**Experiment 1**

**Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.**

**Import Necessary Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv(r"C:\Users\Asus\Documents\Datasets[1]\Datasets\housing.csv")

df.head()

df.shape

df.info()

df.nunique()

**Data Cleaning**

df.isnull().sum()

df.duplicated().sum()

df['total\_bedrooms'].median()

# Handling missing values

df['total\_bedrooms'].fillna(df['total\_bedrooms'].median(), inplace=True)

**Feature Engineering**

for i in df.iloc[:,2:7]:

df[i] = df[i].astype('int')

**Descriptive Statistics**

df.describe().T

Numerical = df.select\_dtypes(include=[np.number]).columns

print(Numerical)

**Uni-Variate Analysis**

for col in Numerical:

plt.figure(figsize=(10, 6))

df[col].plot(kind='hist', title=col, bins=60, edgecolor='black')

plt.ylabel('Frequency')

plt.show()

for col in Numerical:

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

sns.boxplot(df[col], color='blue')

plt.title(col)

plt.ylabel(col)

plt.show()

**Experiment 2**

**Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import fetch\_california\_housing

**# Load California Housing dataset**

data = fetch\_california\_housing()

**# Convert to DataFrame**

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

df['Target'] = data.target # Adding the target variable (median house value)

# Table of Meaning of Each Variable

variable\_meaning = {

"MedInc": "Median income in block group",

"HouseAge": "Median house age in block group",

"AveRooms": "Average number of rooms per household",

"AveBedrms": "Average number of bedrooms per household",

"Population": "Population of block group",

"AveOccup": "Average number of household members",

"Latitude": "Latitude of block group",

"Longitude": "Longitude of block group",

"Target": "Median house value (in $100,000s)"

}

variable\_df = pd.DataFrame(list(variable\_meaning.items()), columns=["Feature", "Des

print("\nVariable Meaning Table:")

print(variable\_df)

# Basic Data Exploration

print("\nBasic Information about Dataset:")

print(df.info()) # Overview of dataset

print("\nFirst Five Rows of Dataset:")

print(df.head()) # Display first few rows

# Check for missing values

print("\nMissing Values in Each Column:")

print(df.isnull().sum()) # Count of missing values

# Histograms for distribution of features

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

df.hist(figsize=(12, 8), bins=30, edgecolor='black')

plt.suptitle("Feature Distributions", fontsize=16)

plt.show()

# Boxplots for outlier detection

plt.figure(figsize=(12, 6))

sns.boxplot(data=df)

plt.xticks(rotation=45)

plt.title("Boxplots of Features to Identify Outliers")

plt.show()

# Correlation Matrix

plt.figure(figsize=(10, 6))

corr\_matrix = df.corr()

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title("Feature Correlation Heatmap")

plt.show()

# Pairplot to analyze feature relationships (only a subset for clarity)

sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'Target']], diag\_kind='kde')

plt.show()

# Insights from Data Exploration

print("\nKey Insights:")

print("1. The dataset has", df.shape[0], "rows and", df.shape[1], "columns.")

print("2. No missing values were found in the dataset.")

print("3. Histograms show skewed distributions in some features like 'MedInc'.")

print("4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.")

print("5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.")

**Experiment 3 Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.**

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Load the Iris dataset

iris = load\_iris()

features = iris.data # The 4 features: Sepal Length, Sepal Width, Petal Length, Petal Width

target = iris.target # The target class (species)

# Step 2: Standardize the features

scaler = StandardScaler()

features\_standardized = scaler.fit\_transform(features)

# Step 3: Apply PCA to reduce to 2 components

pca = PCA(n\_components=2)

features\_pca = pca.fit\_transform(features\_standardized)

# Step 4: Create a DataFrame for the reduced data

pca\_df = pd.DataFrame(data=features\_pca, columns=["Principal Component 1", "Principal Component 2"])

pca\_df["Target"] = target

# Step 5: Visualize the results

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

for label, color in zip(iris.target\_names, ["red", "green", "blue"]):

plt.scatter(

pca\_df.loc[pca\_df["Target"] == list(iris.target\_names).index(label), "Principal Component 1"],

pca\_df.loc[pca\_df["Target"] == list(iris.target\_names).index(label), "Principal Component 2"],

label=label,

alpha=0.7

)

plt.title("PCA on Iris Dataset (4 features to 2 features)", fontsize=14)

plt.xlabel("Principal Component 1", fontsize=12)

plt.ylabel("Principal Component 2", fontsize=12)

plt.legend(title="Species")

plt.grid()

plt.show()

explained\_variance = pca.explained\_variance\_ratio\_

print("Explained Variance by each Principal Component:")

print("Principal Component 1: ",explained\_variance[0])

print("Principal Component 2: ",explained\_variance[1])

print("Total Variance Retained: ",sum(explained\_variance))

Experiment 4: **For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.**

import pandas as pd

data = pd.read\_csv(r"C:\Users\Asus\Desktop\training\_data.csv")

print(data)

def find\_s\_algorithm(data):

"""Implements the Find-S algorithm to find the most specific hypothesis."""

# Extract feature columns and target column

attributes = data.iloc[:, :-1].values # All columns except last

target = data.iloc[:, -1].values # Last column (class labels)

# Step 1: Initialize hypothesis with first positive example

for i in range(len(target)):

if target[i] == "Yes": # Consider only positive examples

hypothesis = attributes[i].copy()

break

# Step 2: Update hypothesis based on other positive examples

for i in range(len(target)):

if target[i] == "Yes":

for j in range(len(hypothesis)):

if hypothesis[j] != attributes[i][j]:

hypothesis[j] = '?' # Generalize inconsistent attributes

return hypothesis

# Run Find-S Algorithm

final\_hypothesis = find\_s\_algorithm(data)

# Print the learned hypothesis

print("Most Specific Hypothesis:", final\_hypothesis)

**Experiment 5:Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.**

1. Label the first 50 points {x1,......,x50} as follows: if (xi ≤ 0.5), then xi ∊ Class1, else xi ∊ Class1
2. Classify the remaining points, x51,......,x100 using KNN. Perform this for k=1,2,3,4,5,20,30

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import warnings

warnings.filterwarnings('ignore')

# Step 1: Generate dataset

np.random.seed(42)

values = np.random.rand(100)

labels = []

for i in values[:50]:

if i <=0.5:

labels.append('Class1')

else:

labels.append('Class2')

labels += [None] \* 50

data = {

"Point": [f"x{i+1}" for i in range(100)],

"Value": values,

"Label": labels

}

print(data)

type(data)

df = pd.DataFrame(data)

df.head()

df.nunique()

df.shape

df.info()

df.describe().T

df.isnull().sum()

num\_col = df.select\_dtypes(include=['int', 'float']).columns

for col in num\_col:

df[col].hist(bins=10, alpha=0.5, edgecolor='black',grid=False)

plt.title(f'Histogram for {col}')

plt.xlabel(col)

plt.ylabel('Frequency')

plt.show()

# Split data into labeled and unlabeled

labeled\_df = df[df["Label"].notna()]

X\_train = labeled\_df[["Value"]]

y\_train = labeled\_df["Label"]

unlabeled\_df = df[df["Label"].isna()]

X\_test = unlabeled\_df[["Value"]]

# Generate true labels for testing (for accuracy calculation)

true\_labels = ["Class1" if x <= 0.5 else "Class2" for x in values[50:]]

# Step 2: Perform KNN classification for different values of k

k\_values = [1, 2, 3, 4, 5, 20, 30]

results = {}

accuracies = {}

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

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

predictions = knn.predict(X\_test)

results[k] = predictions

# Calculate accuracy

accuracy = accuracy\_score(true\_labels, predictions) \* 100

accuracies[k] = accuracy

print(f"Accuracy for k={k}: {accuracy:.2f}%")

# Assign predictions back to the DataFrame for this k

unlabeled\_df[f"Label\_k{k}"] = predictions

print(predictions)

df1 = unlabeled\_df.drop(columns=['Label'], axis=1)

df1

# Display accuracies

print("\nAccuracies for different k values:")

for k, acc in accuracies.items():

print(f"k={k}: {acc:.2f}%")

**Experiment 6: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make\_pipeline

from scipy.spatial.distance import cdist

# Load datasets

df\_linear = pd.read\_csv("linear\_dataset.csv")

df\_lwr = pd.read\_csv("lwr\_dataset.csv")

df\_poly = pd.read\_csv("polynomial\_dataset.csv")

# Linear Regression

def linear\_regression(df):

X, y = df[['X']], df['Y']

model = LinearRegression()

model.fit(X, y)

y\_pred = model.predict(X)

plt.scatter(X, y, label='Data')

plt.plot(X, y\_pred, color='red', label='Linear Regression')

plt.legend()

plt.title("Linear Regression")

plt.show()

linear\_regression(df\_linear)

# Locally Weighted Regression (LWR)

def gaussian\_kernel(x, X, tau):

return np.exp(-cdist([[x]], X, 'sqeuclidean') / (2 \* tau\*\*2))

def locally\_weighted\_regression(X\_train, y\_train, tau=0.5):

X\_train = np.hstack([np.ones((X\_train.shape[0], 1)), X\_train]) # Add intercept

X\_range = np.linspace(X\_train[:, 1].min(), X\_train[:, 1].max(), 100)

y\_pred = []

for x in X\_range:

x\_vec = np.array([1, x]) # Intercept term

weights = gaussian\_kernel(x, X\_train[:, 1:], tau).flatten()

W = np.diag(weights)

theta = np.linalg.pinv(X\_train.T @ W @ X\_train) @ (X\_train.T @ W @ y\_train)

y\_pred.append(x\_vec @ theta) # Use dot product for prediction

plt.scatter(X\_train[:, 1], y\_train, label='Data')

plt.plot(X\_range, y\_pred, color='red', label='LWR')

plt.legend()

plt.title("Locally Weighted Regression")

plt.show()

# Run the models

locally\_weighted\_regression(df\_lwr[['X']].values, df\_lwr['Y'].values)

# Polynomial Regression

def polynomial\_regression(df, degree=3):

X, y = df[['X']], df['Y']

model = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

model.fit(X, y)

y\_pred = model.predict(X)

plt.scatter(X, y, label='Data')

plt.plot(X, y\_pred, color='red', label=f'Polynomial Regression (deg={degree})')

plt.legend()

plt.title("Polynomial Regression")

plt.show()

polynomial\_regression(df\_poly, degree=3)

**Experiment 7 A: Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression**.

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.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

import warnings

warnings.filterwarnings('ignore')

data=pd.read\_csv(r"C:\Users\Asus\Documents\ML6thSEM\_FDP\_Day2\ML6thSEM\_FDP\_Day2\Experiment\_7\_Lin\_Poly\_reg\Boston housing dataset.csv")

data.head()

data.shape

data.info()

data.nunique()

data.ZN.unique()

# \*\*Data Cleaning\*\*

data.isnull().sum()

data.duplicated().sum()

df = data.copy()

df.isnull().sum()

df.head()

df['CHAS'] = df['CHAS'].astype('int')

df.describe().T

for i in df.columns:

plt.figure(figsize=(6,3))

plt.subplot(1, 2, 1)

df[i].hist(bins=20, alpha=0.5, color='b',edgecolor='black')

plt.title(f'Histogram of {i}')

plt.xlabel(i)

plt.ylabel('Frequency')

plt.subplot(1, 2, 2)

plt.boxplot(df[i], vert=False)

plt.title(f'Boxplot of {i}')

plt.show()

corr = df.corr(method='pearson')

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

sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.xticks(rotation=90, ha='right')

plt.yticks(rotation=0)

plt.title("Correlation Matrix Heatmap")

plt.show()

X = df.drop('MEDV', axis=1) # All columns except 'MEDV'

y = df['MEDV'] # Target variable

# Scale the features

scale = StandardScaler()

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

# Split the data into training (80%) and testing (20%) sets

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

# Initialize the linear regression model

model = LinearRegression()

# Fit the model on the training data

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

# Predict on the test set

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

y\_pred

# Calculate Mean Squared Error

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

# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(mse)

# Calculate R-squared value

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

print(f'Mean Squared Error: {mse}')

print(f'Root Mean Squared Error: {rmse}')

print(f'R-squared: {r2}')

Experiment 7 B Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

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

import warnings

warnings.filterwarnings("ignore")

sns.get\_dataset\_names()

data = sns.load\_dataset('mpg')

data.head()

data.shape

data.info()

data.nunique()

data.horsepower.unique()

### \*\*Data Cleaning\*\*

data.isnull().sum()

data.duplicated().sum()

### \*\*Data Handling\*\*

df = data.copy()

df['horsepower'].fillna(df['horsepower'].median(), inplace=True)

df.describe().T

### \*\*EDA\*\*

numerical = df.select\_dtypes(include=['int','float']).columns

categorical = df.select\_dtypes(include=['object']).columns

print(numerical)

print(categorical)

for i in numerical:

plt.figure(figsize=(10,4))

plt.subplot(1, 2, 1)

df[i].hist(bins=20, alpha=0.5, color='b',edgecolor='black')

plt.title(f'Histogram of {i}')

plt.xlabel(i)

plt.ylabel('Frequency')

plt.subplot(1, 2, 2)

plt.boxplot(df[i], vert=False)

plt.title(f'Boxplot of {i}')

plt.show()

import seaborn as sns

for col in categorical:

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

sns.countplot(x=col, data=df, order=df[col].value\_counts().sort\_values().head(10).index, palette='viridis')

plt.title(f'Countplot of {col}')

plt.xticks(rotation=90)

plt.show()

# Select the relevant features

X = df[['horsepower']] # You can select other features here

y = df['mpg']

# Split the data into training and test sets

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

# Create polynomial features

degree = 2 # Change the degree of the polynomial

poly = PolynomialFeatures(degree)

X\_poly\_train = poly.fit\_transform(X\_train)

# Fit a polynomial regression model

model = LinearRegression()

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

# Fit a polynomial regression model

model = LinearRegression()

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

# Make predictions

X\_poly\_test = poly.transform(X\_test)

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

# Visualize the results

plt.scatter(X, y, color='blue', label='Data')

X\_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)

X\_range\_poly = poly.transform(X\_range)

y\_range\_pred = model.predict(X\_range\_poly)

plt.plot(X\_range, y\_range\_pred, color='red', label='Polynomial Fit')

plt.xlabel('Horsepower')

plt.ylabel('MPG')

plt.legend()

plt.title(f'Polynomial Regression (degree {degree})')

plt.show()

# Evaluate the model on the test set

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

rmse = np.sqrt(mse)

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

# Print the evaluation metrics

print(f'Mean Squared Error (MSE): {mse:.2f}')

print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')

print(f'R-squared (R²): {r2:.2f}')

**Experiment 8 Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and applying this knowledge to classify a new sample.**

# Importing necessary libraries

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.tree import DecisionTreeClassifier, plot\_tree

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

from sklearn.tree import export\_graphviz

from IPython.display import Image

import pydotplus

import warnings

warnings.filterwarnings('ignore')

data=pd.read\_csv(r'C:\Users\Admin\OneDrive\Documents\MachineLearning Lab\Datasets\Breast Cancer Dataset.csv')

pd.set\_option('display.max\_columns', None)

data.head()

data.shape

data.info()

data.diagnosis.unique()

### Data Preprocessing

#### Data Cleaning

data.isnull().sum()

data.duplicated().sum()

df = data.drop(['id'], axis=1)

df['diagnosis'] = df['diagnosis'].map({'M':1, 'B':0}) # Malignant:1, Benign:0

df.describe().T

corr = df.corr(method='pearson')

plt.figure(figsize=(18, 10))

sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

plt.xticks(rotation=90, ha='right')

plt.yticks(rotation=0)

plt.title("Correlation Matrix Heatmap")

plt.show()

X = df.drop('diagnosis', axis=1) # Drop the 'diagnosis' column (target)

y = df['diagnosis']

# Split the dataset into training and testing sets (80% train, 20% test)

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

# Fit the decision tree model

model = DecisionTreeClassifier(criterion='entropy') #criteria = gini, entropy

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

model

import math

# Function to calculate entropy

def entropy(column):

counts = column.value\_counts()

probabilities = counts / len(column)

return -sum(probabilities \* probabilities.apply(math.log2))

# Function to calculate conditional entropy

def conditional\_entropy(data, X, target):

feature\_values = data[X].unique() # Corrected: use .unique() on the series

weighted\_entropy = 0

for value in feature\_values:

subset = data[data[feature] == value]

weighted\_entropy += (len(subset) / len(data)) \* entropy(subset[target])

return weighted\_entropy

# Function to calculate information gain

def information\_gain(data, X, target):

total\_entropy = entropy(data[target])

feature\_conditional\_entropy = conditional\_entropy(data, X, target)

return total\_entropy - feature\_conditional\_entropy

# Calculate information gain for each feature

for feature in X:

ig = information\_gain(df,feature,'diagnosis')

print(f"Information Gain for {feature}: {ig}")

# Export the tree to DOT format

dot\_data = export\_graphviz(model, out\_file=None,

feature\_names=X\_train.columns,

rounded=True, proportion=False,

precision=2, filled=True)

# Convert DOT data to a graph

graph = pydotplus.graph\_from\_dot\_data(dot\_data)

# Display the graph

Image(graph.create\_png())

# Visualize the Decision Tree (optional)

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

plot\_tree(model, filled=True, feature\_names=X.columns, class\_names=['Benign', 'Malignant'], rounded=True)

plt.show()

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

y\_pred

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred) \* 100

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

# Print the results

print("Accuracy:", accuracy)

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

df.head(1)

new = [[12.5, 19.2, 80.0, 500.0, 0.085, 0.1, 0.05, 0.02, 0.17, 0.06,

0.4, 1.0, 2.5, 40.0, 0.006, 0.02, 0.03, 0.01, 0.02, 0.003,

16.0, 25.0, 105.0, 900.0, 0.13, 0.25, 0.28, 0.12, 0.29, 0.08]]

y\_pred = model.predict(new)

# Output the prediction (0 = Benign, 1 = Malignant)

if y\_pred[0] == 0:

print("Prediction: Benign")

else:

print("Prediction: Malignant")

Experiment 9 Develop a program to implement the Naive Bayesian classifier, considering the Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data set.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_olivetti\_faces

data = fetch\_olivetti\_faces()

data.keys()

print("Data Shape:", data.data.shape)

print("Target Shape:", data.target.shape)

print("There are {} unique persons in the dataset".format(len(np.unique(data.target))))

print("Size of each image is {}x{}".format(data.images.shape[1],data.images.shape[1]))

def print\_faces(images, target, top\_n):

# Ensure the number of images does not exceed available data

top\_n = min(top\_n, len(images))

# Set up figure size based on the number of images

grid\_size = int(np.ceil(np.sqrt(top\_n)))

fig, axes = plt.subplots(grid\_size, grid\_size, figsize=(15, 15))

fig.subplots\_adjust(left=0, right=1, bottom=0, top=1, hspace=0.2, wspace=0.2)

for i, ax in enumerate(axes.ravel()):

if i < top\_n:

ax.imshow(images[i], cmap='bone')

ax.axis('off')

ax.text(2, 12, str(target[i]), fontsize=9, color='red')

ax.text(2, 55, f"face: {i}", fontsize=9, color='blue')

else:

ax.axis('off')

plt.show()

#let us extract unique charaters present in dataset

def display\_unique\_faces(pics):

fig = plt.figure(figsize=(24, 10)) # Set figure size

columns, rows = 10, 4 # Define grid dimensions

# Loop through grid positions and plot each image

for i in range(1, columns \* rows + 1):

img\_index = 10 \* i - 1 # Calculate the image index

if img\_index < pics.shape[0]: # Check for valid image index

img = pics[img\_index, :, :]

ax = fig.add\_subplot(rows, columns, i)

ax.imshow(img, cmap='gray')

ax.set\_title(f"Person {i}", fontsize=14)

ax.axis('off')

plt.suptitle("There are 40 distinct persons in the dataset", fontsize=24)

plt.show()

display\_unique\_faces(data.images)

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

X = data.data

Y = data.target

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.3, random\_state=46)

print("x\_train: ",x\_train.shape)

print("x\_test: ",x\_test.shape)

from sklearn.naive\_bayes import GaussianNB

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

# Train the model

nb = GaussianNB()

nb.fit(x\_train, y\_train)

# Predict the test set results

y\_pred = nb.predict(x\_test)

# Calculate accuracy

nb\_accuracy = round(accuracy\_score(y\_test, y\_pred) \* 100, 2)

# Display the confusion matrix

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

print("Confusion Matrix:")

print(cm)

# Display accuracy result

print(f"Naive Bayes Accuracy: {nb\_accuracy}%")

from sklearn.naive\_bayes import MultinomialNB

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

# Initialize and fit Multinomial Naive Bayes

nb = MultinomialNB()

nb.fit(x\_train, y\_train)

# Predict the test set results

y\_pred = nb.predict(x\_test)

# Calculate accuracy

accuracy = round(accuracy\_score(y\_test, y\_pred) \* 100, 2)

print(f"Multinomial Naive Bayes Accuracy: {accuracy}%")

Calculate the number of misclassified images

misclassified\_idx = np.where(y\_pred != y\_test)[0]

num\_misclassified = len(misclassified\_idx)

# Print the number of misclassified images and accuracy

print(f"Number of misclassified images: {num\_misclassified}")

print(f"Total images in test set: {len(y\_test)}")

print(f"Accuracy: {round((1 - num\_misclassified / len(y\_test)) \* 100, 2)}%")

# Visualize some of the misclassified images

n\_misclassified\_to\_show = min(num\_misclassified, 5) # Show up to 5 misclassified images

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

for i in range(n\_misclassified\_to\_show):

idx = misclassified\_idx[i]

plt.subplot(1, n\_misclassified\_to\_show, i + 1)

plt.imshow(x\_test[idx].reshape(64, 64), cmap='gray')

plt.title(f"True: {y\_test[idx]}, Pred: {y\_pred[idx]}")

plt.axis('off')

plt.show()

from sklearn.preprocessing import label\_binarize

from sklearn.metrics import roc\_auc\_score

# Binarize the test labels

y\_test\_bin = label\_binarize(y\_test, classes=np.unique(y\_test))

# Get predicted probabilities for each class

y\_pred\_prob = nb.predict\_proba(x\_test)

# Calculate and print AUC for each class

for i in range(y\_test\_bin.shape[1]):

roc\_auc = roc\_auc\_score(y\_test\_bin[:, i], y\_pred\_prob[:, i])

print(f"Class {i} AUC: {roc\_auc:.2f}")