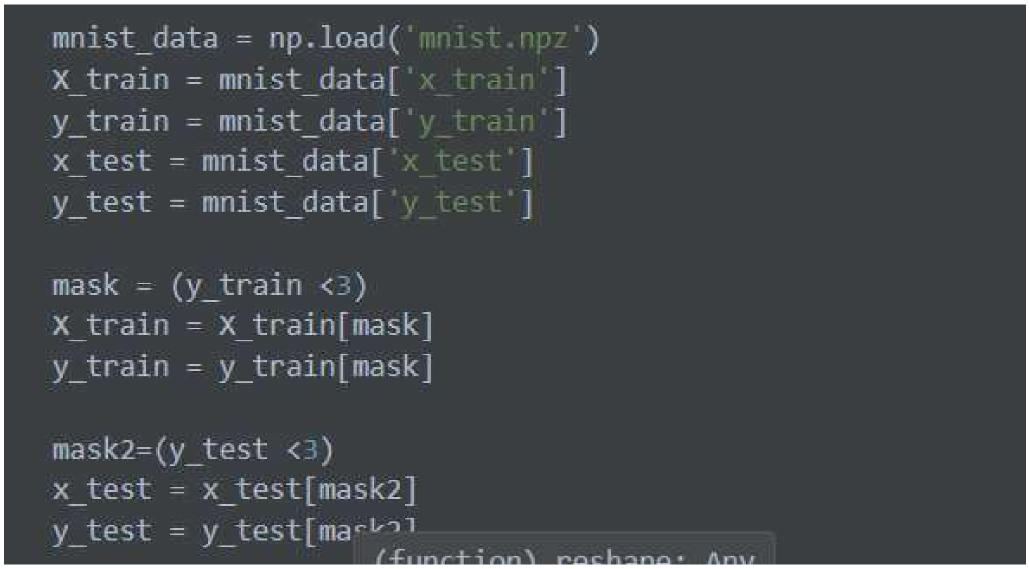
SML
Assignment 2
Vickey kumar
2021299

Question 1:
Use https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz MNIST dataset for this question and perform following tasks.
• It has all in all 60K train samples from 10 classes and 10K test samples. 10 classes are digits from 0-9. Labels or classes for all samples in train and test set is available.
We reduce the class buy only 0 ,1, 2

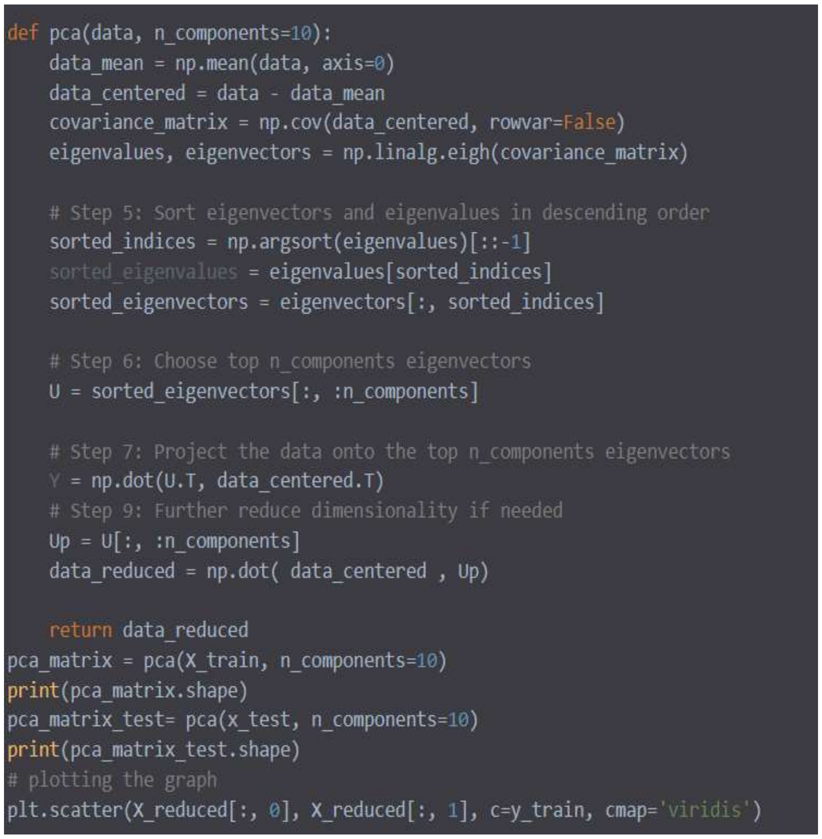
According to the question , i make a class list of “0’ , “1” and so on to “9”
To store the data of y_train which is labels and x_tain data ,



```
mnist_data = np.load('mnist.npz')
x_train = mnist_data['x_train']
y_train = mnist_data['y_train']
x_test = mnist_data['x_test']
y_test = mnist_data['y_test']

mask = (y_train <3)
x_train = x_train[mask]
y_train = y_train[mask]

mask2=(y_test <3)
x_test = x_test[mask2]
y_test = y_test[mask2]
                (function) reshape: Any
```

•

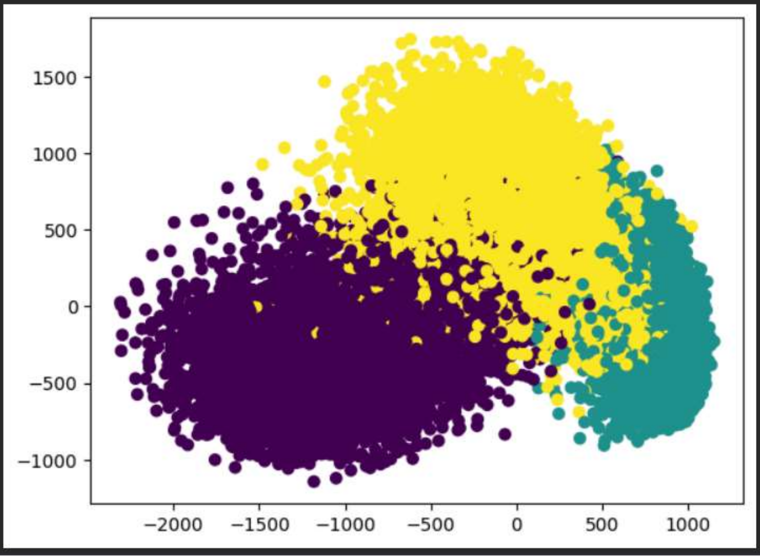
Now applied the pca on the 3 class 👍



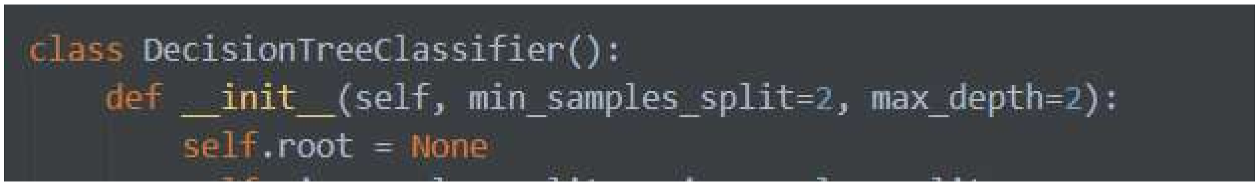
```python
def pca(data, n_components=10):
    data_mean = np.mean(data, axis=0)
    data_centered = data - data_mean
    covariance_matrix = np.cov(data_centered, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

    # Step 5: Sort eigenvectors and eigenvalues in descending order
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvalues = eigenvalues[sorted_indices]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]

    # Step 6: Choose top n_components eigenvectors
    U = sorted_eigenvectors[:, :n_components]

    # Step 7: Project the data onto the top n_components eigenvectors
    Y = np.dot(U.T, data_centered.T)
    # Step 9: Further reduce dimensionality if needed
    Up = U[:, :n_components]
    data_reduced = np.dot( data_centered , Up)

    return data_reduced
pca_matrix = pca(X_train, n_components=10)
print(pca_matrix.shape)
pca_matrix_test= pca(x_test, n_components=10)
print(pca_matrix_test.shape)
# plotting the graph
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y_train, cmap='viridis')
```

Which calculate the data of the X-train and X-test
And plotting the data

```
<matplotlib.collections.PathCollection at 0x207c6d82e10>
```

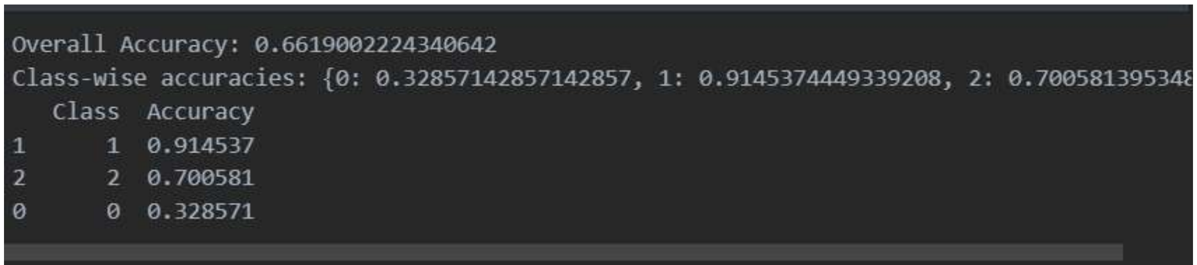


Now I create the decisiontreeClassifier()

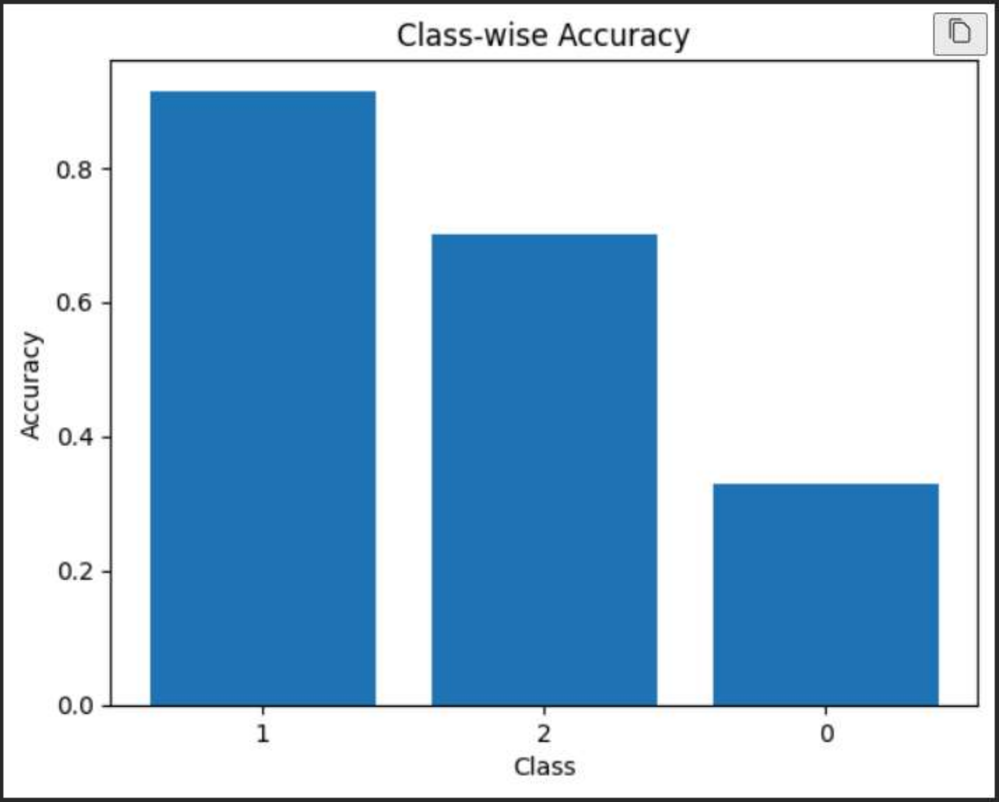


```
class DecisionTreeClassifier():
    def __init__(self, min_samples_split=2, max_depth=2):
        self.root = None
```

Which have all required def function to calculate the split , build_gini and so on…
And we call the x_test and y_test data on it
And get the result



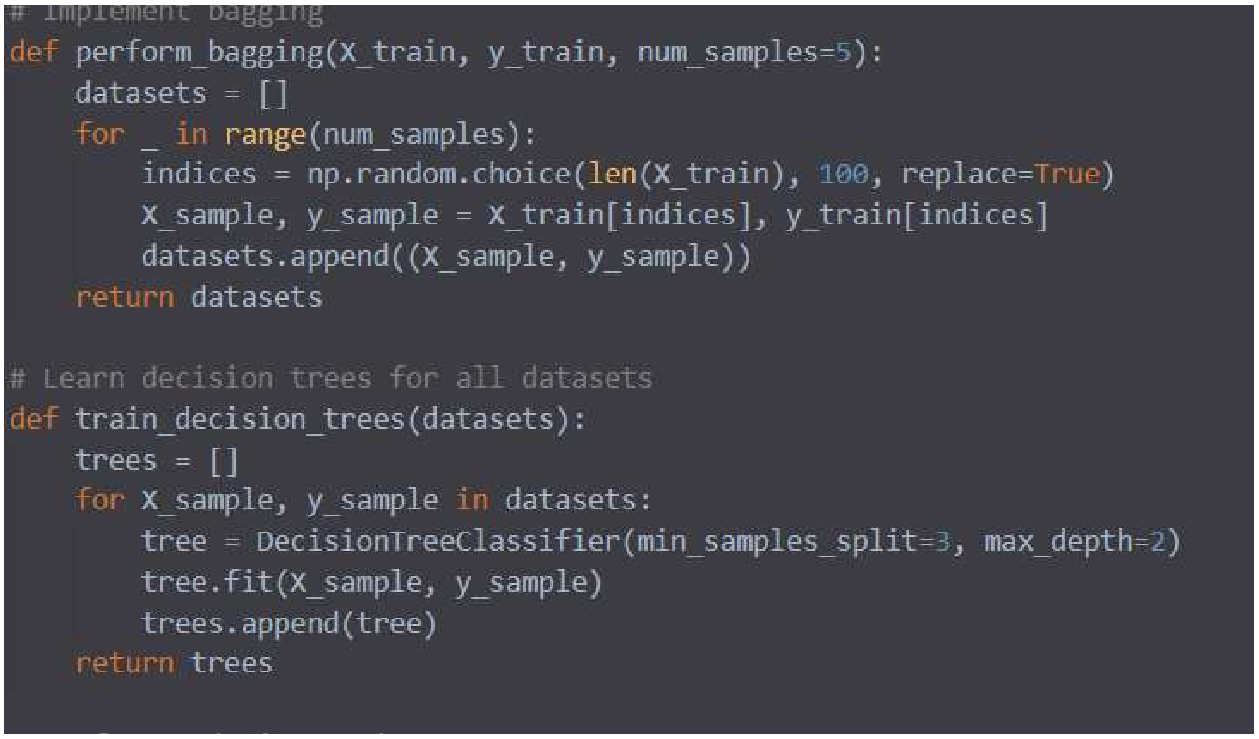
```
Overall Accuracy: 0.6619002224340642
Class-wise accuracies: {0: 0.32857142857142857, 1: 0.9145374449339208, 2: 0.700581395348
   Class  Accuracy
1      1  0.914537
2      2  0.700581
0      0  0.328571
```



And the last part:
Now use bagging, develop 5 different datasets from the original dataset.
Learn trees for all these datasets. For test samples, use majority voting
(atleast 3 trees should predict the same class) to find the class of a given

sample. In case there is a tie, that is two trees predict one class and other
two trees predict another class, then you can choose either of the classes.
Report the total accuracy and class-wise accuracy.

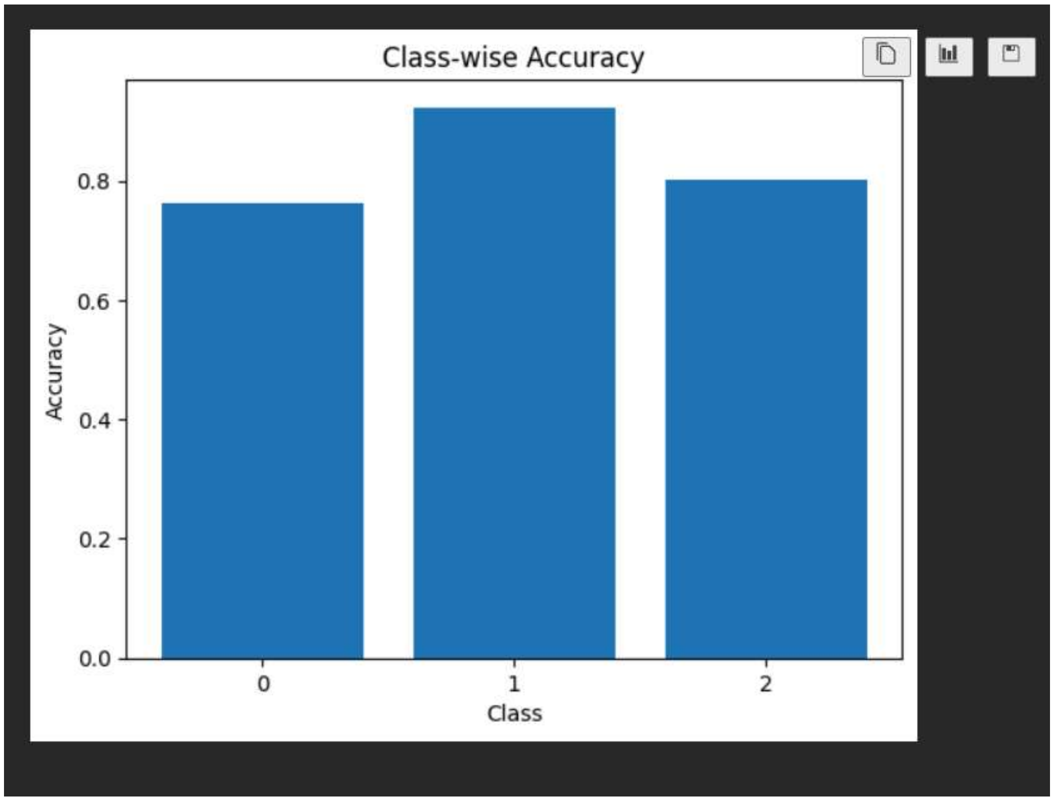
We have to perfom_bagging function and train_decision and implement the bagging and finally



```python
# Implement bagging
def perform_bagging(X_train, y_train, num_samples=5):
    datasets = []
    for _ in range(num_samples):
        indices = np.random.choice(len(X_train), 100, replace=True)
        X_sample, y_sample = X_train[indices], y_train[indices]
        datasets.append((X_sample, y_sample))
    return datasets

# Learn decision trees for all datasets
def train_decision_trees(datasets):
    trees = []
    for X_sample, y_sample in datasets:
        tree = DecisionTreeClassifier(min_samples_split=3, max_depth=2)
        tree.fit(X_sample, y_sample)
        trees.append(tree)
    return trees
```

we got the accuracy

Total Accuracy: 0.8334922148077534
Class 0 Accuracy: 0.7622448979591837
Class 1 Accuracy: 0.9233480176211454
Class 2 Accuracy: 0.8023255813953488


Class-wise Accuracy
Accuracy
Class
0.0
0.2
0.4
0.6
0.8
0
1
2