**Ex4 - Classification of Email spam and MNIST data**

**GitHub Link:**

[GitHub Link](https://github.com/Ojus999/Machine-Learning-Sem-6/tree/main)

**Colab Links:**

[4.1 and 4.3](https://colab.research.google.com/drive/1y6s-LNbCCJAekVY-aowbxfK3nDtYDELn?usp=sharing)

[4.2](https://colab.research.google.com/drive/1OKi67wllS_bUAODcfjAPlHB4TJlQPmeY?usp=sharing)

**Aim:**

To develop a python program

(i) To classify Emails as Spam or Ham

(ii) To recognize the digits of the MNIST dataset

Using Support Vector Machine (SVM) Model

**4.1, 4.3 Classification of Email Spam or Ham using Support Vector Machine (SVM) and Naïve Bayes Algorithm**

**Clone GitHub Repo For Data**

!git clone https://github.com/Ojus999/Machine-Learning-Sem-6.git

**Import Dependencies**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn import metrics

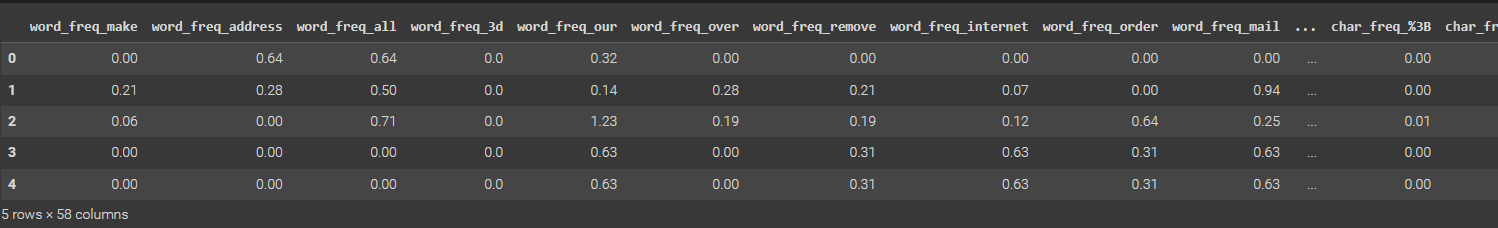
from sklearn import svm

**Read Data**

df = pd.read\_csv("/content/Machine-Learning-Sem-6/Ex 4/spambase\_csv.csv")

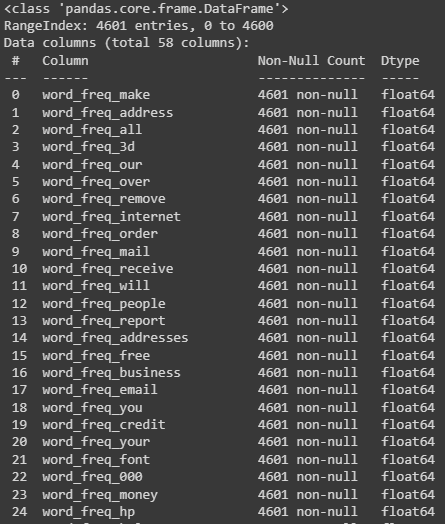
**Read First Few Rows**

df.head()



**DataFrame Info**

df.info()



**Data Visualization**

**Data Distribution**

class\_counts = df['class'].value\_counts()

plt.bar(class\_counts.index, class\_counts.values)

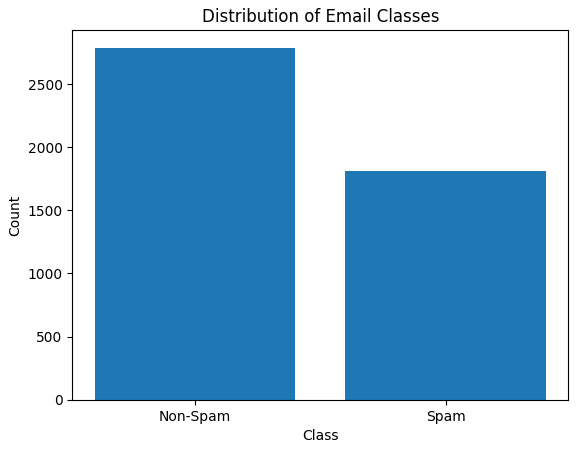
plt.xlabel('Class')

plt.ylabel('Count')

plt.title('Distribution of Email Classes')

plt.xticks(class\_counts.index, ['Non-Spam', 'Spam'])

plt.show()



**Correlation Heatmap**

correlation\_matrix = df.corr()

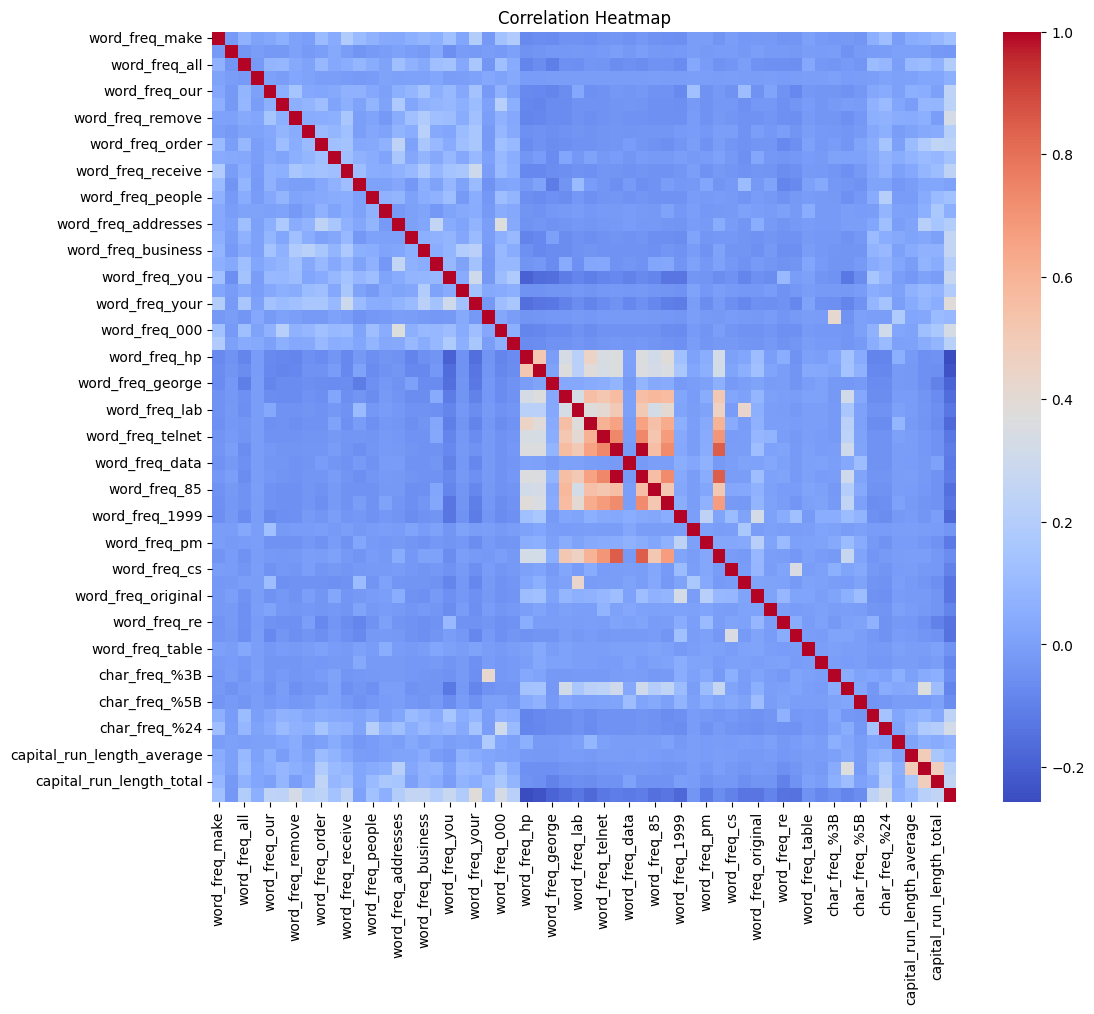
# Create a heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Heatmap')

plt.show()



**Histograms & Boxplot**

word\_freq\_columns = df.loc[:, 'word\_freq\_make':'word\_freq\_conference'].columns

index = 1

# Plot boxplots for word frequency features

plt.figure(figsize=(12, 8))

sns.boxplot(data=df[word\_freq\_columns[index]])

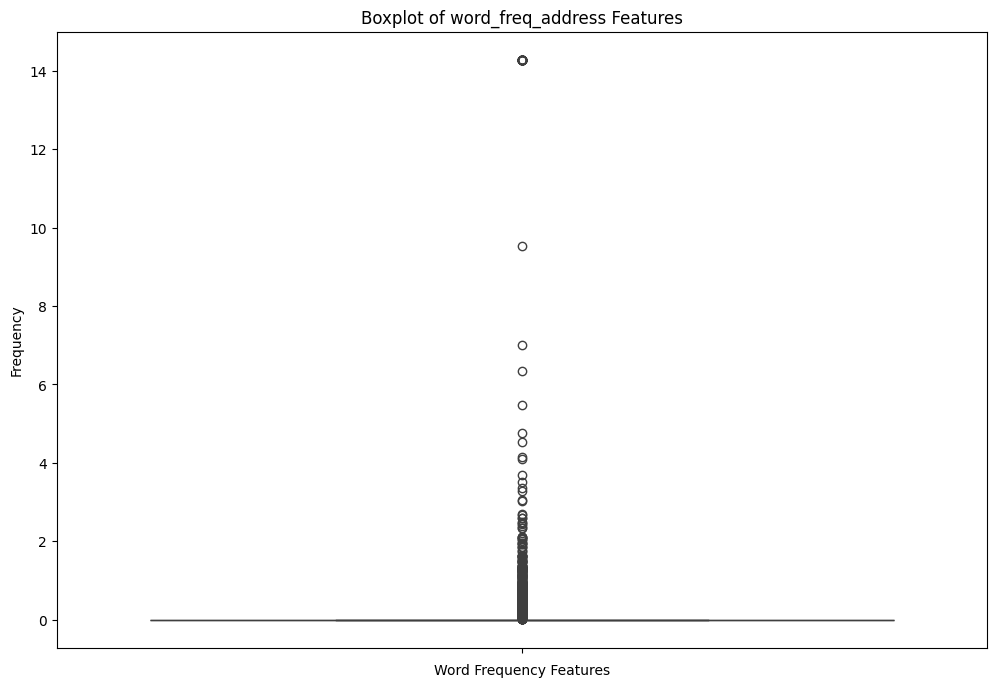
plt.xlabel('Word Frequency Features')

plt.ylabel('Frequency')

plt.title(f'Boxplot of {word\_freq\_columns[index]} Features')

plt.xticks(rotation=45)

plt.show()



word\_freq\_columns = df.loc[:, 'word\_freq\_make':'word\_freq\_conference'].columns

index = 0

# Plot histogram for the selected word frequency feature

plt.figure(figsize=(12, 8))

sns.histplot(data=df[word\_freq\_columns[index]], bins=20) # Adjust bins and kde as needed

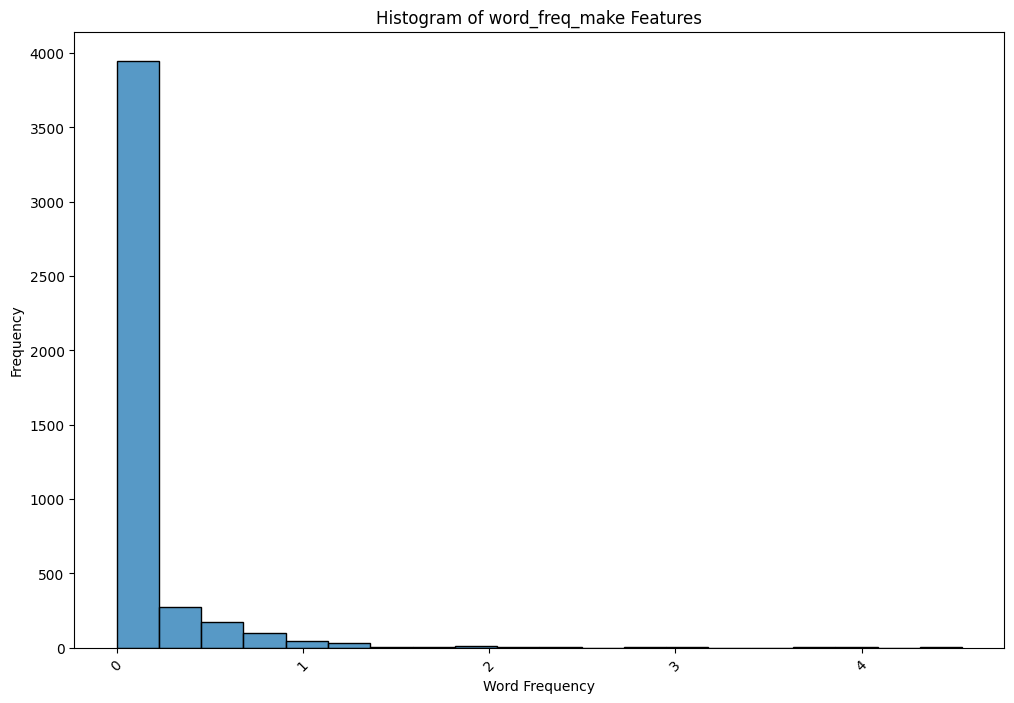
plt.xlabel('Word Frequency')

plt.ylabel('Frequency')

plt.title(f'Histogram of {word\_freq\_columns[index]} Features')

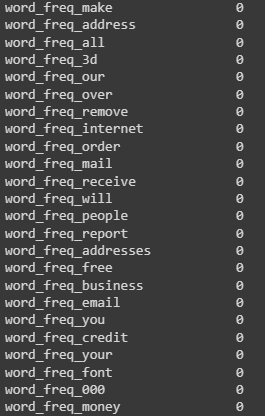
plt.xticks(rotation=45)

plt.show()



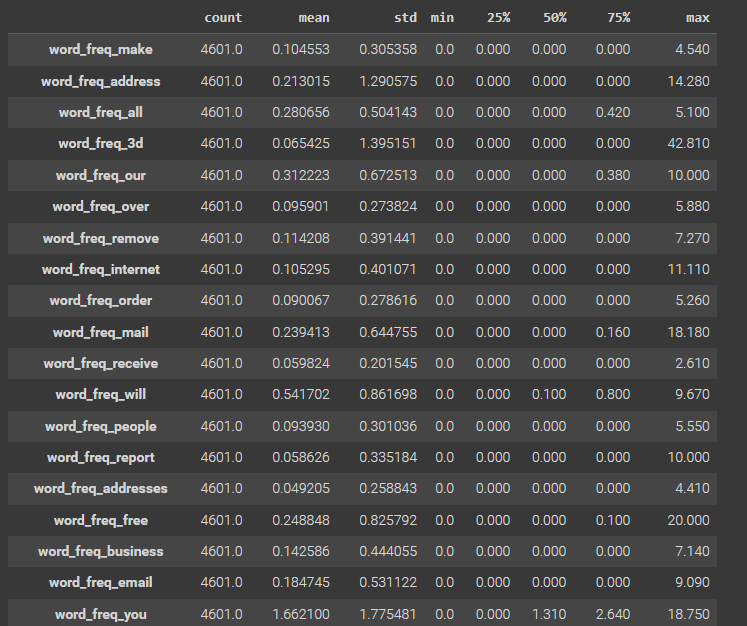
**Null Values**

df.isnull().sum()



**Statistics of Data**

df.describe().transpose()

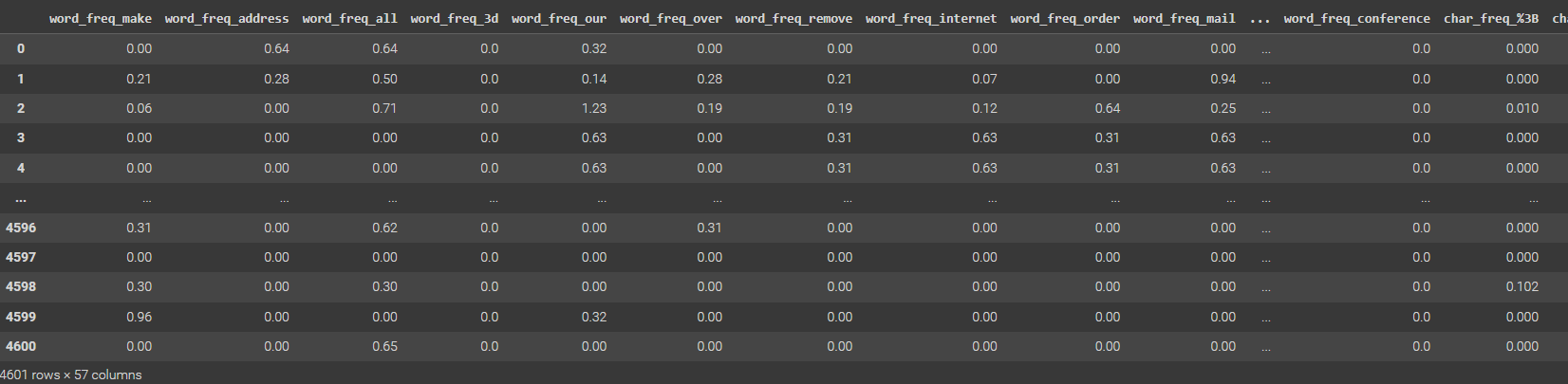


**Building Model – SVM**

**Define Train And Target Columns**

X = df.loc[:,'word\_freq\_make':'capital\_run\_length\_total']

X



y = df['class']

**Train Test Split**

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.3,random\_state=109)

**Perform Feature Scaling – Standardization**

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**Fit And Predict**

kernels = ['linear','poly','rbf','sigmoid']

for ker in kernels:

#Create a svm Classifier

clf = svm.SVC(kernel=ker) # Linear Kernel

#Train the model using the training sets

clf.fit(X\_train, y\_train)

#Predict the response for test dataset

y\_pred = clf.predict(X\_test)

# Model Accuracy: how often is the classifier correct?

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print(f"Kernel: {ker}")

print("Accuracy:", accuracy)

# Model Precision: what percentage of positive tuples are labeled as such?

precision = metrics.precision\_score(y\_test, y\_pred)

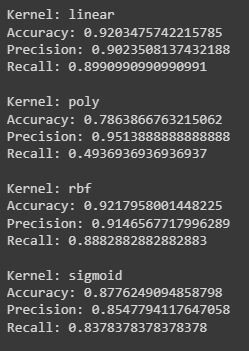
print("Precision:", precision)

# Model Recall: what percentage of positive tuples are labelled as such?

recall = metrics.recall\_score(y\_test, y\_pred)

print("Recall:", recall)

print()



#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets

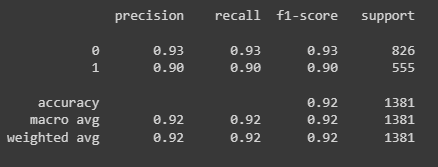
clf.fit(X\_train, y\_train)

#Predict the response for test dataset

y\_pred = clf.predict(X\_test)

**Visualize Output**

**Classification Report**



**Confusion Matrix**

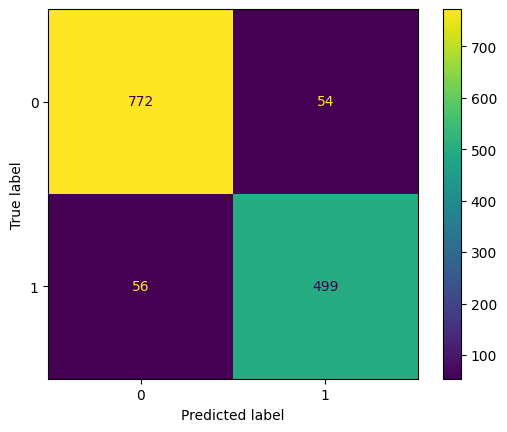
# Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test,y\_pred)

cm\_display = metrics.ConfusionMatrixDisplay(cm)

cm\_display.plot()



**Building Model - Naive Bayes**

**Train Test Split**

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.3,random\_state=109)

**Feature Scaling**

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

**Initialize Model**

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

**Fit And Predict**

y\_pred = gnb.fit(X\_train, y\_train).predict(X\_test)

**Accuracy**

# Model Accuracy: how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

Accuracy: 0.8037653874004345

**Precision & Recall**

# Model Precision: what percentage of positive tuples are labeled as such?

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

# Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:",metrics.recall\_score(y\_test, y\_pred))



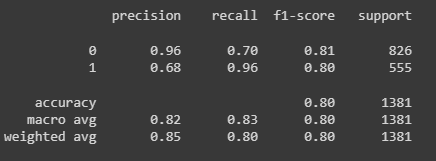
**Visualizing Output**

**Classification Report**

# Classification Report

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))



**Confusion Matrix**

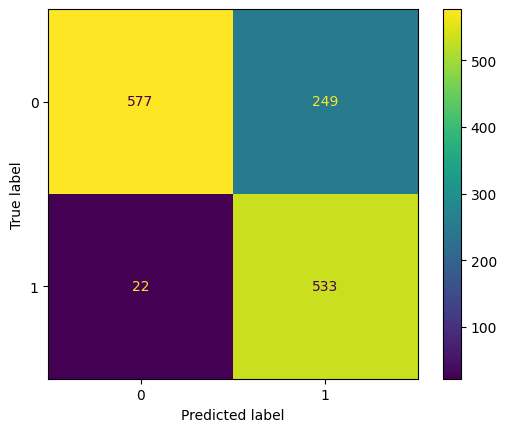
# Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test,y\_pred)

cm\_display = metrics.ConfusionMatrixDisplay(cm)

cm\_display.plot()



**4.2 Classification of MNIST dataset using Support Vector Machine (SVM)**

**Clone Repo**

!git clone https://github.com/Ojus999/Machine-Learning-Sem-6.git

**Import Dependencies**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets, svm, metrics

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

**Loading the Dataset**

def load\_mnist\_images(path):

with open(path, 'rb') as f:

data = np.frombuffer(f.read(), dtype=np.uint8, offset=16)

return data.reshape(-1, 28\*28)

def load\_mnist\_labels(path):

with open(path, 'rb') as f:

data = np.frombuffer(f.read(), dtype=np.uint8, offset=8)

return data

X\_train = load\_mnist\_images('/content/Machine-Learning-Sem-6/Ex 4/mnist/train-images-idx3-ubyte/train-images.idx3-ubyte')

y\_train = load\_mnist\_labels('/content/Machine-Learning-Sem-6/Ex 4/mnist/train-labels-idx1-ubyte/train-labels.idx1-ubyte')

X\_test = load\_mnist\_images('/content/Machine-Learning-Sem-6/Ex 4/mnist/t10k-images-idx3-ubyte/t10k-images.idx3-ubyte')

y\_test = load\_mnist\_labels('/content/Machine-Learning-Sem-6/Ex 4/mnist/t10k-labels-idx1-ubyte/t10k-labels.idx1-ubyte')

**Pre-Processing the Data**

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

**Exploratory Data Analysis:**

# Visualization of some samples from the dataset

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

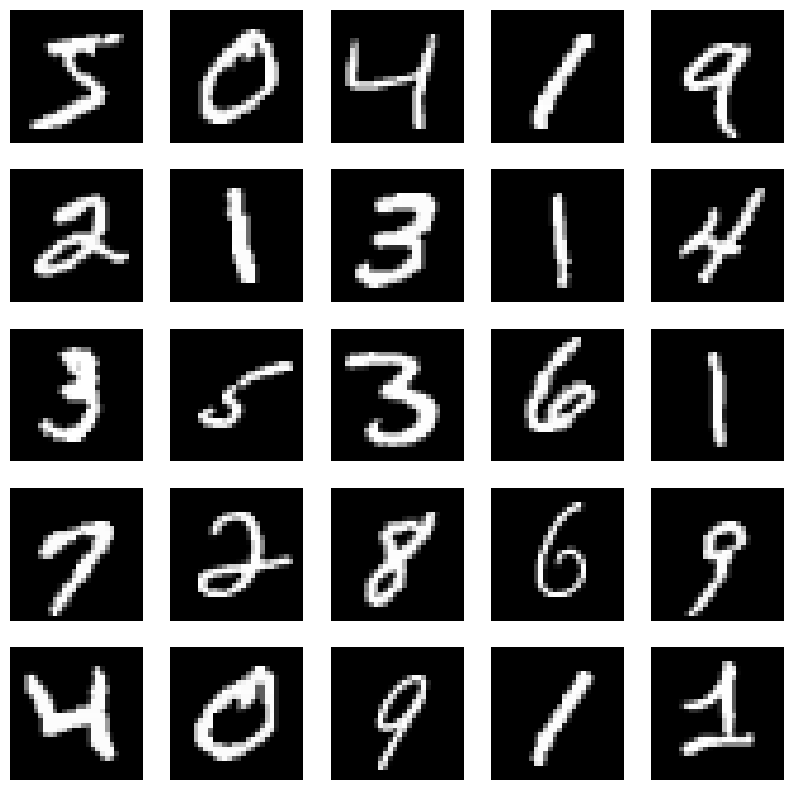
for i in range(25):

plt.subplot(5, 5, i+1)

plt.imshow(X\_train[i].reshape(28, 28), cmap='gray')

plt.axis('off')

plt.show()



**Train Test Split**

#Split the data into training, testing, and validation sets

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.1, random\_state=42)

**Train the Model**

# Train the model

svm\_model = svm.SVC(kernel='rbf', C=10, gamma='scale')

svm\_model.fit(X\_train, y\_train)

**Test the Model**

#Test the model

y\_pred = svm\_model.predict(X\_test)

**Measure Performance**

#Measure the performance of the trained model

accuracy = accuracy\_score(y\_test, y\_pred)

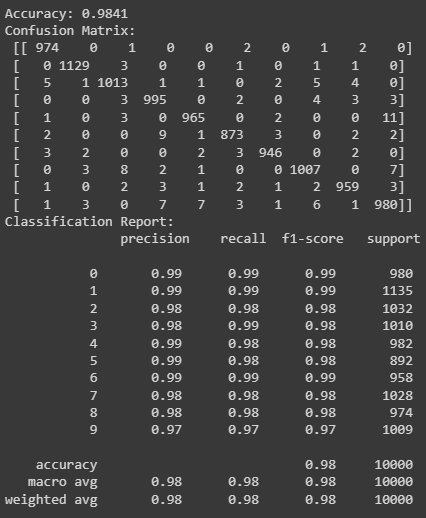
conf\_matrix = confusion\_matrix(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", classification\_rep)



**Visualize Confusion Matrix**

# Visualize confusion matrix

plt.figure(figsize=(8, 6))

plt.imshow(conf\_matrix, cmap='Blues')

plt.colorbar()

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

