**DL ASSIGNMENT**

**Made by- Priyanshu Gupta 22UCS155**

### **1-Dataset: MNIST**

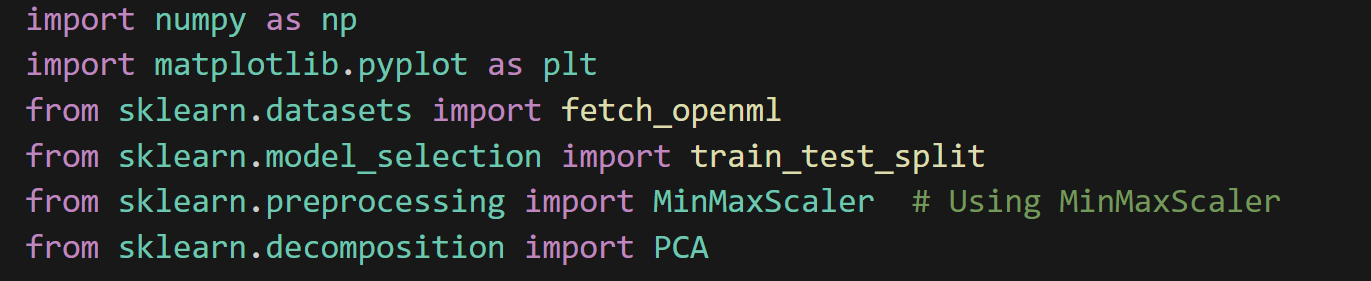
**MNIST (Modified National Institute of Standards and Technology) dataset** is a classic benchmark dataset widely used in machine learning and computer vision. It consists of a large collection of handwritten digits (0-9) and is often used for image classification tasks.

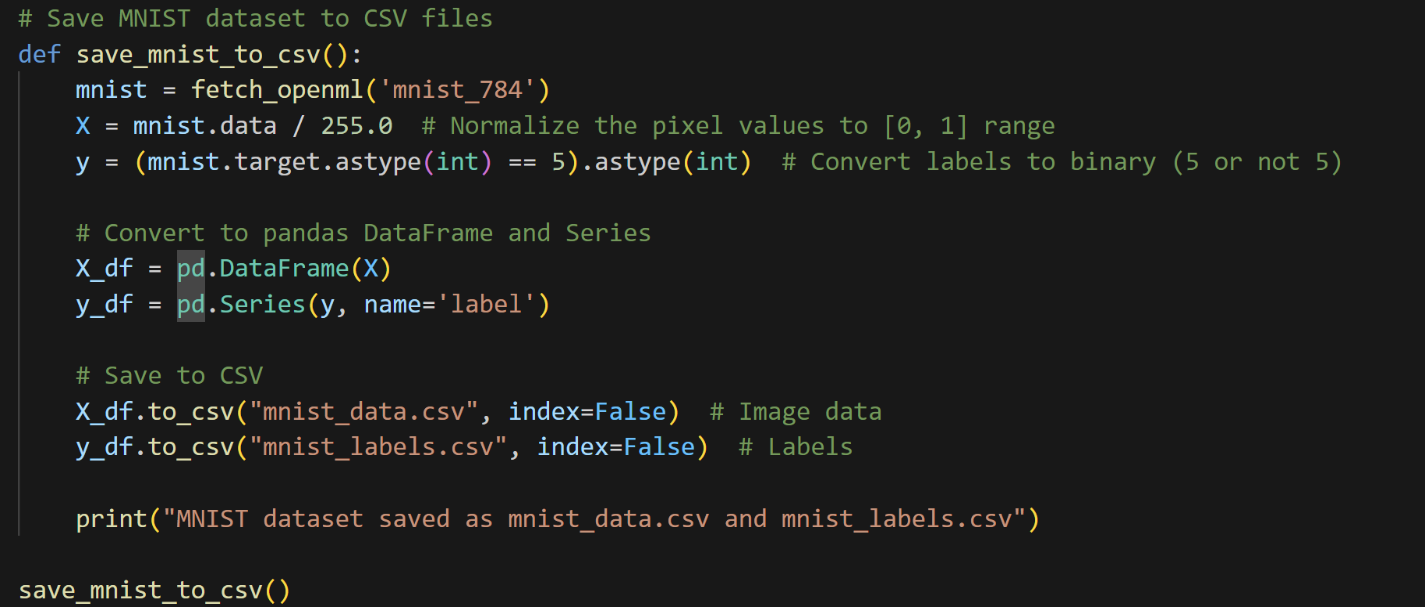
**Key Characteristics:**

* **Total Samples:** 70,000 images (60,000 for training and 10,000 for testing).
* **Image Size:** Each image is 28x28 pixels, resulting in a total of 784 pixels per image.
* **Classes:** There are 10 classes, corresponding to the digits 0 through 9.

### **Types of Variables**

1. **Independent Variables (Features):**
   * The independent variables are the pixel values of the images. Each pixel in a 28x28 image can be considered a feature, resulting in 784 features per image. Each pixel value ranges from 0 to 255, representing the grayscale intensity.
2. **Dependent Variable (Target):**
   * The dependent variable is the label indicating the digit represented in the image. In your case, the labels are binary: 1 if the digit is 5 and 0 for all other digits. Thus, your target variable is binary.





Explanation of Code:

**1. Import Necessary Libraries:**

**NumPy**: Used for numerical computations and array manipulations.

**Matplotlib**: Used for plotting graphs and visualizations.

**sklearn.datasets**: Used to load the MNIST dataset.

**sklearn.model\_selection**: Provides utilities for splitting datasets into training and testing sets.

**sklearn.preprocessing**: Contains the MinMaxScaler for normalizing data.

**sklearn.decomposition**: Used for PCA (Principal Component Analysis) to reduce dimensionality.

**2. Load the MNIST Dataset:**

**fetch\_openml('mnist\_784')**: Loads the MNIST dataset from OpenML.

**Normalization**: The pixel values are divided by 255.0 to bring them into the range [0, 1].

**Label Conversion**: The labels are converted into a binary format:

* 1 if the digit is 5
* 0 for all other digits.

**train\_test\_split**: Splits the dataset into training (80%) and testing (20%) sets

### **Data Dictionary**

Here's a simple data dictionary that describes the dataset:

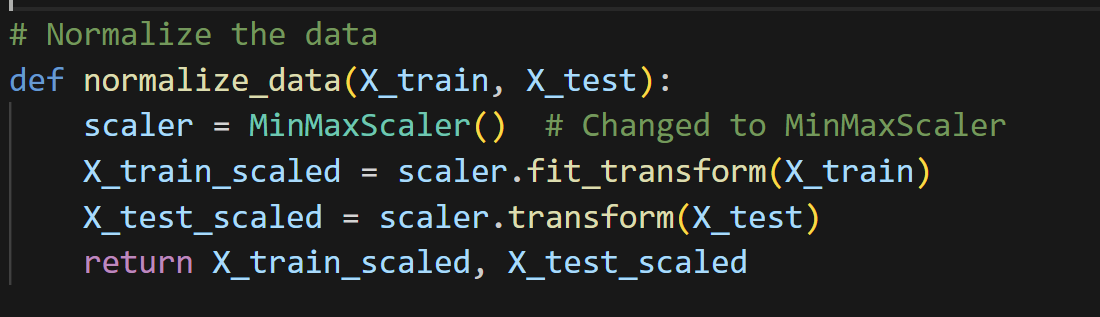
|  |  |  |  |
| --- | --- | --- | --- |
| **Feature Name** | **Description** | **Data Type** | **Range** |
| pixel\_0 | Pixel value at (0,0) position in the image | Numeric | 0 to 255 |
| pixel\_1 | Pixel value at (0,1) position in the image | Numeric | 0 to 255 |
| pixel\_2 | Pixel value at (0,2) position in the image | Numeric | 0 to 255 |
| ... | ... | ... | ... |
| pixel\_783 | Pixel value at (27,27) position in the image | Numeric | 0 to 255 |
| label | Binary label for the digit (1 if 5, else 0) | Categorical | 0 or 1 |

### **2-Normalization**

the normalization technique used is **Min-Max Scaling**. This method scales the pixel values of the MNIST dataset to a range of **[0, 1]**.

### **Key Points:**

* **Normalization Technique**: Min-Max Scaling
* **Formula**: X′=(X−Xmin)/(Xmax-Xmin)
* **Resulting Range**: [0, 1]
* **Purpose**: This scaling helps to ensure that the model trains effectively by preventing issues related to varying scales of input features



Explanation of code:

### **Function: normalize\_data(X\_train, X\_test):**

### This function applies Min-Max Scaling to normalize the training and test datasets. The purpose of normalization is to ensure that the pixel values of the images are on a consistent scale, which can improve the performance of machine learning models**.**

**Initialize the Scaler**: **scaler = MinMaxScaler()**

An instance of MinMaxScaler is created from sklearn.preprocessing. This scaler will transform the data into the range [0, 1].

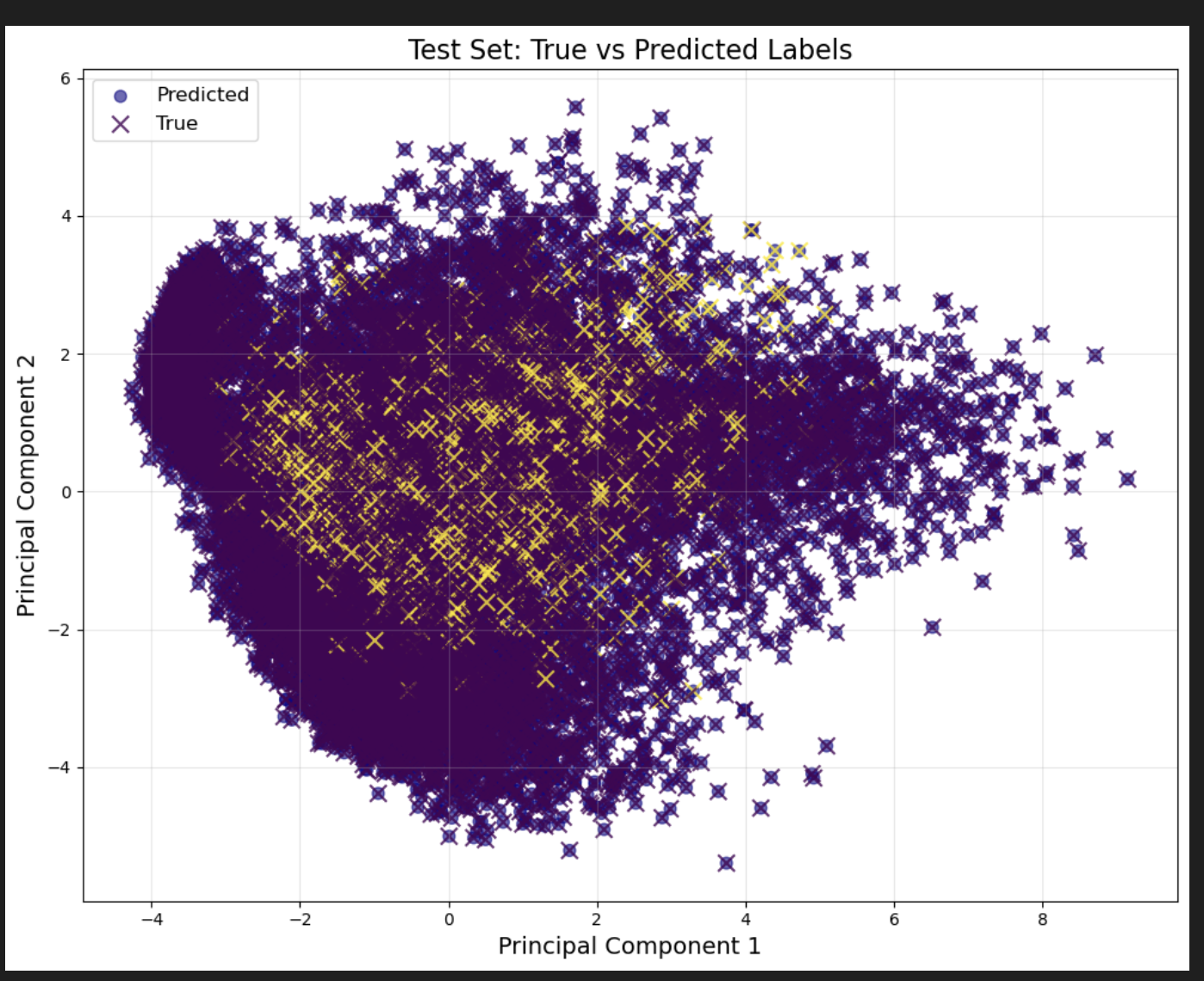
**Fit and Transform the Training Data**:**X\_test\_scaled = scaler.transform(X\_test)**

The transform method scales the test data using the same parameters (min and max) learned from the training data, ensuring consistency.

**Return Scaled Data**:**return X\_train\_scaled, X\_test\_scaled**

The function returns the normalized training and test datasets.

### **3-Scatter plot:**



Based on the scatter plot showing the relationship between Principal Component 1 and Principal Component 2, there appears to be a complex, non-linear correlation between these components. Here are the key observations:

1. Shape: The overall distribution forms an irregular, curved shape that somewhat resembles a heart or kidney shape, suggesting a non-linear relationship between the components.
2. Spread:

* Principal Component 1 ranges roughly from -4 to 8
* Principal Component 2 ranges roughly from -4 to 6
* The densest concentration of points appears in the central region of the plot

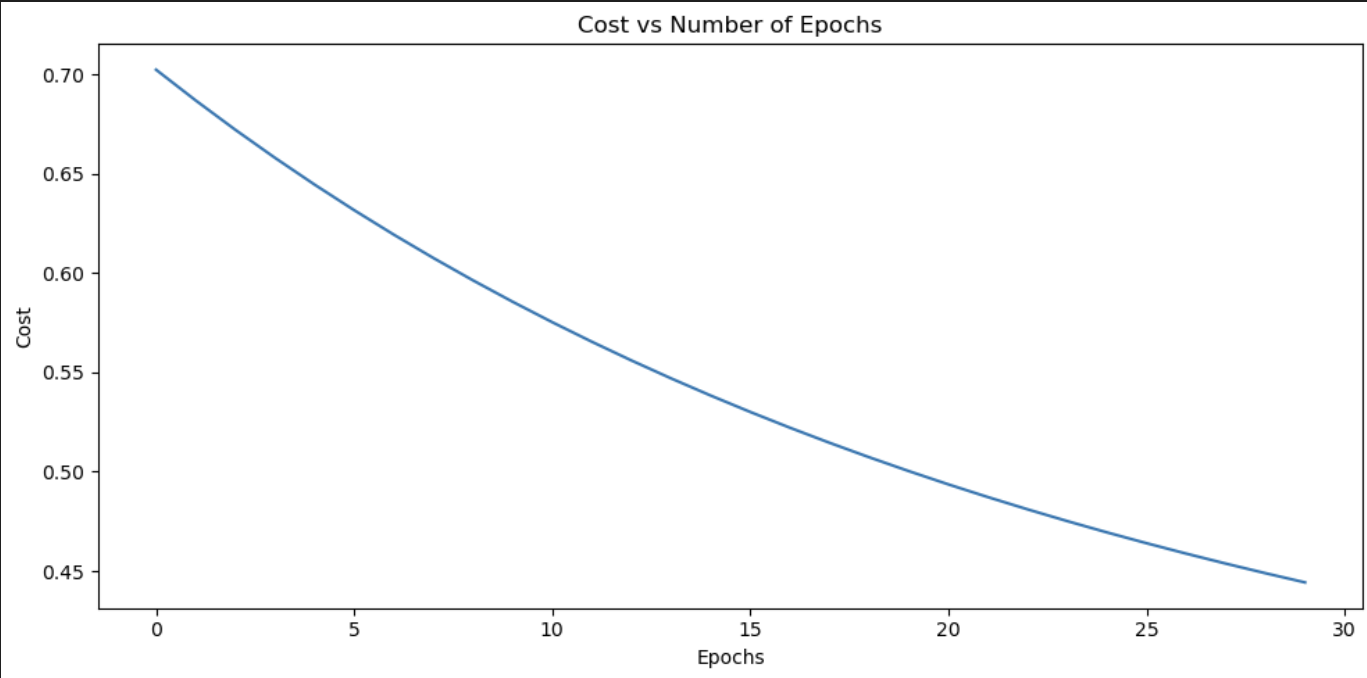
1. Pattern: There's no simple linear correlation between the components. Instead, we see:

* A dense central cluster
* A curved spread of points that extends toward the right side
* More dispersion/scatter on the right side of the plot
* Some outliers, particularly in the positive direction of both components

1. Density: The purple dots (Predicted) and yellow/orange crosses (True) appear to overlap significantly in many areas, suggesting good prediction accuracy in those regions, though there are some areas where they differ.

This type of distribution is common in dimensionality reduction techniques like PCA when the underlying data has complex, non-linear relationships that can't be captured by a simple linear correlation coefficient.

### **4-Cost vs number of epochs:**



As you can see from the graph,the cost/loss is continuously decreasing as the number of epochs are increasing.

**Explaining the main components of the code:**

**1.Data Loading and Preparation**

* **Function: load\_mnist\_dataset()**
* **Purpose:** Fetches the MNIST dataset from OpenML, normalizes pixel values, and converts labels to binary (5 or not 5). It splits the data into training and test sets.
* **Key Step**:

X = mnist.data / 255.0

**2.Normalization of Data:**

* **Function**: normalize\_data(X\_train, X\_test)
* **Purpose**: Applies Min-Max scaling to the training and test datasets to standardize the input feature values.

**3.Neural Network Initialization:**

* Function: initialize\_parameters(layer\_dims)
* Purpose: Initializes weights and biases for each layer in the network. Weights are initialized with small random values, while biases are set to zero

parameters[f'W{l}'] = np.random.randn(layer\_dims[l], layer\_dims[l-1]) \* 0.01

parameters[f'b{l}'] = np.zeros((layer\_dims[l], 1))

**4.Forward Propagation**

* Function: forward\_propagation(X, parameters)
* Purpose: Computes the output of the network for given input X using the current parameters. It applies the sigmoid activation function to determine the activations for each layer.
* Key Steps:

Z = np.dot(parameters[f'W{l}'], A) + parameters[f'b{l}']

A = sigmoid(Z)

**5.Cost Computation**

* Function: compute\_cost(AL, Y)
* Purpose**:** Calculates the cost (loss) of the predictions compared to the actual labels. The cost function used is binary cross-entropy.
* Key Code:

cost = -1/m \* np.sum(Y \* np.log(AL + 1e-10) + (1 - Y) \* np.log(1 - AL + 1e-10))

**6.Backward Propagation**

* Function: backward\_propagation(X, Y, parameters, cache)
* Purpose: Computes gradients for weights and biases using backpropagation. It propagates the error from the output layer back through the network.
* Key Code:

dZ = dAL \* sigmoid\_derivative(cache[f'A{l}'])

grads[f'dW{l}'] = 1/m \* np.dot(dZ, A\_prev.T)

grads[f'db{l}'] = 1/m \* np.sum(dZ, axis=1, keepdims=True)

**7.Parameter Update**

* **Function**: update\_parameters(parameters, grads, learning\_rate)
* **Purpose**: Updates the weights and biases using the computed gradients and the specified learning rate.
* **Key Code**:
* parameters[f'W{l}'] -= learning\_rate \* grads[f'dW{l}']
* parameters[f'b{l}'] -= learning\_rate \* grads[f'db{l}']

**8.Training the Neural Network**

* Function: train\_nn(X, Y, layer\_dims, learning\_rate=0.01, num\_epochs=30)
* Purpose: Runs the training process for the specified number of epochs. It calls forward propagation, computes cost, runs backpropagation, and updates parameters in each epoch.
* Key Steps:
* AL, cache = forward\_propagation(X, parameters)
* cost = compute\_cost(AL, Y)

grads = backward\_propagation(X, Y, parameters, cache)

**9.Prediction**

* Function: predict(X, parameters)
* Purpose: Makes predictions based on the trained parameters. It uses a threshold (0.5) to determine if the predicted class is 5 or not 5.
* **Key Code:**

predictions = (AL > 0.5).astype(int)

**10.PCA and Visualization**

* Functions: plot\_pca\_variance(X) and plot\_pca\_scatter(X, y\_true, y\_pred, title)
* Purpose: These functions use PCA (Principal Component Analysis) for dimensionality reduction and visualization of the dataset. The first plots the explained variance, while the second plots the true vs. predicted labels in a 2D space.
* Key Code for Scatter Plot:

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y\_pred.flatten(), cmap='plasma', alpha=0.6, s=50, label='Predicted') # Changed color map

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y\_true.flatten(), marker='x', alpha=0.8, s=100, label='True')

**Main Execution Code Breakdown**

**1.Main Program Entry Point**

* Condition: if \_\_name\_\_ == "\_\_main\_\_":
* Purpose: This ensures that the following code runs only if the script is executed directly (not imported as a module).

**2.Loading and Normalizing the Dataset**

X\_train, X\_test, y\_train, y\_test = load\_mnist\_dataset()

X\_train\_scaled, X\_test\_scaled = normalize\_data(X\_train, X\_test)

Function: load\_mnist\_dataset()

* Loads the MNIST dataset, normalizes the pixel values, and splits it into training and test sets.

Function: normalize\_data(X\_train, X\_test)

* Applies Min-Max Scaling to the training and test datasets, ensuring that input values are between 0 and 1.

**3.Converting Labels to NumPy Arrays**

y\_train = y\_train.to\_numpy() # or use y\_train.values

y\_test = y\_test.to\_numpy()

Purpose: Converts the target labels (y\_train and y\_test) into NumPy arrays for easier manipulation and processing during training and evaluation

**4.Optional PCA Variance Plotting**

Function: plot\_pca\_variance(X\_train\_scaled)

plot\_pca\_variance(X\_train\_scaled) # Plotting variance for analysis

Purpose: Plots the cumulative explained variance of the PCA components for the training dataset, helping to analyze how many components are necessary to capture the variance in the data.

**5.Defining Neural Network Architecture**

layer\_dims = [X\_train\_scaled.shape[1], 64, 32, 1]

Purpose: Specifies the architecture of the neural network. The first element is the number of input features (the size of the flattened images), followed by the number of neurons in the hidden layers (64 and 32), and the final element (1) represents the output layer for binary classification.

**6.Training the Neural Network**

num\_epochs = 30 # Specify the number of epochs

parameters, costs = train\_nn(X\_train\_scaled.T, y\_train.reshape(1, -1), layer\_dims, num\_epochs=num\_epochs)

Function: train\_nn(X\_train\_scaled.T, y\_train.reshape(1, -1), layer\_dims, num\_epochs=num\_epochs)

Purpose: Trains the neural network using the specified training data and architecture.

* Inputs:
  + X\_train\_scaled.T: Transposed training data for correct shape.
  + y\_train.reshape(1, -1): Reshapes the target labels for compatibility.
  + layer\_dims: Architecture of the network.
  + num\_epochs: Number of training iterations.

**7.Making Predictions**

train\_predictions = predict(X\_train\_scaled.T, parameters)

test\_predictions = predict(X\_test\_scaled.T, parameters)

Purpose: Uses the trained parameters to predict labels for both the training and test datasets based on the input features

**8.Calculating Accuracy**

train\_accuracy = np.mean(train\_predictions == y\_train.reshape(1, -1))

test\_accuracy = np.mean(test\_predictions == y\_test.reshape(1, -1))

Purpose: Computes the accuracy of the model by comparing predicted labels to actual labels for both training and test datasets

**9.Displaying Accuracy Results**

print(f"Train accuracy: {train\_accuracy}")

print(f"Test accuracy: {test\_accuracy}")

Purpose: Prints the training and test accuracies to the console for evaluation of model performance.

**10.Plotting Cost vs. Number of Epochs**

plt.figure(figsize=(10, 5))

plt.plot(costs)

plt.title("Cost vs Number of Epochs")

plt.xlabel("Epochs")

plt.ylabel("Cost")

plt.tight\_layout()

plt.show()

Purpose: Visualizes the cost function over the training epochs, providing insight into how well the model is learning and whether it converges over time.

**11.Scatter Plot of True vs. Predicted Labels**

Function: plot\_pca\_scatter(X\_test\_scaled, y\_test, test\_predictions, title)

plot\_pca\_scatter(X\_test\_scaled, y\_test, test\_predictions, "Test Set: True vs Predicted Labels"

Purpose: Creates a scatter plot that compares true labels with predicted labels using PCA to visualize the data in a reduced dimensionality space.

**THANK YOU SO MUCH!**