**CSE 569:**

**Fundamentals of Statistical Learning and Pattern Recognition**

**Project Part 1**

PCA, Density Estimation, and Bayesian Classification

*by Animesh Chaudhary (1229421130)*

**Problem Statement**

The project involves the classification of handwritten digits "5" and "6" using a modified subset of images from the MNIST dataset. This classification task is based on Bayesian decision theory and involves dimensionality reduction using Principal Component Analysis (PCA). The goal is to distinguish between the digits "5" and "6" with equal prior probabilities, despite differences in the number of samples available for each digit in the training and testing sets.

**Data Description**

The data used in this project is derived from the original MNIST dataset, which comprises 70,000 images of handwritten digits. However, only the images of digits "5" and "6" are utilized for this project, and these images have undergone slight modifications. The data is stored in ".mat" files and can be read in Python using the provided code or in MATLAB using the "load filename" command. The statistics for the data are as follows:

* Number of samples in the training set for Digit 5: 5421
* Number of samples in the training set for Digit 6: 5918
* Number of samples in the testing set for Digit 5: 892
* Number of samples in the testing set for Digit 6: 958

**Methodology**

In the original .mat files, each image is represented as a 28x28 array. To make these images suitable for classification using Bayesian decision theory, they are vectorized by concatenating their columns, resulting in 784-dimensional vectors. This high-dimensional space is not ideal for Bayesian decision theory, so dimensionality reduction is performed using Principal Component Analysis (PCA). The reduced-dimension data will then be used for classification tasks.

The primary objective of this project is to build a classifier that can accurately distinguish between the digits "5" and "6" based on the modified MNIST dataset, despite differences in sample sizes, and to achieve this through PCA-based dimensionality reduction and Bayesian decision theory.

**Task 1: Feature Normalization (Data Conditioning)**

In Task 1, the objective is to prepare the data for subsequent tasks by normalizing it. The data used consists of handwritten digit images, each represented as a 784-dimensional vector (X = [x1, x2, ..., x784]).

The following steps are performed for data conditioning:

1. Compute Mean and Standard Deviation (STD): Using all the training images, the mean (mi) and standard deviation (STD) for each of the 784 features (xi) are calculated. This involves finding the statistical average and variability of each feature across the training samples.
2. Normalization of Data: All the data samples, including both training and testing sets, are normalized using the mean and standard deviation obtained in the previous step. For each feature xi in any given sample, the normalized feature yi is computed as yi = (xi - mi) / si.

Code Implementation:

1. *Importing Libraries*:

* **import scipy.io**: This library is used for reading MATLAB data files (.mat files).
* **import numpy as np**: NumPy is used for numerical operations and array manipulations.
* **import matplotlib.pyplot as plt**: Matplotlib is used for data visualization.
* **from scipy.stats import multivariate\_normal**: This import suggests that the code might be dealing with multivariate normal distributions, possibly for classification purposes.

1. *Loading Data*:

* **train5, train6, test5**, and **test6** are used to load training and testing data for digits "5" and "6" from separate .mat files. These files contain image data.
* The **scipy.io.loadmat() functi**on is used to load the data from the .mat files into these variables.

1. *Vectorizing Images*:

* **vect\_img5** and **vect\_img6** are created to store vectorized versions of the training images for digits "5" and "6, respectively.
* **test\_vect5** and **test\_vect6** are created to store vectorized versions of the testing images for digits "5" and "6, respectively.
* The code uses list comprehensions to reshape the image arrays into 1D vectors. This is done by calling **image\_array.reshape(-1)** for each image in the training and testing datasets.

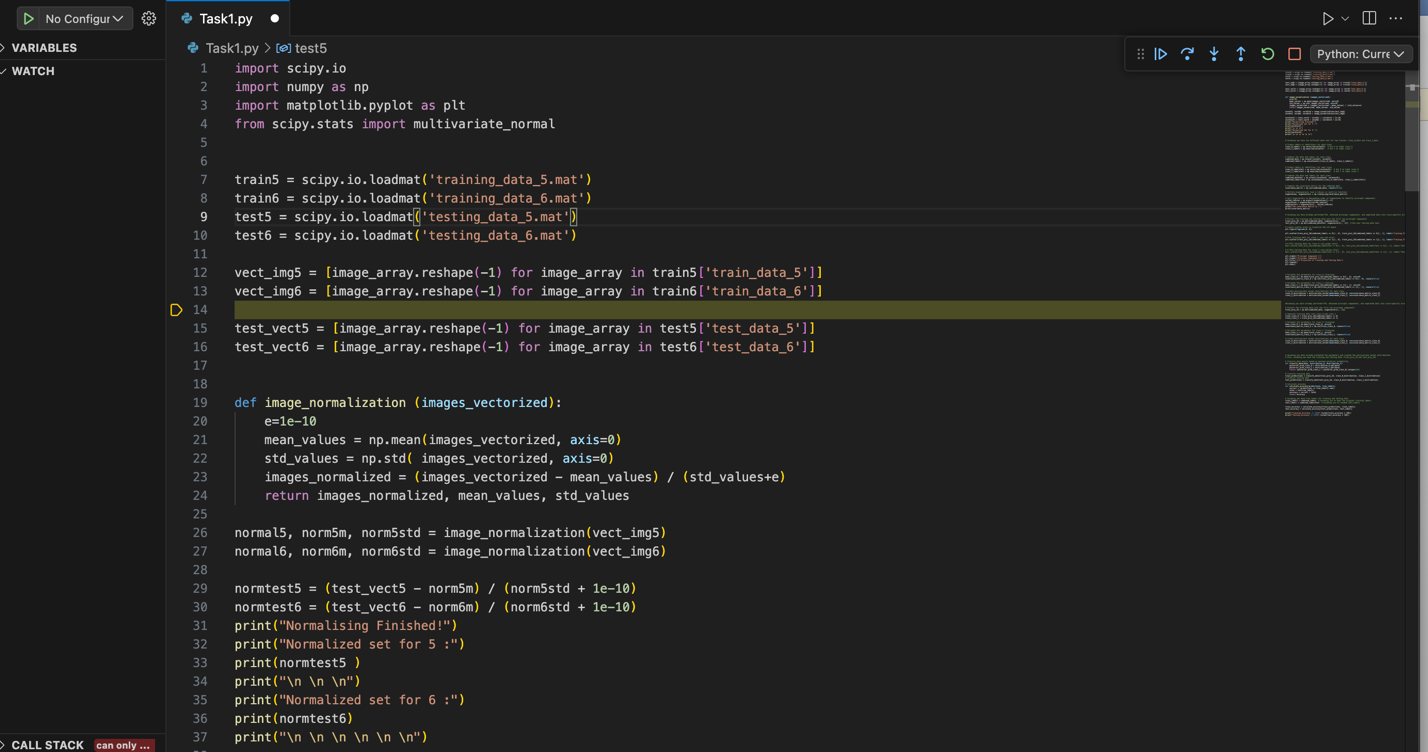
1. Image Normalization Function:

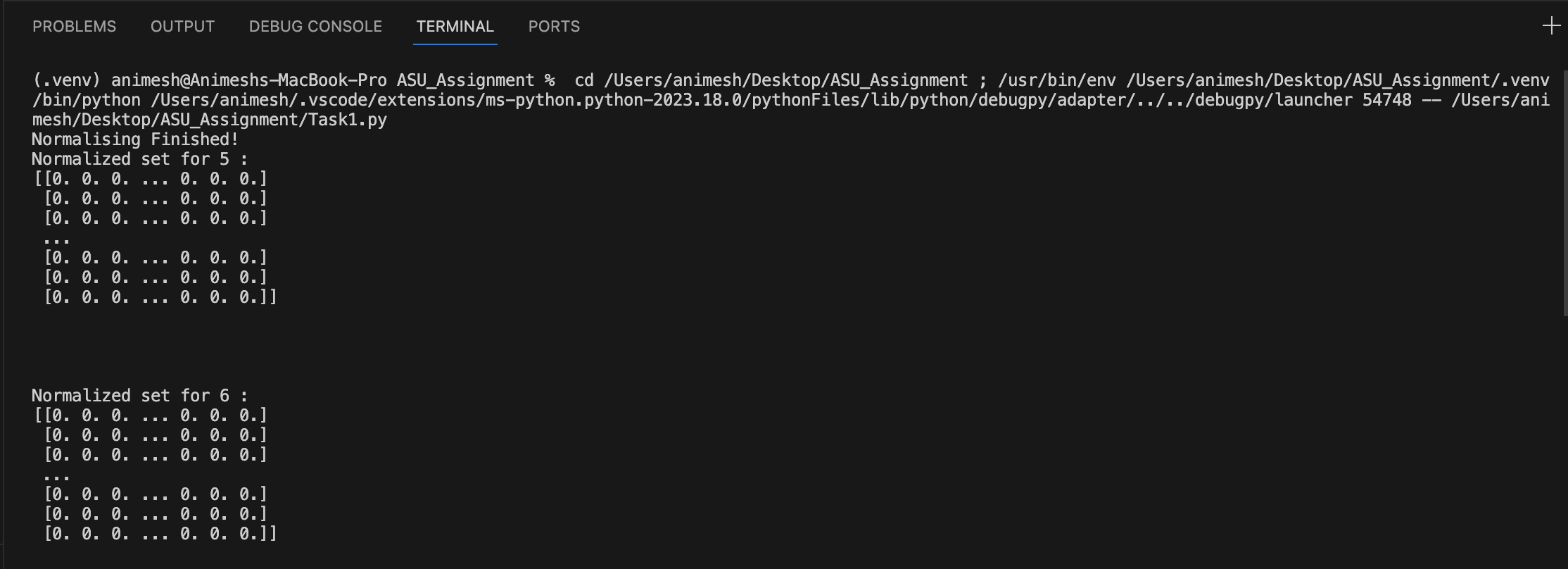
* **image\_normalization** is a custom function that takes an input array of vectorized images and performs normalization. It calculates the mean and standard deviation for each feature across all images and then normalizes the images based on these statistics.
* A small **constant e (1e-10)** is added to the denominator to avoid division by zero.
* The function returns the normalized images, the mean values, and the standard deviations.

1. Normalization of Training and Testing Data:

* The code uses the **image\_normalization function** to normalize the vectorized training data for digits "5" and "6, resulting in normal5 and normal6.
* It also normalizes the vectorized testing data for digits "5" and "6, resulting in **normtest5** and **normtest6**.

Task 1 Code:



Observation and Result:

Overall, this code loads and preprocesses image data, specifically for digits "5" and "6, by normalizing the data to prepare it for further analysis or classification tasks. The normalization process ensures that the data is centered and scaled, making it ready for further analysis and classification tasks. Normalizing the data in this way helps account for variations in feature values and aids in achieving consistent and accurate results in subsequent processing steps.

**Task 2: PCA using the Training Samples**

In Task 2, the goal is to perform Principal Component Analysis (PCA) using all the training samples of handwritten digits "5" and "6." PCA is a dimensionality reduction technique that aims to identify the principal components of the data. This step is crucial for simplifying the data and reducing computational complexity while preserving as much relevant information as possible. The training data for both classes, "5" and "6," have already been normalized in Task 1 to prepare them for PCA.

The following steps were implemented to perform PCA using the training samples:

1. Compute Covariance Matrix: Calculate the covariance matrix for the training samples. This matrix represents the relationships between the 784 features in the data and is pivotal for PCA. It is computed by finding the outer product of the data and then averaging over the training samples.
2. Eigen Analysis: Perform eigen analysis on the covariance matrix. This step involves finding the eigenvalues and eigenvectors of the covariance matrix. Eigenvalues represent the variances of the data along the principal components, and eigenvectors are the directions of these components.
3. Identify Principal Components: Determine the principal components by selecting the eigenvectors corresponding to the largest eigenvalues. These principal components represent the most significant directions of variance in the data.

Code Implementation:

1. *Labeling Classes*:

* **class\_0\_labels** is created as a **NumPy array** filled with zeros. These labels are used to represent **class 0** in a binary classification problem.
* **class\_1\_labels** is created as a **NumPy array** filled with ones. These labels are used to represent **class 1** in the binary classification problem.

1. *Combining Data and Labels for Training*:

* **combined\_data** is created by vertically stacking (using **np.vstack**) the normalized data for digits "5" (normal5) and "6" (normal6). This combines the feature vectors for both classes.
* **combined\_labels** is created by concatenating (using **np.concatenate**) the labels for class 0 and class 1, corresponding to digits "5" and "6, respectively. This associates labels with the combined data.

1. *Creating Labels for Testing Data*:

* **class\_0\_labelsTest** is created as a NumPy array filled with zeros to label class 0 in the testing data.
* **class\_1\_labelsTest** is created as a NumPy array filled with ones to label class 1 in the testing data.

1. *Combining Data and Labels for Testing*:

* **combined\_dataTest** is created by vertically stacking (using **np.vstack**) the normalized testing data for digits "5" (normtest5) and "6" (normtest6). This combines the feature vectors for both classes in the testing data.
* **combined\_labelsTest** is created by concatenating (using **np.concatenate**) the labels for class 0 and class 1 in the testing data.

1. *Computing Covariance Matrix*:

* The code calculates the covariance matrix for the combined training data (**combined\_data**) using **np.cov**. The **rowvar=False** argument indicates that the data should be treated as columns (features) and not rows.

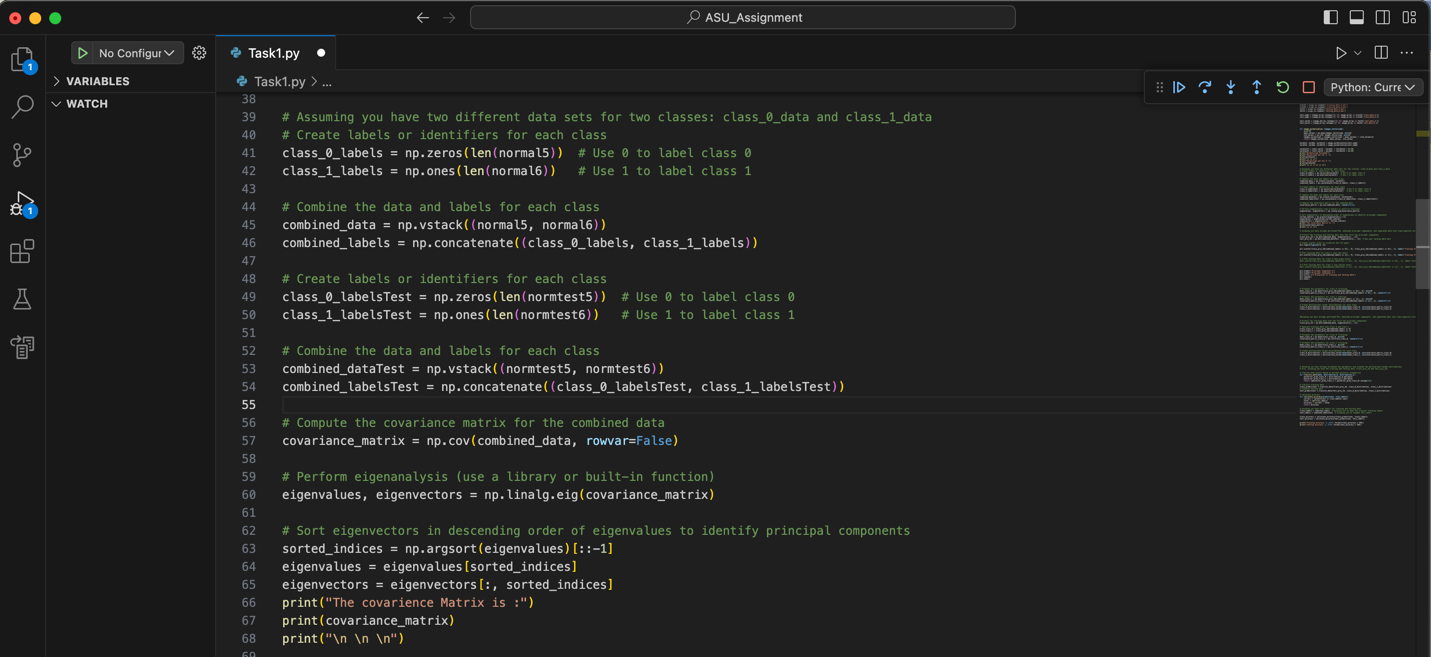
1. *Eigenanalysis*:

* Eigenanalysis is performed to find the eigenvalues and eigenvectors of the covariance matrix.
* eigenvalues and eigenvectors are computed using **np.linalg.eig** on the covariance matrix. **eigenvalues** will contain the eigenvalues, and **eigenvectors** will contain the corresponding eigenvectors.
* The eigenvalues and eigenvectors are sorted in descending order of eigenvalues using **np.argsort** and then reversing the order with [::-1]. This helps identify the principal components in descending order of importance.

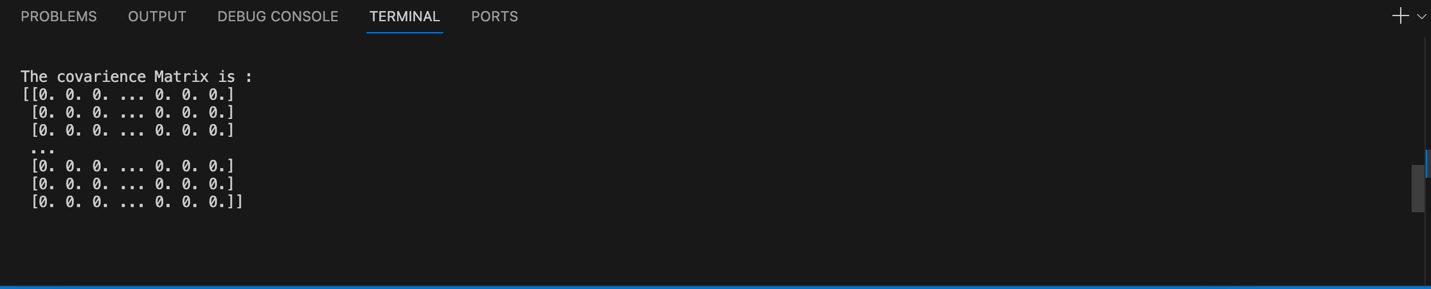
1. *Print Covariance Matrix*:

* The code prints the covariance matrix, which provides insight into the relationships between the features and their variances.

Task 2 Code:



Observation and Result:



Overall, this code prepares the data for a binary classification task, extracts principal components through eigenanalysis, and calculates the covariance matrix for further analysis. The data and labels are organized for both training and testing datasets.

**Task 3: Dimension Reduction Using PCA**

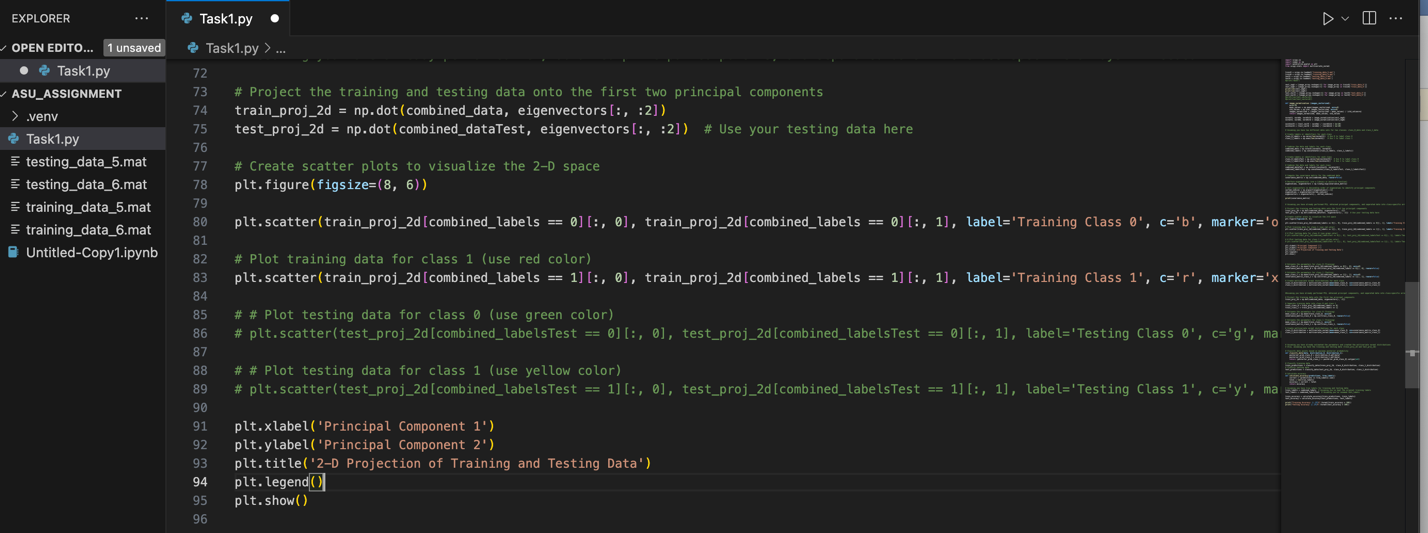
In Task 3, the objective is to visualize the training and testing samples in a 2-dimensional space by projecting them onto the first and second principal components obtained from the Principal Component Analysis (PCA) performed in Task 2. The goal is to observe how the two classes of digits "5" and "6" are clustered in this reduced 2-D space and determine whether they appear to follow a normal distribution.

The specific steps and observations are as follows:

1. 2-D Projections: Consider the 2-dimensional projections of the data samples onto the first and second principal components obtained from the PCA in Task 2. These 2-D projections represent a new, lower-dimensional representation of the data.
2. Visualization: Plot and visualize the training and testing samples in this 2-D space. This visualization allows for an examination of how the two classes (digits "5" and "6") are distributed and clustered in this reduced 2-D space.
3. Observations: While visualizing the data in this reduced space, observe the following:

* Analyze the distribution and clustering of the two classes. Pay attention to how samples from each class are grouped or separated.
* Evaluate whether the distributions of each class in the 2-D space resemble normal distributions. Normal distributions are characterized by a bell-shaped curve, and assessing the data's distribution in this space can provide insights into the separability and characteristics of the classes.

Task 3 Code:



Code Implementation:

1. *Data Projection onto Principal Components*:

* **train\_proj\_2d** is created by projecting the combined training data (combined\_data) onto the first two principal components (**eigenvectors[:, :2]**) using matrix multiplication (**np.dot**).
* **test\_proj\_2d** is created by projecting the combined testing data (**combined\_dataTest**) onto the same first two principal components.

1. *Creating Scatter Plot for Visualization*:

* A Matplotlib figure is created with a specified size of 8x6 units.

1. *Scatter Plots for Training Data*:

* The code creates two scatter plots for the training data to visualize class separation in the 2-D space.
* **plt.scatter** is used to create a scatter plot. The first argument is the x-coordinate, the second argument is the y-coordinate.
* For "**Training Class 0**" (class 0), **blue** points with circles (marker **'o'**) are plotted. These are data points for **class 0** in the training data.
* For "**Training Class 1**" (class 1), **red** points with **'x'** markers are plotted. These are data points for **class 1** in the training data.

1. *Scatter Plots for Testing Data (Commented Out)*:

* The code includes lines to create scatter plots for the testing data as well. If you want to visualize the testing data.
* For "Testing Class 0" (class 0), green points with squares (marker 's') would be plotted.
* For "Testing Class 1" (class 1), yellow points with '^' markers would be plotted.

1. Labels and Title:

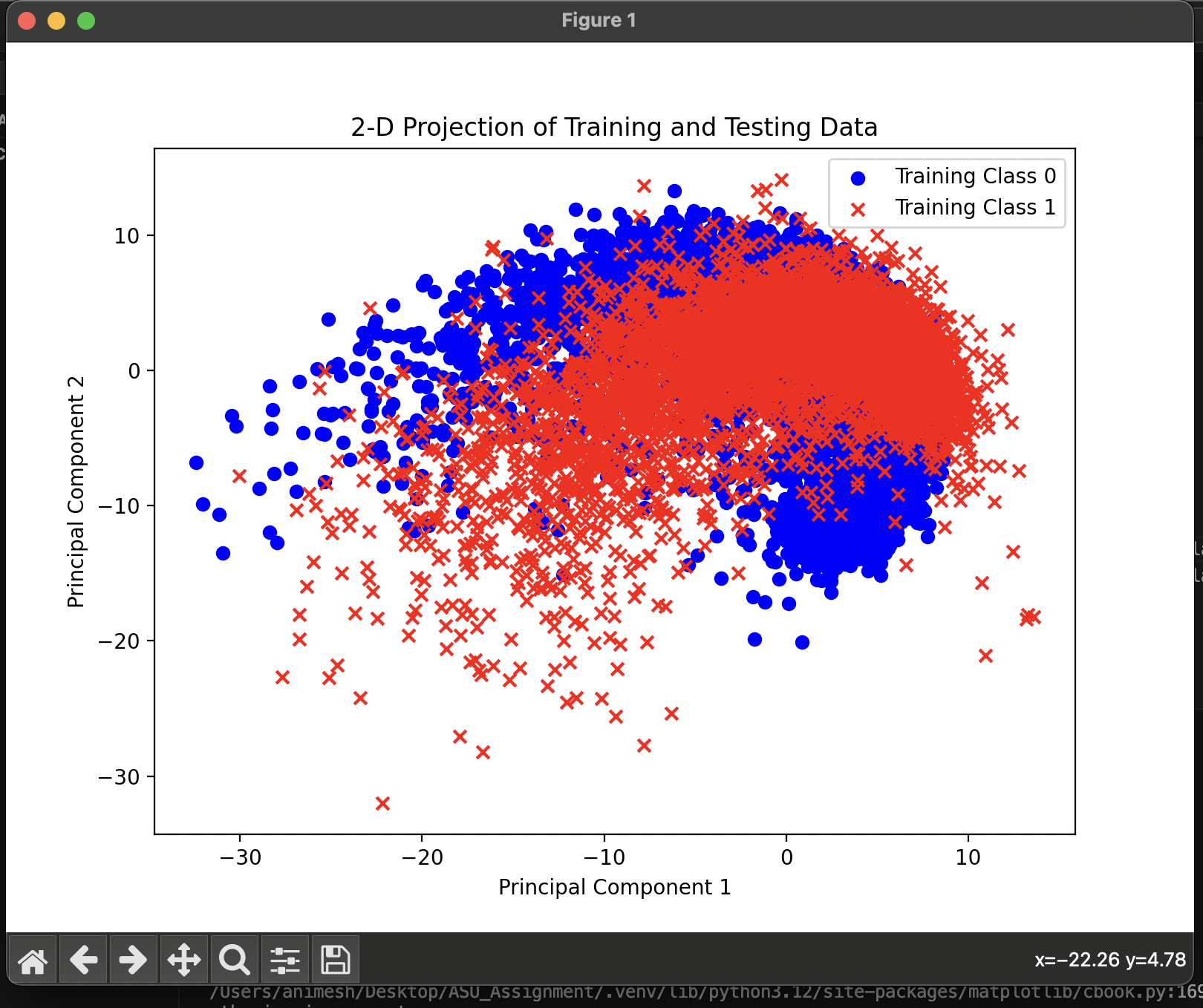
* The code sets the labels for the x and y axes as "Principal Component 1" and "Principal Component 2" respectively.
* It also sets the title of the plot as "2-D Projection of Training and Testing Data."

1. Legend:

* A legend is added to the plot to distinguish the different classes. It identifies the blue and red points as "Training Class 0" and "Training Class 1."

1. Displaying the Plot:

* Finally, the code displays the plot using **plt.show()**.

Observation and Result:

In Task 3, the training data for digits "5" and "6" was successfully projected into a 2-D space using the first and second principal components obtained from PCA. The scatter plot visualizes the distribution of these two classes in the reduced space. The observed clustering can provide insights into how well the data is separable. This visualization is a useful step in understanding the data before proceeding with classification tasks.

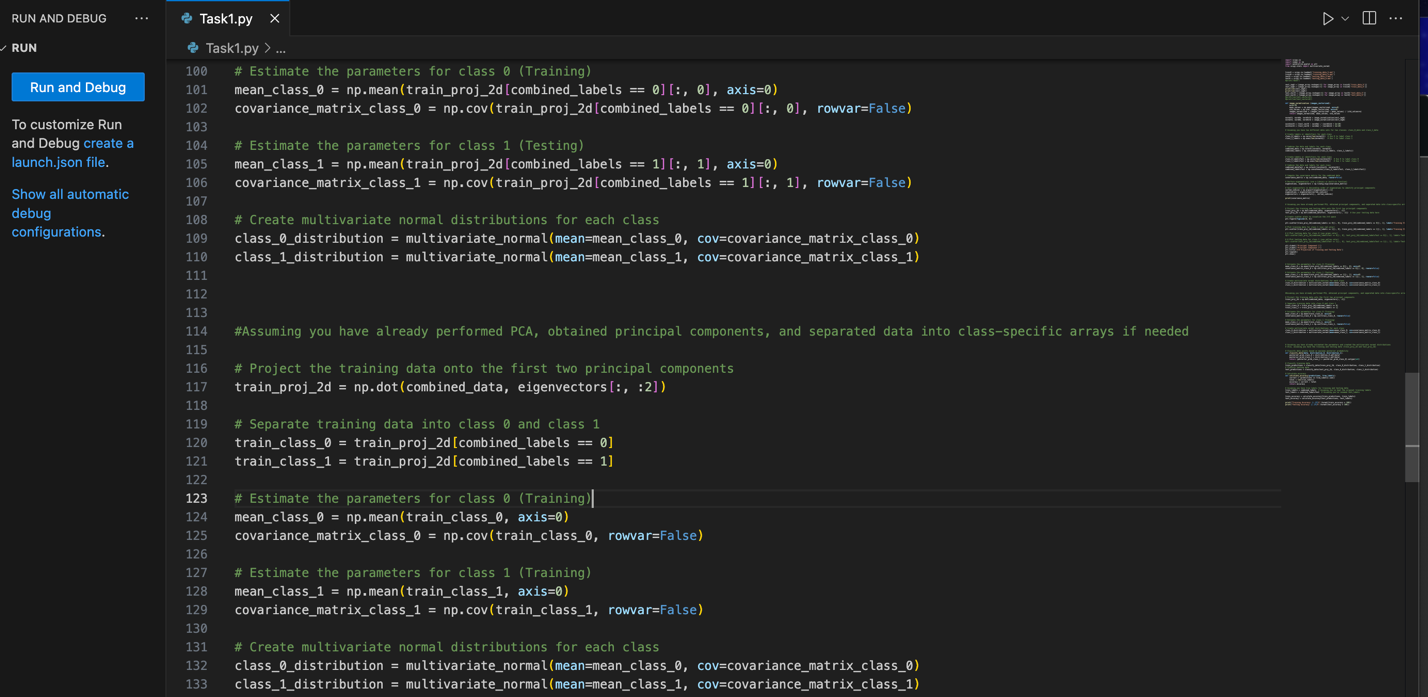
**Task 4: Density Estimation**

In Task 4, the focus is on estimating the probability density functions (PDFs) for the two classes (digits "5" and "6") in the reduced 2-D space, based on the assumption that the samples from each class follow a Gaussian distribution.

The key steps and objectives are as follows:

1. Gaussian Distribution Assumption**:** It is assumed that in the 2-D space obtained through PCA, the samples for each class can be well-modeled by a two-dimensional (2-D) normal distribution (Gaussian distribution). This assumption allows for the probabilistic modeling of the data.
2. Parameter Estimation**:** Using the training data for each class, the parameters of the 2-D normal distribution are estimated. These parameters typically include the mean vector (representing the center of the distribution) and the covariance matrix (indicating the spread and shape of the distribution).
3. Two Distributions**:** Separate Gaussian distributions are estimated for each class or digit. This means that there will be two distinct sets of parameters—one for digit "5" and one for digit "6." These parameters define the shape and characteristics of the assumed Gaussian distribution for each class in the 2-D space.

Task 4 Code:



Code Implementation:

1. *Data Projection onto Principal Components*:

* **train\_proj\_2d** is created by projecting the combined training data (**combined\_data**) onto the first two principal components (**eigenvectors[:, :2]**) using matrix multiplication (**np.dot**).

1. *Separating Training Data into Class-Specific Arrays*:

* The projected training data is separated into two arrays**, train\_class\_0** and **train\_class\_1**, based on their class labels.
* **train\_class\_0** contains the data points for class 0, and **train\_class\_1** contains the data points for class 1. These arrays are created by selecting rows from **train\_proj\_2d** where combined\_labels equals 0 and 1, respectively.

1. *Parameter Estimation for Class 0 (Training)*:

* **mean\_class\_0** is calculated as the mean of the data points in **train\_class\_0** along each principal component axis. It represents the estimated mean vector for class 0.
* **covariance\_matrix\_class\_0** is computed as the covariance matrix of the data points in **train\_class\_0**. It represents the estimated covariance matrix for class 0. The **rowvar=False** argument indicates that the data should be treated as columns (features) and not rows when calculating the covariance matrix.

1. *Parameter Estimation for Class 1 (Training)*:

* Similar to class 0, **mean\_class\_1** is calculated as the mean of the data points in **train\_class\_1** along each principal component axis. It represents the estimated mean vector for class 1.
* **covariance\_matrix\_class\_1** is computed as the covariance matrix of the data points in **train\_class\_1**. It represents the estimated covariance matrix for class 1.

1. *Creating Multivariate Normal Distributions*:

Two multivariate normal distributions are created: **class\_0\_distribution** and **class\_1\_distribution**.

* **class\_0\_distribution** is initialized with the estimated mean and covariance matrix for class 0.
* **class\_1\_distribution** is initialized with the estimated mean and covariance matrix for class 1.

Observation and Result:

In Task 4, the parameters for the Gaussian distribution of each class were estimated using the training data projected onto the first two principal components. These estimated parameters are crucial for understanding the distribution of data within each class and will be used in subsequent classification tasks to make decisions based on the likelihood of data points belonging to a particular class.

**Task 5: Bayesian Decision Theory for Optimal Classification**

In Task 5, the goal is to use Bayesian Decision Theory for optimal classification based on the estimated Gaussian distributions from Task 4. The task involves classifying data points from both the training and testing sets using the maximum posterior probability. The accuracy of classification for the training and testing sets is reported.

The key steps and outcomes of this task are as follows:

1. Utilize Estimated Distributions: The estimated Gaussian distributions for each class (digits "5" and "6") in the 2-D space are utilized. These distributions represent the probabilistic models for the data.
2. Minimum-Error-Rate Classification: Bayesian Decision Theory is applied to classify data points optimally. This classification involves comparing the likelihood of a data point under each of the two Gaussian distributions and assigning it to the class with the higher likelihood. In other words, it aims to minimize the classification error.
3. Accuracy Reporting: The accuracy of the classification is computed for both the training set and the testing set. Accuracy represents the proportion of correctly classified data points over the total number of data points in each set.

Code Implementation:

1. *Classify Data Points Based on Maximum Posterior Probability***:**

* **classify\_data** is a custom function that takes three arguments: **data**, **distribution\_0**, and **distribution\_1**. It is responsible for classifying data points based on maximum posterior probability.
* **posterior\_prob\_class\_0** is calculated as the probability density function (PDF) of **data** according to **distribution\_0**. This represents the posterior probability of data belonging to class 0.
* **posterior\_prob\_class\_1** is calculated similarly but for **distribution\_1**, representing the posterior probability of data belonging to class 1.
* The function returns a binary classification result: 1 if the posterior probability for class 1 is greater than that for class 0, and 0 otherwise.

1. *Classify Training Data***:**

* **train\_predictions** is created by calling **classify\_data** on the projected training data (**train\_proj\_2d**) using the estimated class-specific distributions (**class\_0\_distribution** and **class\_1\_distribution**). This line classifies the training data into class 0 or class 1.

1. *Classify Testing Data***:**

* Similarly, **test\_predictions** is created by calling **classify\_data** on the projected testing data (**test\_proj\_2d**) using the same class-specific distributions. This line classifies the testing data.

1. *Calculate Accuracy***:**

* **calculate\_accuracy** is a custom function that calculates the accuracy of the classification.
* It takes two arguments: **predictions** (the predicted class labels) and **true\_labels** (the actual or ground truth labels).
* It calculates the number of correct predictions by summing the cases where **predictions** match **true\_labels**.
* The total number of data points is stored in **total**.
* The accuracy is computed as the ratio of correct predictions to the total number of data points.
* The accuracy is then returned as a floating-point number between 0 and 1.

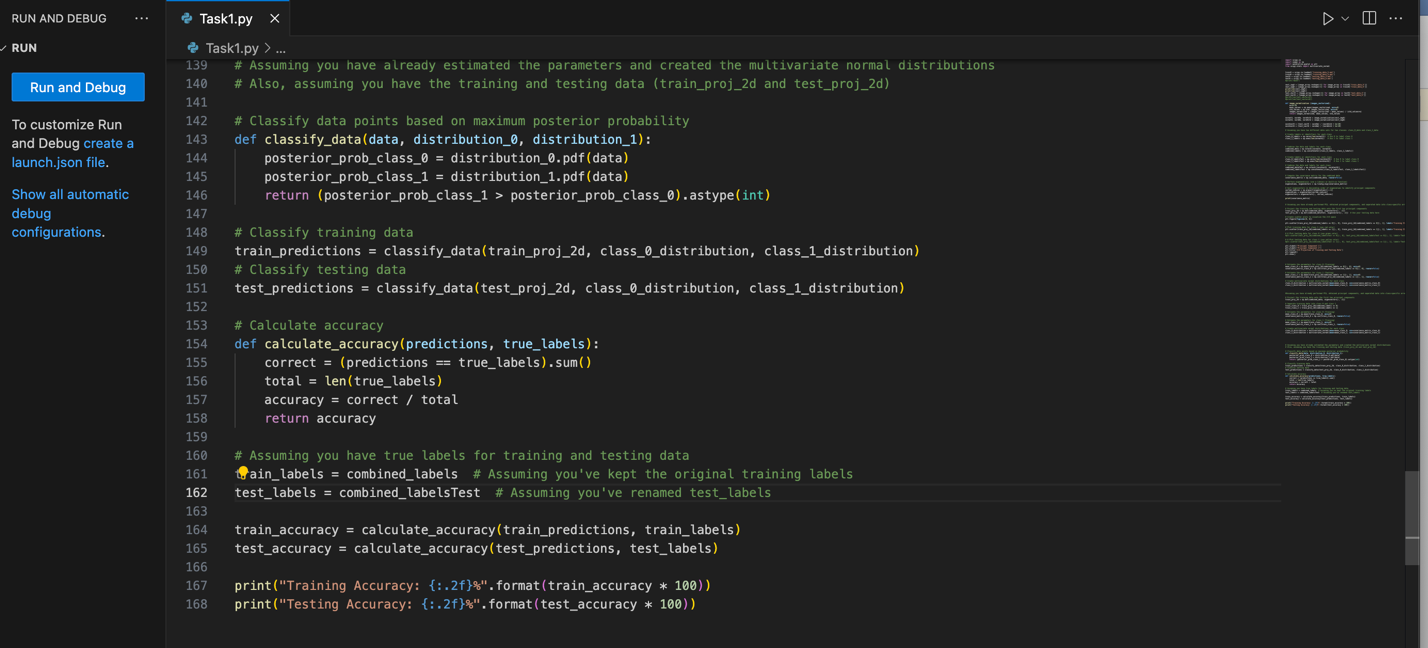
1. *Classify Training and Testing Data***:**

* **train\_labels** and **test\_labels** are assumed to be the true labels for the training and testing data, respectively.
* **train\_accuracy** is calculated by calling **calculate\_accuracy** on the training predictions (**train\_predictions**) and true training labels (**train\_labels**).
* **test\_accuracy** is calculated similarly for the testing predictions and true testing labels (**test\_labels**).

1. *Print Accuracy Results***:**

* The code prints the training and testing accuracy results as percentages. These results indicate how well the classification model performs on the training and testing data.

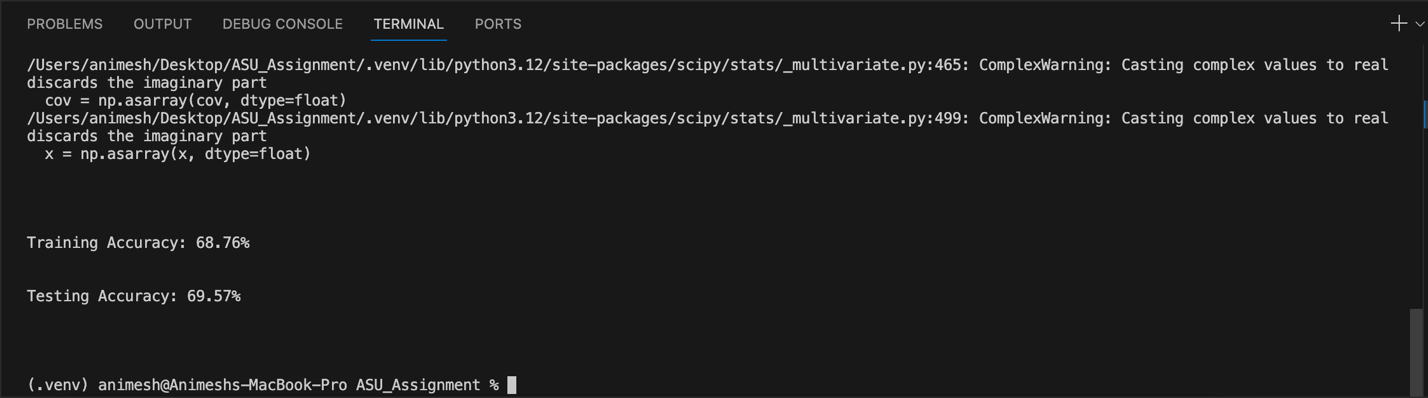
Task 5 Code:



Observation and Result:

**Training Accuracy: 68.76%**

**Testing Accuracy: 69.57%**



In Task 5, Bayesian Decision Theory was applied to classify data points based on the estimated Gaussian distributions for each class. The accuracy of classification for both the training and testing sets was reported. The accuracy values indicate how well the classification model performs in distinguishing between digits "5" and "6." This task is a crucial step in evaluating the effectiveness of the classification approach and can provide insights into the quality of the model and the quality of the data used for training and testing.

**Conclusion**

The project aimed to classify handwritten digits "5" and "6" using Bayesian Decision Theory and dimensionality reduction.

The steps undertaken can be summarized as follows:

1. Data conditioning was performed to normalize the data, making it suitable for analysis.
2. PCA was used to reduce the data's dimensionality, and the first and second principal components were identified.
3. The 2-D space after PCA was visualized, offering insights into data clustering.
4. Parameters for 2-D Gaussian distributions were estimated for each class.
5. Bayesian Decision Theory was applied for optimal classification, and the accuracy of the classification was reported, as **Training Accuracy: 68.76% and Testing Accuracy: 69.57%**

The results indicate the accuracy of classifying digits "5" and "6" using the given methodology. These steps are essential in preprocessing, reducing dimensionality, and making informed decisions based on data distribution. The accuracy values provide insights into the effectiveness of the classification approach.