\_

Q1] Introduction to various Python libraries for machine learning a. NumPy b. Pandas c. Matplotlib d. Seaborn e. Scikit learn

# Import the quired libraries import numpy as np

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.datasets import load\_iris

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

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

# NumPy: Introduction

print("a. NumPy - Numerical Python") arr = np.array([1, 2, 3, 4, 5]) print("Sample NumPy array:", arr) print("Array Shape:", arr.shape) print("Array Sum:", np.sum(arr)) print("\n")

# Pandas: Introduction

print("b. Pandas - Data Analysis Library")

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 22, 35]} df = pd.DataFrame(data) print("Sample Pandas DataFrame:") print(df)

print("Summary Statistics:") print(df.describe()) print("\n")

# Matplotlib: Introduction

print("c. Matplotlib - Data Visualization Library")

x = np.linspace(0, 2 \* np.pi, 100)

y = np.sin(x) plt.figure() plt.plot(x, y)

plt.title("Sine Wave") plt.xlabel("x") plt.ylabel("sin(x)") plt.show()

print("\n")

# Seaborn: Introduction

print("d. Seaborn - Statistical Data Visualization Library")

iris = sns.load\_dataset("iris") sns.pairplot(iris, hue="species") plt.show()

print("\n")

# Scikit-Learn: Introduction

print("e. Scikit-Learn - Machine Learning Library")

# Load a sample dataset (Iris dataset)

iris = load\_iris()

X, y = iris.data, iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Random Forest Classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

# Make predictions and calculate accuracy y\_pred = clf.predict(X\_test)

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

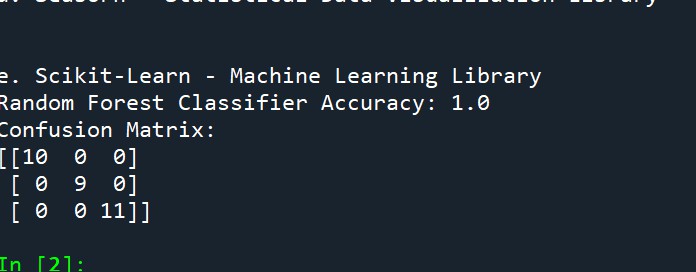
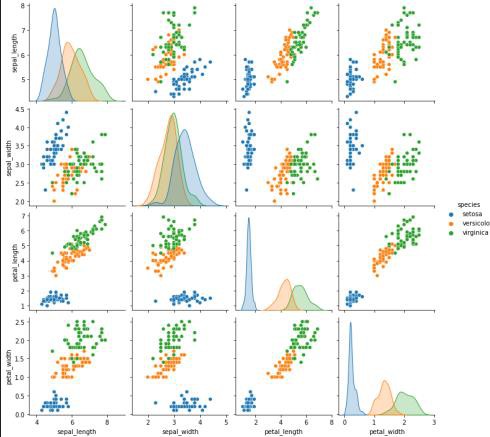
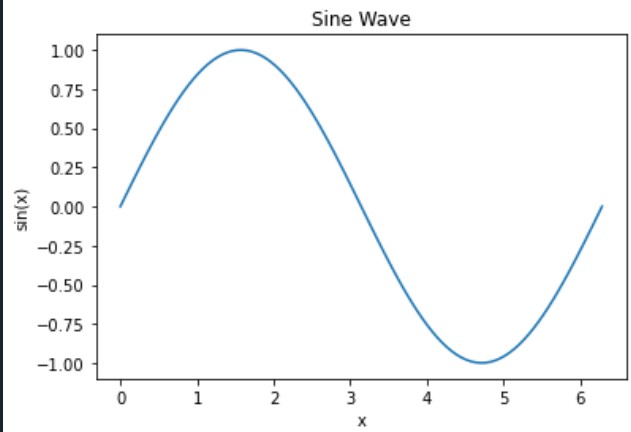
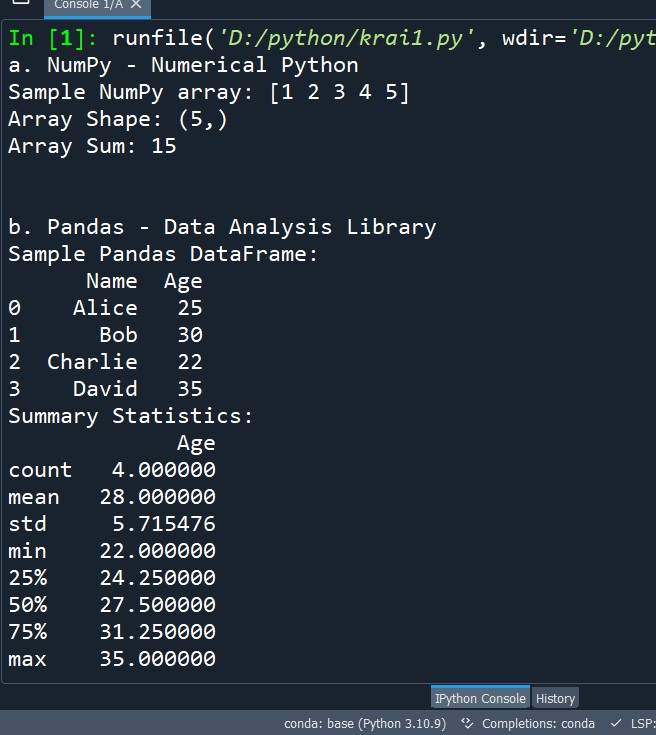
print("Random Forest Classifier Accuracy:", accuracy)

# Generate a confusion matrix

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

print("Confusion Matrix:")

print(conf\_matrix)



4

Q2] Write a program to find the correlation matrix import pandas as pd

data = {

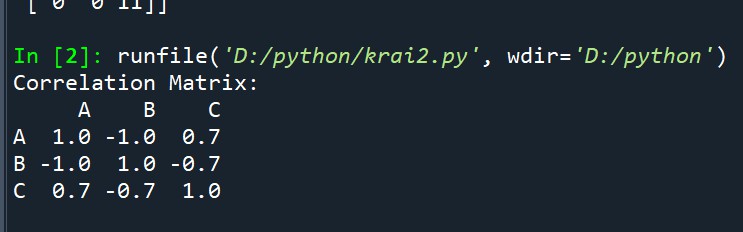
'A': [1, 2, 3, 4, 5],

'B': [5, 4, 3, 2, 1],

'C': [2, 3, 1, 4, 5],

}

df = pd.DataFrame(data) correlation\_matrix = df.corr() print("Correlation Matrix:") print(correlation\_matrix)



Q3] Write a program to Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

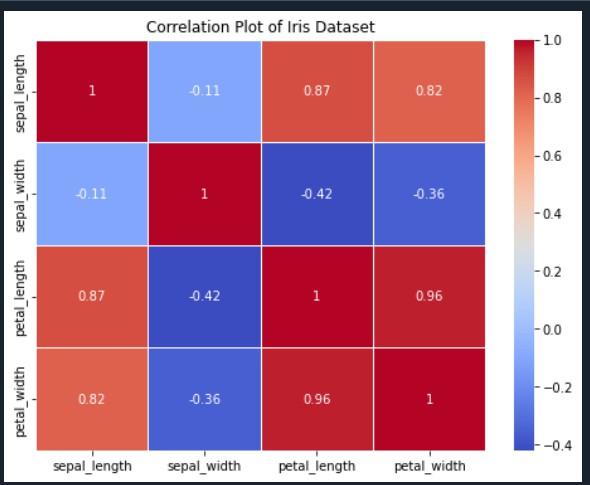
import seaborn as sns

import matplotlib.pyplot as plt iris = sns.load\_dataset("iris") correlation\_matrix = iris.corr() plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", linewidths=0.5)

plt.title("Correlation Plot of Iris Dataset")

plt.show()



Q4] Write a program to implement Analysis of covariance: variance (ANOVA) on IRIS

dataset

import pandas as pd

from scipy.stats import f\_oneway

iris = pd.read\_csv("https://raw.githubusercontent.com/mwaskom/seaborn- data/master/iris.csv")

grouped\_data = [group["sepal\_length"] for name, group in iris.groupby("species")]

f\_statistic, p\_value = f\_oneway(\*grouped\_data)

alpha = 0.05

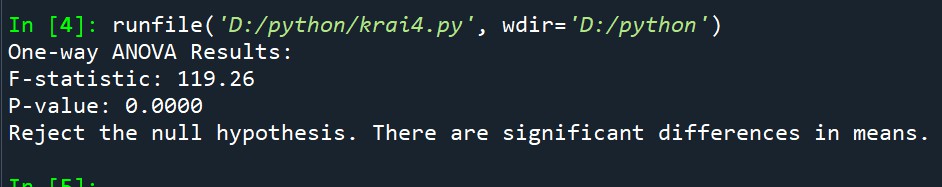
print("One-way ANOVA Results:") print(f"F-statistic: {f\_statistic:.2f}") print(f"P-value: {p\_value:.4f}")

if p\_value < alpha:

print("Reject the null hypothesis. There are significant differences in means.")

else:

print("Fail to reject the null hypothesis. There are no significant differences in means.")



Q5] Write a program to implement linear regression algorithm to create and evaluate a model on a given dataset

import numpy as np import pandas as pd

from sklearn.datasets import fetch\_openml

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score import matplotlib.pyplot as plt

california\_housing = fetch\_openml(name="california\_housing", as\_frame=True)

data = california\_housing.frame target = data["target"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, target, test\_size=0.2, random\_state=42)

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

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error: {:.2f}".format(mse))

print("R-squared (Coefficient of Determination): {:.2f}".format(r2))

plt.scatter(y\_test, y\_pred) plt.xlabel("Actual Values") plt.ylabel("Predicted Values")

plt.title("Actual vs. Predicted Values in Linear Regression")

plt.show()

Q6] Write a program to classify the given dataset using logistic regression and evaluate the model

import numpy as np import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix data = pd.DataFrame({

'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

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

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

print("Accuracy:", accuracy)

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

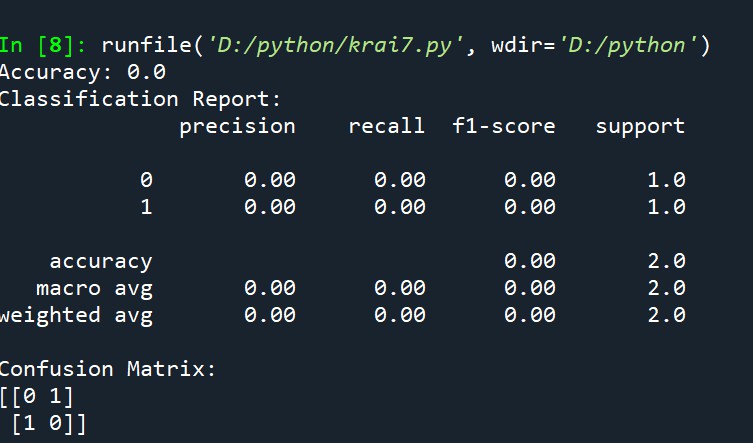
print("Classification Report:")

print(classification\_rep)

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

print("Confusion Matrix:")

print(conf\_matrix)



Q7] Write a program to implement support vector machine algorithm import numpy as np

import pandas as pd

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix data = pd.DataFrame({

'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = SVC(kernel='linear', C=1.0)

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

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

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

print("Accuracy:", accuracy)

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

print("Classification Report:")

print(classification\_rep)

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

print("Confusion Matrix:")

print(conf\_matrix)

Q8] Write a program to implement Decision Tree model on the given dataset import numpy as np

import pandas as pd

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix data = pd.DataFrame({

'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = DecisionTreeClassifier()

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

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

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

print("Accuracy:", accuracy)

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

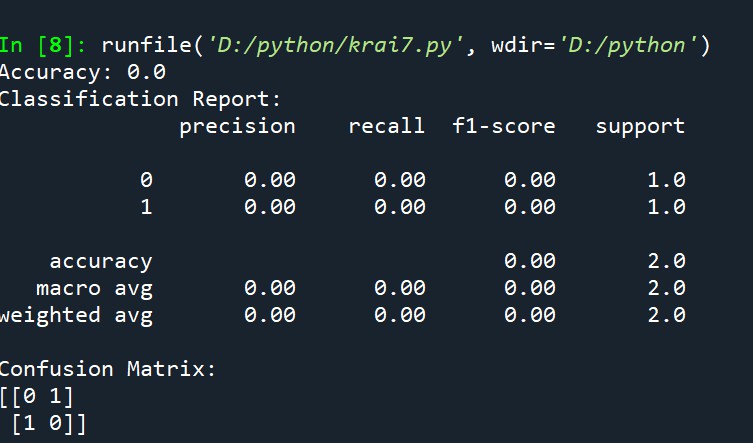
print("Classification Report:")

print(classification\_rep)

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

print("Confusion Matrix:")

print(conf\_matrix)



Q9] Write a program to implement Bayesian classification on given dataset. import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix data = pd.DataFrame({

'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = GaussianNB()

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

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

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

print("Accuracy:", accuracy)

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

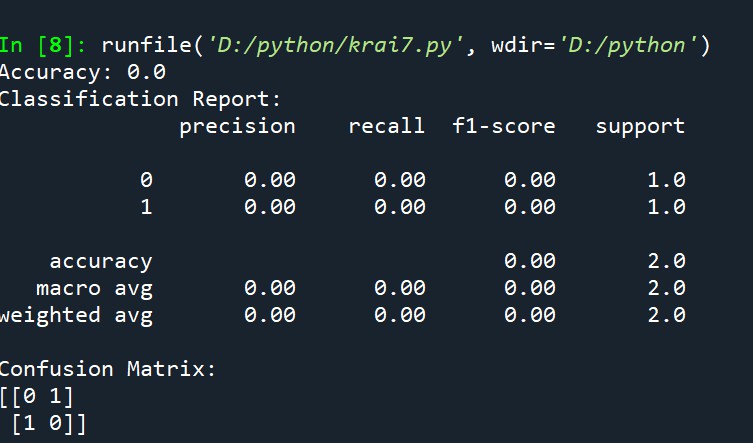
print("Classification Report:")

print(classification\_rep)

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

print("Confusion Matrix:")

print(conf\_matrix)



Q10] Write a program to implement K-Nearest Neighbor algorithm on given dataset. import numpy as np

import pandas as pd

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix data = pd.DataFrame({

'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = KNeighborsClassifier(n\_neighbors=3)

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

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

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

print("Accuracy:", accuracy)

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

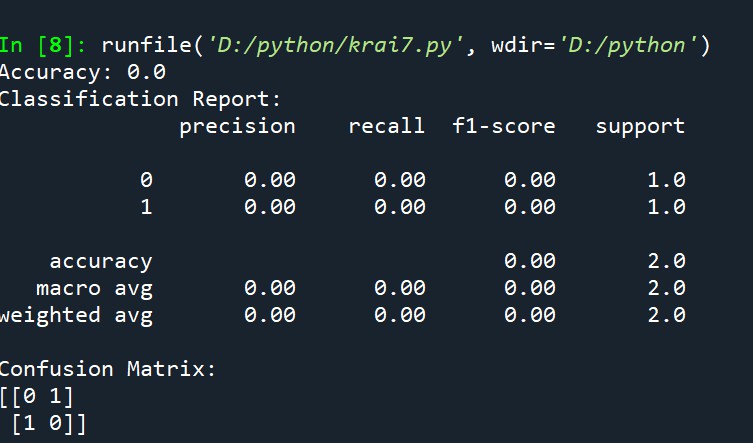
print("Classification Report:")

print(classification\_rep)

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

print("Confusion Matrix:")

print(conf\_matrix)



Q11] Write a program to implement K-Means algorithm on given dataset and visualize the clusters.

import pandas as pd import numpy as np

from sklearn.cluster import KMeans import matplotlib.pyplot as plt

data = pd.DataFrame({

'X1': [1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6],

'X2': [1, 2, 3, 4, 3.5, 4.5, 5.5, 6, 5, 6]

})

K = 2

model = KMeans(n\_clusters=K, random\_state=42)

model.fit(data)

cluster\_centers = model.cluster\_centers\_ labels = model.labels\_ plt.figure(figsize=(8, 6))

for k in range(K):

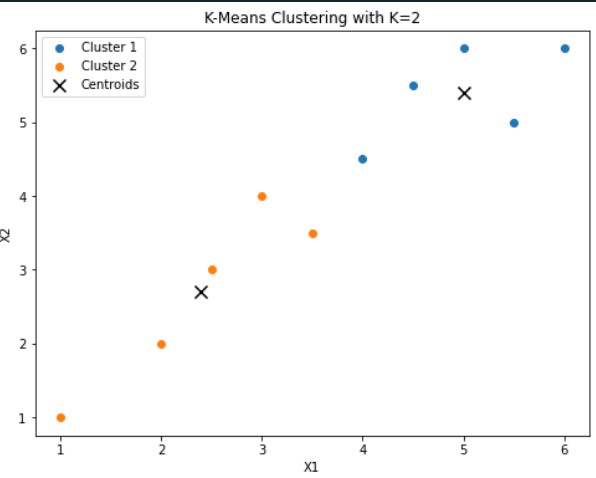
cluster\_data = data[labels == k]

plt.scatter(cluster\_data['X1'], cluster\_data['X2'], label=f'Cluster {k + 1}')

plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], c='black', marker='x', s=100, label='Centroids')

plt.title(f'K-Means Clustering with K={K}')

plt.xlabel('X1') plt.ylabel('X2') plt.legend() plt.show()



Q12] Write a program to implement deep learning algorithm using ANN

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dense, Flatten from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0 model = keras.models.Sequential([

Flatten(input\_shape=(28, 28)), Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5)

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=2)

print(f"Test accuracy: {test\_accuracy}") predictions = model.predict(x\_test) plt.figure(figsize=(10, 10))

for i in range(25): plt.subplot(5, 5, i + 1) plt.xticks([]) plt.yticks([]) plt.grid(False)

plt.imshow(x\_test[i], cmap=plt.cm.binary) predicted\_label = tf.argmax(predictions[i]) true\_label = y\_test[i]

if predicted\_label == true\_label:

color = 'green' else:

color = 'red'

plt.xlabel(f"{predicted\_label} ({true\_label})", color=color)

plt.show()

Q13] Write a program to implement deep learning algorithm using CNN

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0 model = keras.Sequential([

layers.Input(shape=(28, 28, 1)), layers.Conv2D(32, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)), layers.Conv2D(64, (3, 3), activation='relu'), layers.MaxPooling2D((2, 2)), layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1)

model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test)) test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=2) print(f"Test accuracy: {test\_accuracy}")

predictions = model.predict(x\_test)

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

for i in range(25): plt.subplot(5, 5, i + 1) plt.xticks([]) plt.yticks([]) plt.grid(False)

plt.imshow(x\_test[i].reshape(28, 28), cmap=plt.cm.binary)

predicted\_label = tf.argmax(predictions[i])

true\_label = y\_test[i]

if predicted\_label == true\_label:

color = 'green' else:

color = 'red'

plt.xlabel(f"{predicted\_label} ({true\_label})", color=color)

plt.show()

Q14] Write a program to implement deep learning algorithm using GAN

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers def build\_generator(latent\_dim):

model = keras.Sequential()

model.add(layers.Dense(7 \* 7 \* 128, input\_dim=latent\_dim))

model.add(layers.Reshape((7, 7, 128)))

model.add(layers.Conv2DTranspose(128, kernel\_size=4, strides=2, padding="same"))

model.add(layers.BatchNormalization(momentum=0.8))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Conv2DTranspose(64, kernel\_size=4, strides=2, padding="same")) model.add(layers.BatchNormalization(momentum=0.8)) model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Conv2D(1, kernel\_size=4, padding="same", activation="tanh"))

return model

def build\_discriminator(img\_shape):

model = keras.Sequential()

model.add(layers.Conv2D(32, kernel\_size=4, strides=2, input\_shape=img\_shape, padding="same"))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Conv2D(64, kernel\_size=4, strides=2, padding="same")) model.add(layers.BatchNormalization(momentum=0.8)) model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Conv2D(128, kernel\_size=4, strides=2, padding="same")) model.add(layers.BatchNormalization(momentum=0.8)) model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Flatten()) model.add(layers.Dense(1, activation="sigmoid")) return model

discriminator = build\_discriminator((28, 28, 1))

discriminator.compile(loss="binary\_crossentropy", optimizer=keras.optimizers.Adam(0.0002, 0.5), metrics=["accuracy"])

latent\_dim = 100

generator = build\_generator(latent\_dim)

discriminator.trainable = False

gan\_input = keras.Input(shape=(latent\_dim,))

x = generator(gan\_input)

gan\_output = discriminator(x)

gan = keras.models.Model(gan\_input, gan\_output)

gan.compile(loss="binary\_crossentropy", optimizer=keras.optimizers.Adam(0.0002, 0.5))

def train\_gan(generator, discriminator, gan, batch\_size, latent\_dim, num\_epochs):

for epoch in range(num\_epochs):

for \_ in range(int(60000 / batch\_size)):

noise = tf.random.normal((batch\_size, latent\_dim))

generated\_images = generator.predict(noise)

real\_images = x\_train[np.random.randint(0, x\_train.shape[0], batch\_size)]

labels\_real = np.ones((batch\_size, 1))

labels\_fake = np.zeros((batch\_size, 1))

d\_loss\_real = discriminator.train\_on\_batch(real\_images, labels\_real) d\_loss\_fake = discriminator.train\_on\_batch(generated\_images, labels\_fake) d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

noise = tf.random.normal((batch\_size, latent\_dim))

labels\_gan = np.ones((batch\_size, 1))

g\_loss = gan.train\_on\_batch(noise, labels\_gan)

print(f"Epoch {epoch}/{num\_epochs}, D Loss: {d\_loss[0]}, G Loss: {g\_loss}")

plot\_generated\_images(generator, epoch, latent\_dim)

def plot\_generated\_images(generator, epoch, latent\_dim, examples=10, dim=(1, 10), figsize=(10, 1)):

noise = tf.random.normal((examples, latent\_dim)) generated\_images = generator.predict(noise) generated\_images = 0.5 \* generated\_images + 0.5 plt.figure(figsize=figsize)

for i in range(examples): plt.subplot(dim[0], dim[1], i + 1) plt.imshow(generated\_images[i, :, :, 0], cmap="gray") plt.axis("off")

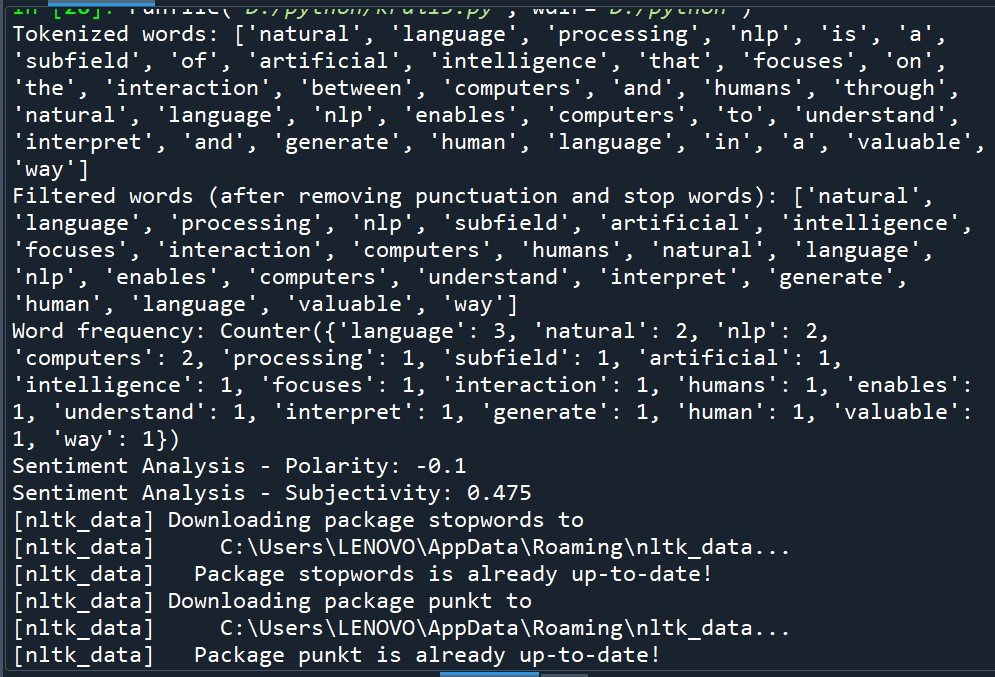
plt.tight\_layout()

plt.savefig(f"gan\_generated\_image\_epoch\_{epoch}.png") (x\_train, \_), (\_, \_) = keras.datasets.mnist.load\_data()

x\_train = x\_train / 127.5 - 1.0

x\_train = np.expand\_dims(x\_train, axis=3)

train\_gan(generator, discriminator, gan, batch\_size=64, latent\_dim=latent\_dim, num\_epochs=100)



Q16] Write a program to implement web scrapping on the given URL

import requests

from bs4 import BeautifulSoup url = "https://example.com" response = requests.get(url)

if response.status\_code == 200:

soup = BeautifulSoup(response.text, "html.parser")

links = soup.find\_all("a")

for link in links:

print(link.get("href"))

else:

print("Failed to retrieve the webpage. Status code:", response.status\_code)

