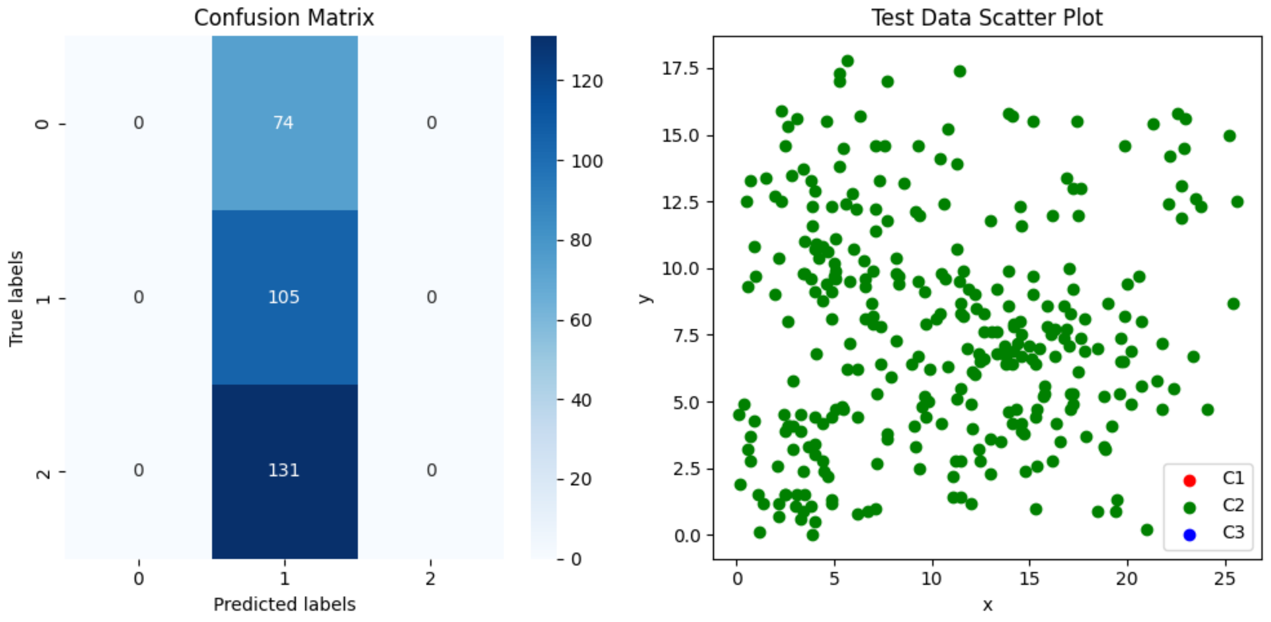
In [2]:
```python
import numpy as np

# Define sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
# Define softmax activation function
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

# Define ANN class
class ANN:
    def __init__(self, input_dim, hidden_dim, output_dim):
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim

        # Initialize weights and biases
        self.w1 = np.random.randn(self.hidden_dim, self.input_dim)
        self.b1 = np.ones((self.hidden_dim, 1))
        self.w2 = np.random.randn(self.output_dim, self.hidden_dim)
        self.b2 = np.ones((self.output_dim, 1))

    def forward(self, x):
        # Forward propagation through the network
        z1 = np.dot(self.w1, x) + self.b1
        a1 = sigmoid(z1)
        z2 = np.dot(self.w2, a1) + self.b2
        a2 = softmax(z2)

        return a2

    def train(self, x_train, y_train, epochs, learning_rate):
        for i in range(epochs):
            # Forward propagation
            z1 = np.dot(self.w1, x_train.T) + self.b1
            a1 = sigmoid(z1)
            z2 = np.dot(self.w2, a1) + self.b2
            a2 = softmax(z2)

            # Backward propagation
            delta2 = (a2 - y_train.T) / y_train.shape[0]
            dW2 = np.dot(delta2, a1.T)
            db2 = np.sum(delta2, axis=1, keepdims=True)
            delta1 = np.dot(self.w2.T, delta2) * a1 * (1 - a1)
            dW1 = np.dot(delta1, x_train)
            db1 = np.sum(delta1, axis=1, keepdims=True)

            # Update weights and biases
            self.w2 -= learning_rate * dW2
            self.b2 -= learning_rate * db2
            self.w1 -= learning_rate * dW1
            self.b1 -= learning_rate * db1
    def predict(self, x):
        # Make predictions on input data
        a2 = self.forward(x.T)
        return np.argmax(a2, axis=0)

    def score(self, x_test, y_test):
        y_pred = self.predict(x_test)
        accuracy = np.mean(y_pred == np.argmax(y_test.T, axis=0))
        return accuracy
```

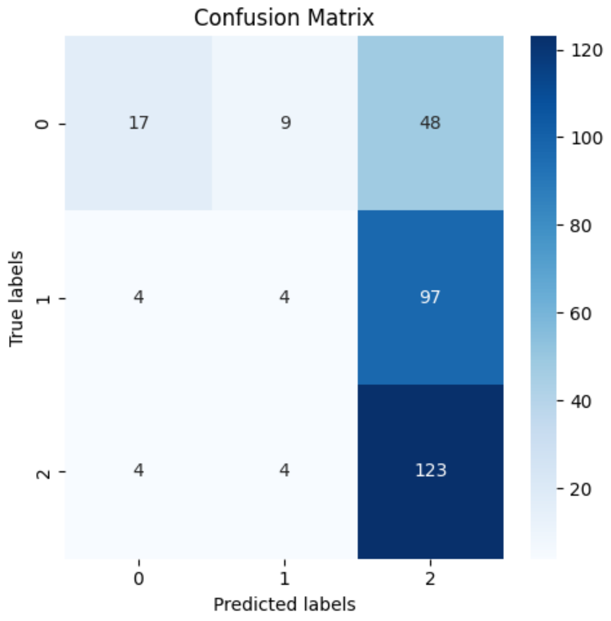
In [5]:
```
def make_graphs(X_test):
    y_pred = ann.predict(X_test)
    cm = confusion_matrix(np.argmax(y_test, axis=1), y_pred)
    # 0 == C1
    # 1 == C2
    # 2 == C3

    # heatmap visualization of the confusion matrix using seaborn's heatmap
    sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', ax = ax1)
    ax1.set_xlabel('Predicted labels')
    ax1.set_ylabel('True labels')
    ax1.set_title('Confusion Matrix')

    # Create scatter plot of test data points colored by predicted label
    colors = ['r', 'g', 'b']
    labels = ['C1', 'C2', 'C3']
    for i in range(len(colors)):
        ax2.scatter(X_test[y_pred==i, 0], X_test[y_pred==i, 1], c=colors[i],

    ax2.set_xlabel('x')
    ax2.set_ylabel('y')
    ax2.set_title('Test Data Scatter Plot')
    ax2.legend()

    plt.tight_layout()
    plt.show()

# Convert class labels to one-hot encoding
labels = pd.get_dummies(data['Class'])
data = pd.concat([data[['x', 'y']], labels], axis=1)

# Split dataset into training and testing sets 70% train, 30% test
train_data = data.sample(frac=0.7, random_state=42)
test_data = data.drop(train_data.index)

# Extract input and output data from training and testing sets
x_train = train_data[['x', 'y']].values
y_train = train_data[['C1', 'C2', 'C3']].values
x_test = test_data[['x', 'y']].values
y_test = test_data[['C1', 'C2', 'C3']].values
```

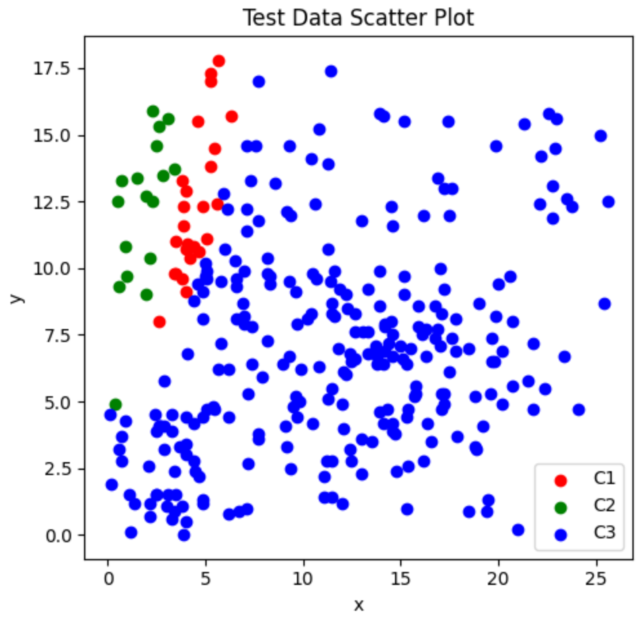
```
# Run the accuracy test 10 times and plot each result
num = 10
i = 0
scores = []
while (i < num):
    ann = ANN(input_dim=2, hidden_dim=5, output_dim=3)
    ann.train(x_train, y_train, 10, 0.03)
    accuracy = ann.score(x_test, y_test)
    print(i,":","Accuracy on test data: {:.2f}%".format(accuracy * 100))
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
    make_graphs(x_test)
    scores.append(accuracy * 100)
    i = i + 1

print("Summarized scores:")
for i in scores:
    print("{:.2f}%".format(i))

print("Average Accuracy:", "{:.2f}%".format(np.mean(scores)))
```

```
0 : Accuracy on test data: 33.87%
```

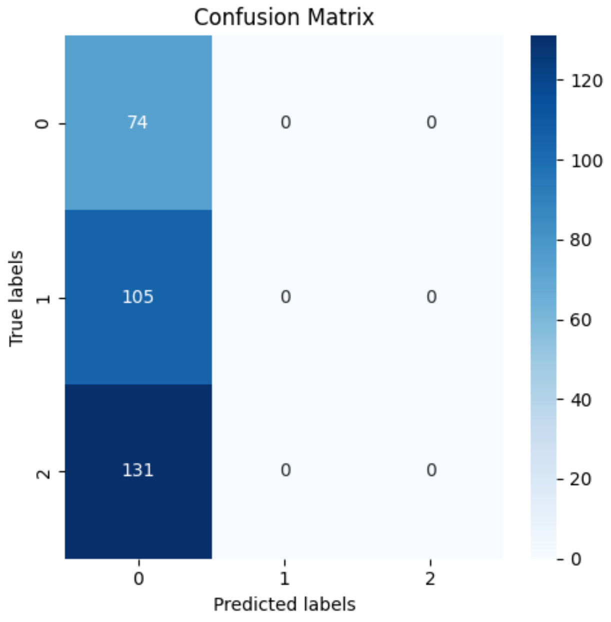


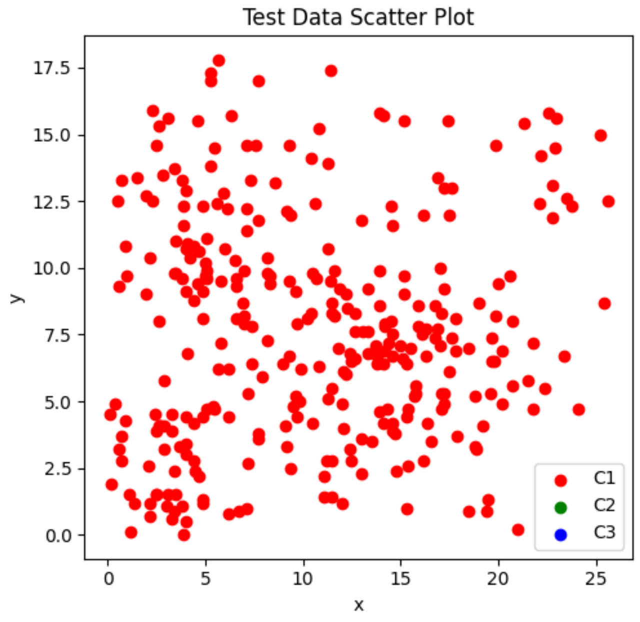
```
1 : Accuracy on test data: 46.45%
```



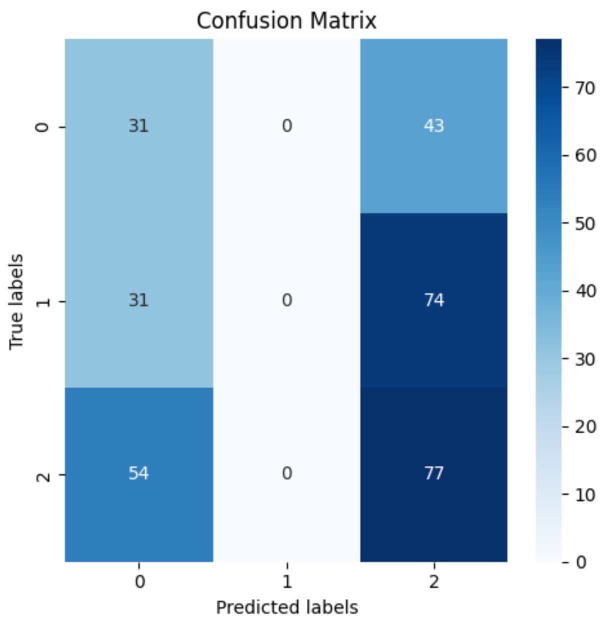


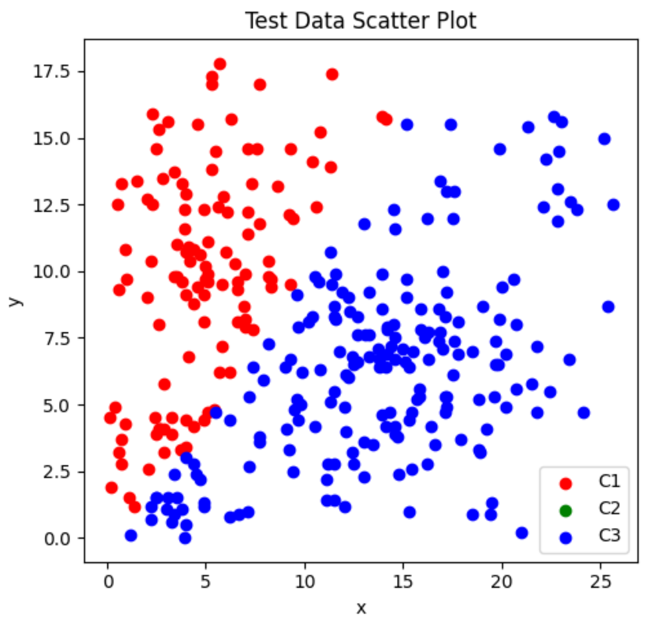
```
2 : Accuracy on test data: 23.87%
```



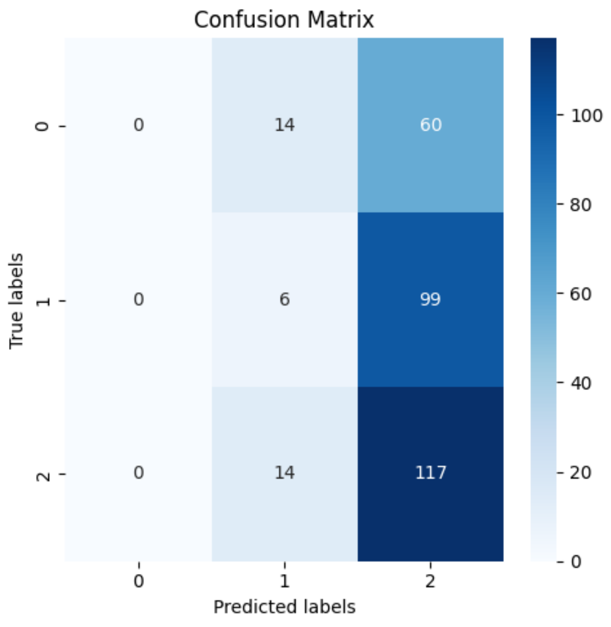


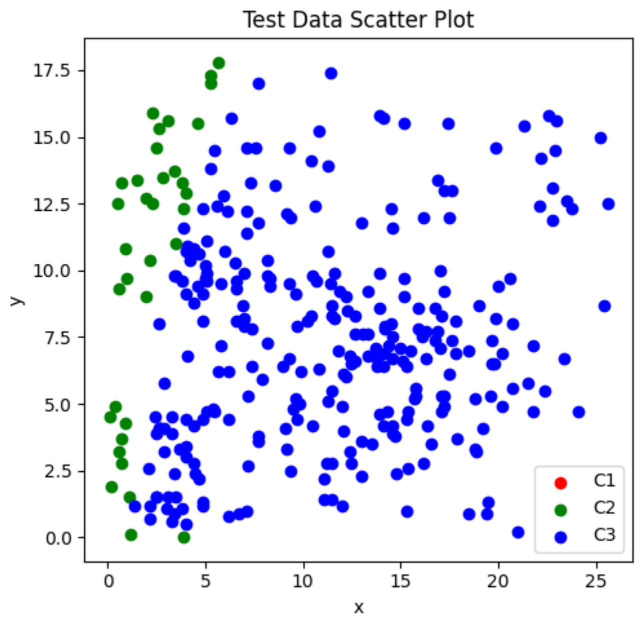
```
3 : Accuracy on test data: 34.84%
```



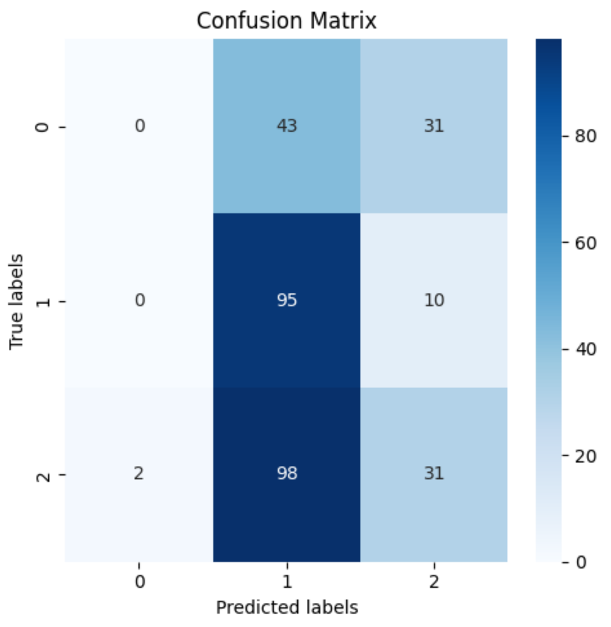


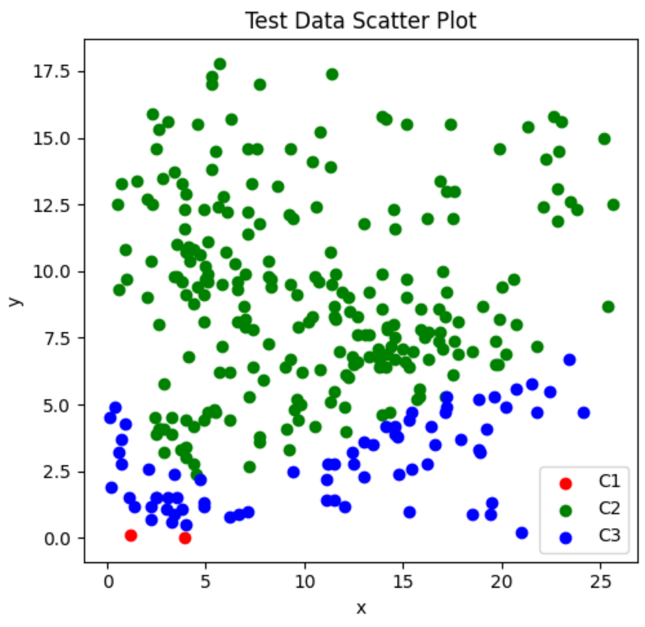
```
4 : Accuracy on test data: 39.68%
```



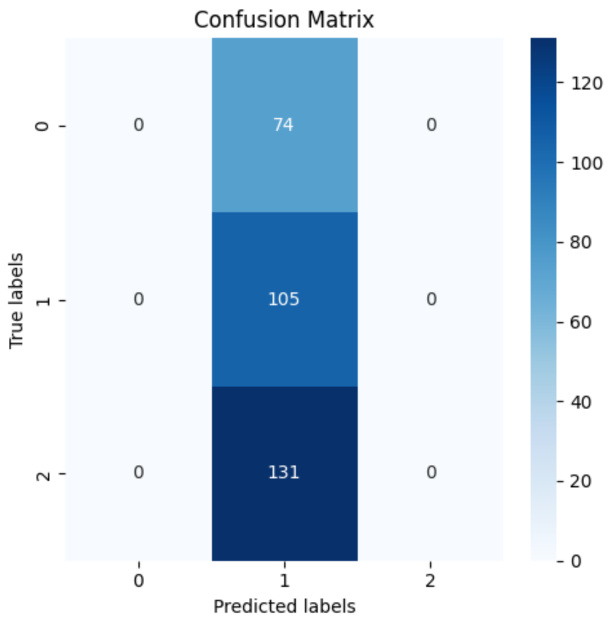


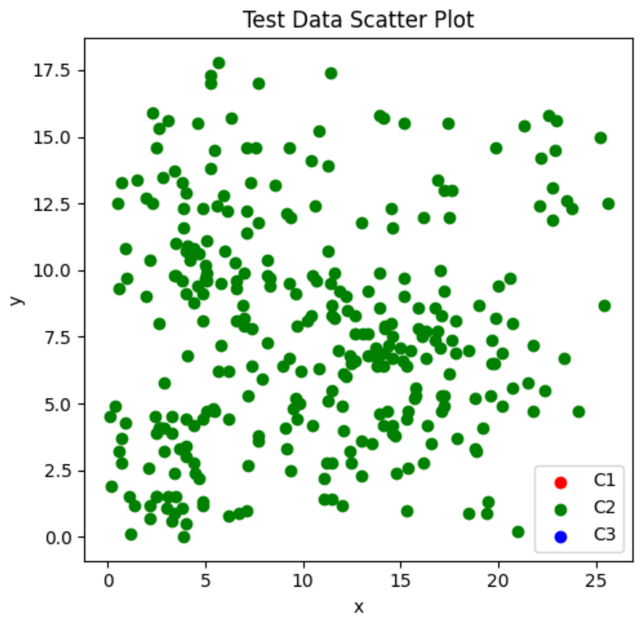
```
5 : Accuracy on test data: 40.65%
```



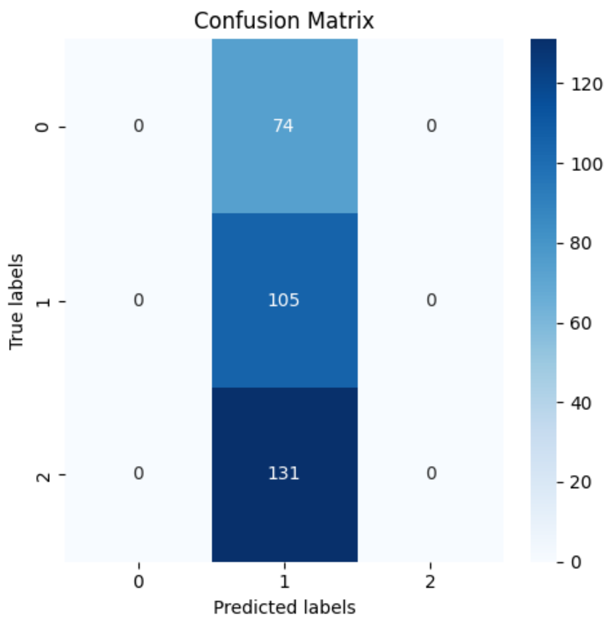


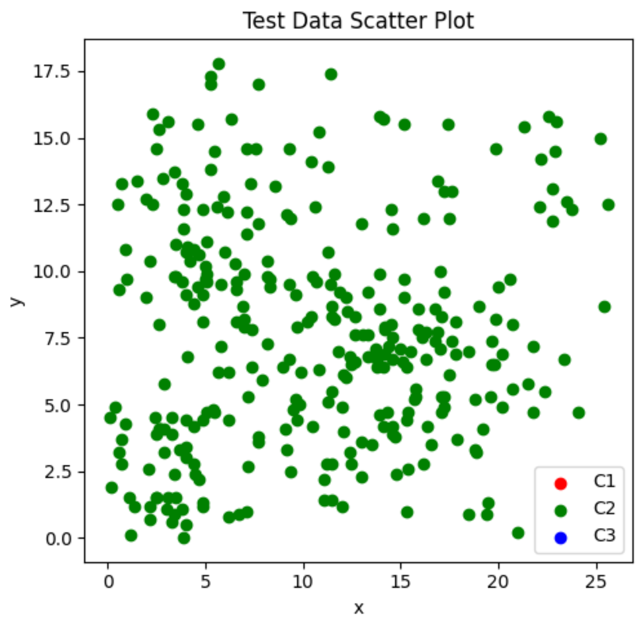
```
6 : Accuracy on test data: 33.87%
```



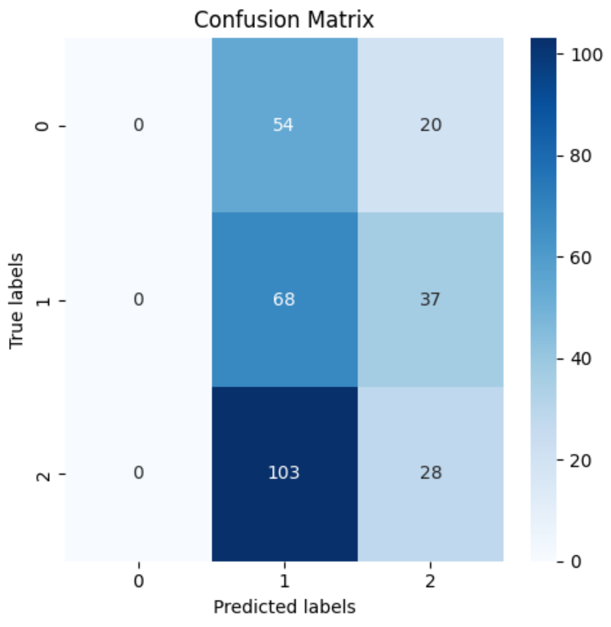


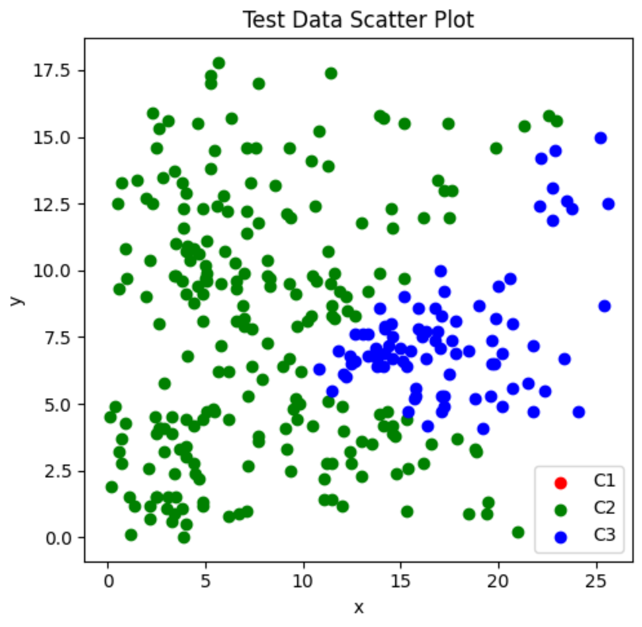
```
7 : Accuracy on test data: 33.87%
```



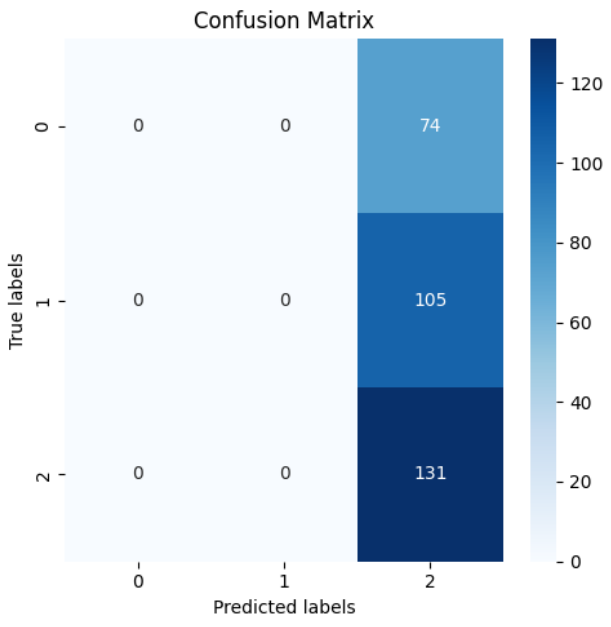


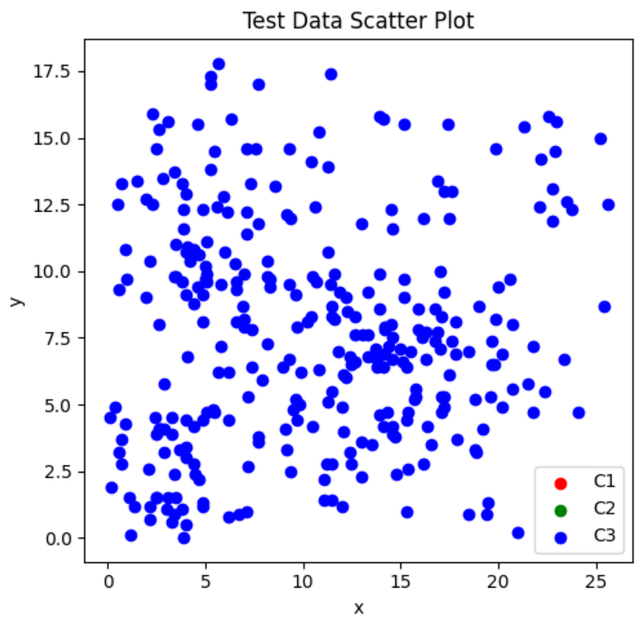
```
8 : Accuracy on test data: 30.97%
```





```
9 : Accuracy on test data: 42.26%
```





```
Summarized scores:
33.87%
46.45%
23.87%
34.84%
39.68%
40.65%
33.87%
33.87%
30.97%
42.26%
Average Accuracy: 36.03%
```

In [40]:

```
data.head()
```

Out[40]:

	x	y	C1	C2	C3
0	24.9	15.9	0	0	1
1	25.2	15.0	1	0	0
2	24.6	14.8	0	0	1
3	25.1	13.6	0	1	0
4	25.6	12.5	1	0	0

Perfomance Summary

See above visualizations for summary

The perfomance of my model was not the best. I have it set to run and get an accuracy ten times since no two tests are identical, and the results ranged from about 23% to 46%. One reason of why the accuracy is so low is because of the small size of data that the neural network is given. For example, I read online that if an ANN was being built to be able to learn and depict whether an image is a cat or a dog, it would be given thousands of images. Compared to this, our model looks shoddy, having 4 instances to learn from.

Process of Coding From Scratch

In order to build the ANN from scratch instea of building with libraries, the following steps had to be excecuted:

- Define the activation functions: In this ANN, two activation functions are used: sigmoid and softmax. The sigmoid function is used in the hidden layer, and the softmax function is used in the output layer.
- Define the ANN class: In this step, we define the ANN class and initialize the input, hidden, and output layer dimensions. We also initialize the weights and biases with random values.
- Implement the forward propagation algorithm: In the forward propagation step, we compute the weighted sum of inputs and biases for each neuron in the hidden layer and apply the activation function.
- Next, we compute the weighted sum of the hidden layer outputs and biases for each neuron in the output layer and apply the softmax function.
- Implement the backpropagation algorithm: In the backpropagation step, we calculate the error in the output layer and use it to adjust the weights and biases in both the output and hidden layers. We use the delta rule to calculate the error in each layer, and we adjust the weights and biases using the gradient descent algorithm.
- Train the ANN: In this step, we train the ANN by iterating over the dataset multiple times (epochs) and updating the weights and biases using the backpropagation algorithm. We also define a learning rate, which controls the size of the weight and bias updates.
- Implement the prediction function: In the prediction step, we use the trained ANN to make predictions on new inputs.
- Implement the accuracy function: In the accuracy step, we compute the accuracy of the ANN by comparing its predictions with the true labels.

I didn't know that I was allowed to use the scikit test train split until after I spent a couple hours figuring how to proceed without it. Instead of regular test train split, I took the potential values of the Class column (C1, C2, and C3) and make them thier own columns and for each row. The way that it transalted was if row 1 had C1, after the update with the new columns row 1 would have a 1 for C1 and 0s for C2 and C3. To do this I used the getDummies(), concat(), and sample() functions

Sources

Youtube ANN help: https://www.youtube.com/watch?v=w8yWXqWQYmU&t=2s

ANN Explanation and Break Down: https://www.forbes.com/sites/bernardmarr/2018/09/24/what-are-artificial-neural-networks-a-simple-explanation-for-absolutely-anyone/?sh=3a82c0331245

Learning about how to do the layers: https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

General ANN example to work off of: https://www.analyticsvidhya.com/blog/2021/10/implementing-artificial-neural-networkclassification-in-python-from-scratch/?

Manuel test train split help: https://www.sharpsightlabs.com/blog/pandas-get-dummies/