[***https://justpaste.it/d6a3n***](https://justpaste.it/d6a3n)

***DL1: Design RNN or its variant including LSTM or GRU***

# Importing necessary libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

# Enable inline plotting for Jupyter Notebook

%matplotlib inline

# Load the dataset

HouseDF = pd.read\_csv('USA\_Housing.csv')

# Display the first few rows

print(HouseDF.head())

# Basic information about the dataset

print(HouseDF.info())

# Statistical summary

print(HouseDF.describe())

# Display column names

print(HouseDF.columns)

# Pairplot to visualize relationships

sns.pairplot(HouseDF)

plt.show()

# Distribution plot of the target variable 'Price'

sns.histplot(HouseDF['Price'], kde=True)

plt.title('Distribution of House Prices')

plt.show()

# Feature matrix and target variable

X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',

'Avg. Area Number of Bedrooms', 'Area Population']]

y = HouseDF['Price']

# Splitting the data into training and testing sets

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=101)

# Creating and training the linear regression model

from sklearn.linear\_model import LinearRegression

lm = LinearRegression()

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

# Printing the intercept

print("Intercept:", lm.intercept\_)

# Displaying coefficients for each feature

coeff\_df = pd.DataFrame(lm.coef\_, X.columns, columns=['Coefficient'])

print(coeff\_df)

# Predicting house prices on test data

predictions = lm.predict(X\_test)

# Scatter plot of actual vs predicted values

plt.scatter(y\_test, predictions)

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.show()

# Plotting the distribution of residuals (errors)

sns.histplot(y\_test - predictions, bins=50, kde=True)

plt.title('Distribution of Residuals')

plt.show()

# Evaluating the model

from sklearn import metrics

print('MAE:', metrics.mean\_absolute\_error(y\_test, predictions))

print('MSE:', metrics.mean\_squared\_error(y\_test, predictions))

print('RMSE:', np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))

***DL2: Build a Multiclass classifier using the CNN model. Use MNIST or any other suitable dataset***

# Import necessary libraries

from tensorflow.keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

# Load MNIST dataset

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

# Display an MNIST image

print("Training data example:")

plt.imshow(x\_train[4], cmap="gray") # Ensuring grayscale display

plt.colorbar() # Adds intensity scale

plt.show()

print("Label of this image is:", y\_train[4])  
# Import necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# ✅ Load the dataset

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

# ✅ Display an image from the dataset

plt.imshow(x\_train[4], cmap="gray")

plt.title(f"Label: {y\_train[4]}")

plt.show()

# ✅ Reshape the dataset for CNN input

x\_train = x\_train.reshape(-1, 28, 28, 1)

x\_test = x\_test.reshape(-1, 28, 28, 1)

# ✅ Normalize the pixel values (scale between 0 and 1)

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# ✅ Convert labels to categorical (One-Hot Encoding)

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# ✅ Create a CNN model

model = Sequential()

model.add(Conv2D(32, kernel\_size=5, strides=1, padding="same", activation="relu", input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2), padding="same"))

model.add(Conv2D(64, kernel\_size=5, strides=1, padding="same", activation="relu"))

model.add(MaxPooling2D(pool\_size=(2, 2), padding="same"))

model.add(Flatten())

model.add(Dense(1024, activation="relu"))

model.add(Dropout(0.2))

model.add(Dense(10, activation="softmax")) # Use "softmax" for multi-class classification

# ✅ Compile the model

model.compile(optimizer="adam", loss="categorical\_crossentropy", metrics=["accuracy"])

# ✅ Train the model

model.fit(x\_train, y\_train, batch\_size=100, epochs=5, validation\_data=(x\_test, y\_test))

pip install tensorflow scikit-learn matplotlib

import tensorflow as tf  
from tensorflow.keras import layers, models  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
import matplotlib.pyplot as plt  
import numpy as np

# a. Data Preprocessing

mnist = tf.keras.datasets.mnist

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

# Normalize images to the range [0, 1]

x\_train = x\_train.astype("float32") / 255.0

x\_test = x\_test.astype("float32") / 255.0

# Reshape to include channel dimension (28x28x1)

x\_train = x\_train[..., np.newaxis]

x\_test = x\_test[..., np.newaxis]

# b. Define Model  
model = models.Sequential([  
    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),  
    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')  # 10 classes for digits 0–9  
])

# Compile model

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# Train model

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

# c. Evaluate using confusion matrix  
y\_pred = model.predict(x\_test)  
y\_pred\_classes = np.argmax(y\_pred, axis=1)

cm = confusion\_matrix(y\_test, y\_pred\_classes)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.arange(10))  
disp.plot(cmap=plt.cm.Blues)  
plt.title("Confusion Matrix - MNIST CNN")  
plt.show()

***DL3: Design RNN or its variant including LSTM or GRU***

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.datasets import imdb

# Load IMDB dataset (only top 10,000 words to keep it simple)

vocab\_size = 10000

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=vocab\_size)

# Pad sequences to the same length

maxlen = 200

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

model = Sequential([

Embedding(vocab\_size, 128, input\_length=maxlen),

LSTM(64, dropout=0.2, recurrent\_dropout=0.2),

Dense(1, activation='sigmoid') # Binary sentiment (positive or negative)

])

# Compile model

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

# Train model

model.fit(x\_train, y\_train, batch\_size=64, epochs=3, validation\_data=(x\_test, y\_test))

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

import numpy as np

# Predict on test data

y\_pred\_probs = model.predict(x\_test)

y\_pred = (y\_pred\_probs > 0.5).astype("int32")

# Confusion matrix

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

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=["Negative", "Positive"])

disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix - Sentiment LSTM Model")

plt.show()

word\_index = imdb.get\_word\_index()

reverse\_word\_index = {value: key for (key, value) in word\_index.items()}

def decode\_review(encoded\_review):

return ' '.join([reverse\_word\_index.get(i - 3, '?') for i in encoded\_review])

# Example review

sample\_index = 123

print("Review Text:\n", decode\_review(x\_test[sample\_index]))

print("Predicted Sentiment:", "Positive" if y\_pred[sample\_index] else "Negative")

print("Actual Sentiment:", "Positive" if y\_test[sample\_index] else "Negative")

***DL4: Design and implement CNN for image classification.***

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.mnist.load\_data()

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

# Add channel dimension to the images

train\_images = train\_images.reshape((60000, 28, 28, 1))

test\_images = test\_images.reshape((10000, 28, 28, 1))

# Split the dataset into training and validation sets

train\_images, val\_images, train\_labels, val\_labels = train\_test\_split( train\_images, train\_labels, test\_size=0.1, random\_state=42 )

# Data augmentation for training images

datagen = ImageDataGenerator(rotation\_range=10, zoom\_range=0.1, width\_shift\_range=0.1, height\_shift\_range=0.1)

datagen.fit(train\_images)

# Create a CNN model with hyperparameter tuning and regularization

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.Flatten())

model.add(layers.Dropout(0.5))

model.add(layers.Dense(128, activation='relu'))

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

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model with data augmentation

history = model.fit(datagen.flow(train\_images, train\_labels, batch\_size=64),epochs=20, validation\_data=(val\_images, val\_labels))

# Evaluate the model on the test set

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f"Test Accuracy: {test\_acc}")

# Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

***DL5: Design and implement Deep Convolutional GAN***

from keras.models import Model

from keras.layers import Input,Dense

import numpy as np

import pandas as pd

import keras.backend as K

import matplotlib.pyplot as plt

from keras import preprocessing

from keras.models import Sequential #

from keras.layers import Conv2D,Dropout,Dense,Flatten,Conv2DTranspose,BatchNormalization,LeakyReLU,Reshape

import tensorflow as tf

from keras.layers import \*

from keras.datasets import fashion\_mnist

(train\_x, train\_y), (val\_x, val\_y) = fashion\_mnist.load\_data()

train\_x = train\_x/255.

val\_x = val\_x/255.

train\_x=train\_x.reshape(-1,28,28,1)

print(train\_x.shape)

#train\_x = train\_x.reshape(-1, 784)

#val\_x = val\_x.reshape(-1, 784)

fig,axe=plt.subplots(2,2)

idx = 0

for i in range(2):

for j in range(2):

axe[i,j].imshow(train\_x[idx].reshape(28,28),cmap='gray')

idx+=1

train\_x = train\_x\*2 - 1

print(train\_x.max(),train\_x.min())

generator = Sequential()

generator.add(Dense(512,input\_shape=[100]))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(256))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(128))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(784))

generator.add(Reshape([28,28,1]))

generator.summary()

discriminator = Sequential()

discriminator.add(Dense(1,input\_shape=[28,28,1]))

discriminator.add(Flatten())

discriminator.add(Dense(256))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(128))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(64))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(1,activation='sigmoid'))

discriminator.summary()

GAN =Sequential([generator,discriminator])

discriminator.compile(optimizer='adam',loss='binary\_crossentropy')

discriminator.trainable = False

GAN.compile(optimizer='adam',loss='binary\_crossentropy')

GAN.summary()

epochs = 30

batch\_size = 100

noise\_shape=100

with tf.device('/gpu:0'):

for epoch in range(epochs):

print(f"Currently on Epoch {epoch+1}")

for i in range(train\_x.shape[0]//batch\_size):

if (i+1)%100 == 0:

print(f"\tCurrently on batch number {i+1} of {train\_x.shape[0]//batch\_size}")

noise=np.random.normal(size=[batch\_size,noise\_shape])

gen\_image = generator.predict\_on\_batch(noise)

train\_dataset = train\_x[i\*batch\_size:(i+1)\*batch\_size]

#training discriminator on real images

train\_label=np.ones(shape=(batch\_size,1))

#train\_label=np.ones((batch\_size, 1))

discriminator.trainable = True

#train\_dataset=train\_x[idx]

d\_loss\_real=discriminator.train\_on\_batch(train\_dataset,train\_label) #training discriminator on fake images

train\_label=np.zeros(shape=(batch\_size,1))

d\_loss\_fake=discriminator.train\_on\_batch(gen\_image,train\_label)

#training generator

noise=np.random.normal(size=[batch\_size,noise\_shape])

train\_label=np.ones(shape=(batch\_size,1))

discriminator.trainable = False

d\_g\_loss\_batch =GAN.train\_on\_batch(noise, train\_label)

#plotting generated images at the start and then after every 10 epoch if epoch % 10 == 0:

samples = 10

x\_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100)))

for k in range(samples):

plt.subplot(2, 5, k+1)

plt.imshow(x\_fake[k].reshape(28, 28), cmap='gray')

plt.xticks([])

plt.yticks([])

plt.tight\_layout()

plt.show()

print('Training is complete')

noise=np.random.normal(size=[10,noise\_shape])

gen\_image = generator.predict(noise)

plt.imshow(noise)

plt.title('How the noise looks')

fig,axe=plt.subplots(2,5)

fig.suptitle('Generated Images from Noise using GANs')

idx=0

for i in range(2):

for j in range(5):

axe[i,j].imshow(gen\_image[idx].reshape(28,28),cmap='gray')

idx+=1

***DL6: Perform sentiment analysis with a recurrent neural networks RNN***

pip install tensorflow matplotlib networkx nltk

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt

import nltk

import networkx as nx

import random

nltk.download('punkt')

# Load IMDb dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.imdb.load\_data(num\_words=10000)

# Pad sequences

maxlen = 200

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

model = Sequential([

Embedding(input\_dim=10000, output\_dim=128, input\_length=maxlen),

LSTM(64, dropout=0.2, recurrent\_dropout=0.2),

Dense(1, activation='sigmoid')

])

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

model.summary()

# Train the model

model.fit(x\_train, y\_train, epochs=2, batch\_size=64, validation\_data=(x\_test, y\_test))

# Generate predictions for a sample set

sample\_reviews = x\_test[:50]

sample\_preds = model.predict(sample\_reviews)

sample\_preds = sample\_preds.flatten()

# Create graph

G = nx.Graph()

# Add nodes with sentiment

for i in range(len(sample\_reviews)):

sentiment = 'Positive' if sample\_preds[i] > 0.5 else 'Negative'

G.add\_node(i, sentiment=sentiment)

# Add random edges (simulate topic connection or user interaction)

for i in range(len(sample\_reviews)):

for j in range(i+1, len(sample\_reviews)):

if random.random() < 0.05: # sparsity

G.add\_edge(i, j)

# Draw graph

colors = ['green' if G.nodes[n]['sentiment'] == 'Positive' else 'red' for n in G.nodes]

plt.figure(figsize=(10, 8))

nx.draw(G, with\_labels=True, node\_color=colors, node\_size=500, font\_size=8)

plt.title("Sentiment Flow in Network Graph")

plt.show()

***BI7: Import Data from different Sources such as (Excel, Sql Server, Oracle etc.) and load in targeted system.***

1) Launch Power BI Desktop.

2) From the Home ribbon, select Get Data. Excel is one of the Most Common data connections, so you can select it directly from the Get Data menu.

3) If you select the Get Data button directly, you can also select FIle > Excel and select Connect.

4) In the Open File dialog box, select the Products.xlsx file.

5) In the Navigator pane, select the Products table and then select Edit

Importing Data from OData Feed In this task, you'll bring in order data. This step represents connecting to a sales system. You import data into Power BI Desktop from the sample Northwind OData feed at the following URL, which you can copy (and then paste) in the steps below: http://services.odata.org/V3/Northwind/Northwind.svc/ Connect to an OData feed:

1) From the Home ribbon tab in Query Editor, select Get Data.

2) Browse to the OData Feed data source.

3) In the OData Feed dialog box, paste the URL for the Northwind OData feed.

4) Select OK.

5) In the Navigator pane, select the Orders table, and then select Edit.

http://services.odata.org/V3/Northwind/Northwind.svc/

RIGHT CLICK, EDIT QUERY

RIGHT CLICK, REPLACE….

***BI8: Data Visualization from Extraction Transformation and Loading (ETL) Process***

import pandas as pd

# Extract data from a CSV (you can replace with your dataset)

url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv"

df = pd.read\_csv(url)

# Preview the data

print(df.head())

# Check for null values

print(df.isnull().sum())

# Feature Engineering: Add Tip % Column

df['tip\_pct'] = df['tip'] / df['total\_bill']

# Convert categorical to category type

df['sex'] = df['sex'].astype('category')

df['smoker'] = df['smoker'].astype('category')

df['day'] = df['day'].astype('category')

df['time'] = df['time'].astype('category')

# Summary

print(df.describe())

# Load into cleaned DataFrame (could be into a DB too)

cleaned\_df = df.copy()

# Save to new CSV if needed

cleaned\_df.to\_csv("cleaned\_tips\_data.csv", index=False)

import seaborn as sns

import matplotlib.pyplot as plt

# Set theme

sns.set(style="whitegrid")

# 1. Tip percentage vs. total bill

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

sns.scatterplot(data=cleaned\_df, x="total\_bill", y="tip\_pct", hue="sex")

plt.title("Tip % vs Total Bill")

plt.xlabel("Total Bill ($)")

plt.ylabel("Tip %")

plt.show()

# 2. Distribution of Tip %

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

sns.histplot(cleaned\_df["tip\_pct"], kde=True, color="purple")

plt.title("Distribution of Tip Percentage")

plt.show()

# 3. Average Tip % by Day

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

sns.barplot(data=cleaned\_df, x="day", y="tip\_pct", hue="sex")

plt.title("Average Tip % by Day and Gender")

plt.ylabel("Tip %")

plt.show()

# 4. Heatmap of correlation

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

sns.heatmap(cleaned\_df.corr(numeric\_only=True), annot=True, cmap="YlGnBu")

plt.title("Feature Correlation Heatmap")

plt.show()

***BI11: Perform the data classification algorithm using any Classification algorithm***

import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

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

import seaborn as sns

import matplotlib.pyplot as plt

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

labels = iris.target\_names

# Create DataFrame

df = pd.DataFrame(X, columns=iris.feature\_names)

df['species'] = y

print(df.head())

# Train-test split

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

# Standardize the features

scaler = StandardScaler()

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

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

# Use Logistic Regression

clf = LogisticRegression(max\_iter=200)

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

# Predict

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

# Accuracy

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

# Confusion Matrix

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

sns.heatmap(cm, annot=True, cmap="Blues", xticklabels=labels, yticklabels=labels)

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

# Classification Report

print("Classification Report:\n", classification\_report(y\_test, y\_pred, target\_names=labels))

**NEW CODE:-**  
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

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

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

# Step 1: Define rainfall data for 12 months

rainfall = [799, 1174.8, 865.1, 1334.6, 635.4, 918.5, 685.5, 998.6, 784.2, 985, 882.8, 1071]

# Create a time series with datetime index

dates = pd.date\_range(start='2012-01', periods=12, freq='M')

rainfall\_series = pd.Series(rainfall, index=dates)

# Step 2: Plot the time series and save it

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

plt.plot(rainfall\_series, marker='o')

plt.title('Monthly Rainfall - 2012')

plt.xlabel('Month')

plt.ylabel('Rainfall (mm)')

plt.grid(True)

plt.savefig("rainfall.png")

plt.show()

# Step 3: Classification - Label rainfall as High (1) or Low (0) based on median

threshold = np.median(rainfall) # Can also use mean

labels = [1 if val >= threshold else 0 for val in rainfall] # 1 = High, 0 = Low

# Feature could be month (numerical), and rainfall

df = pd.DataFrame({

'Month': np.arange(1, 13),

'Rainfall': rainfall,

'Label': labels

})

# Step 4: Train a classification model

X = df[['Month', 'Rainfall']] # Features

y = df['Label'] # Target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = DecisionTreeClassifier(random\_state=42)

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

# Step 5: Evaluate the model

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

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

***BI12: Perform the data clustering algorithm using any Clustering algorithm***

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Load Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

labels = iris.target\_names

# Create a DataFrame

df = pd.DataFrame(X, columns=iris.feature\_names)

print(df.head())

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply KMeans clustering (we know there are 3 clusters in Iris dataset)

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(X\_scaled)

# Add cluster labels to DataFrame

df['Cluster'] = kmeans.labels\_

# Reduce dimensions to 2D using PCA for visualization

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Plot the clusters

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

sns.scatterplot(x=X\_pca[:, 0], y=X\_pca[:, 1], hue=df['Cluster'], palette="Set1", s=100)

plt.title("K-Means Clustering on Iris (PCA Reduced)")

plt.xlabel("PCA 1")

plt.ylabel("PCA 2")

plt.show()