# Practical name: Program to find Mean, Mode, and Median

# ------------------------------------------------------------

from collections import Counter

# Function to calculate the mean

def calculate\_mean(data):

return sum(data) / len(data)

# Function to calculate the median

def calculate\_median(data):

data\_sorted = sorted(data)

n = len(data)

mid = n // 2

if n % 2 == 0:

# If even, take the average of the two middle numbers

median = (data\_sorted[mid - 1] + data\_sorted[mid]) / 2

else:

# If odd, take the middle number

median = data\_sorted[mid]

return median

# Function to calculate the mode

def calculate\_mode(data):

count = Counter(data)

max\_count = max(count.values())

# Find all elements with the highest frequency

mode = [k for k, v in count.items() if v == max\_count]

return mode

# Input data

data = [12, 3, 4, 4, 6, 6, 8]

# Calculations

mean = calculate\_mean(data)

median = calculate\_median(data)

mode = calculate\_mode(data)

# Output results

print("Mean:", mean)

print("Median:", median)

print("Mode:", mode)

*#Practical name:Create an array containing 80 random floats between 0 and 6*

import numpy as np

# Generate an array of 80 random floats between 0 and 6

x = np.random.uniform(0.0, 6.0, 80)

# Print the generated array

print(x)

# Practical Name: Create an array containing 80 random floats between 0 and 6 and draw a histogram

# Importing required libraries

import numpy as np

import matplotlib.pyplot as plt

# Generate an array of 250 random floats with a normal distribution (mean=0.0, std dev=8.0)

x = np.random.normal(0.0, 8.0, 250)

# Plot the histogram of the generated data with 10 bins

plt.hist(x, bins=10, edgecolor='black', color='skyblue')

# Display the histogram

plt.title("Histogram of Random Data")

plt.xlabel("Value")

plt.ylabel("Frequency")

plt.show()

# Practical Name: Distribution of Height using Normal Distribution

# -----------------------------------------------------------------

# Importing required libraries

import numpy as np

import matplotlib.pyplot as plt

# Parameters for the normal distribution

mean\_height = 200 # Mean height in cm

std\_dev\_height = 10 # Standard deviation in cm

num\_samples = 1000 # Number of samples

# Generating random heights based on the normal distribution

heights = np.random.normal(loc=mean\_height, scale=std\_dev\_height, size=num\_samples)

# Plotting the histogram of heights

plt.hist(heights, bins=30, density=True, alpha=0.6, color='white', edgecolor='black')

# Adding title and labels

plt.title('Distribution of Heights')

plt.xlabel('Height (cm)')

plt.ylabel('Density')

# Displaying the histogram

plt.show()

# Practical Name: Normal Distribution of Data

# ---------------------------------------------------------------

# Importing required libraries

import numpy as np

import matplotlib.pyplot as plt

# Parameters for the normal distribution

mean = 0 # Mean of the distribution

std\_dev = 1 # Standard deviation of the distribution

num\_samples = 100 # Number of samples to generate

# Generating data using the normal distribution

data = np.random.normal(loc=mean, scale=std\_dev, size=num\_samples)

# Plotting the histogram of the data

plt.hist(data, bins=20, density=True, alpha=0.9, color='white', edgecolor='blue')

# Adding title and labels

plt.title('Histogram of Normally Distributed Data')

plt.xlabel('Value')

plt.ylabel('Density')

# Displaying the histogram

plt.show()

# Practical Name: Draw Scatter Plot of Linear Regression

# Importing the required library

import matplotlib.pyplot as plt

# Data points for x and y

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

# Plotting the scatter plot

plt.scatter(x, y, color='purple', marker='o')

# Adding title and axis labels

plt.title('Scatter Plot Example')

plt.xlabel('X values')

plt.ylabel('Y values')

# Displaying the scatter plot

plt.show()

# Practical Name: Draw the Line of Linear Regression

# ---------------------------------------------------------------

# Importing the required libraries

import matplotlib.pyplot as plt

from scipy import stats

# Data points for x and y

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

# Performing linear regression

slope, intercept, r, p, std\_err = stats.linregress(x, y)

# Defining the linear regression function

def myfunc(x):

return slope \* x + intercept

# Applying the regression function to the data points

mymodel = list(map(myfunc, x))

# Plotting the scatter plot of the data points

plt.scatter(x, y, color='purple', label='Data Points')

# Plotting the linear regression line

plt.plot(x, mymodel, color='blue', label='Linear Regression Line')

# Adding title and axis labels

plt.title('Linear Regression Example')

plt.xlabel('X values')

plt.ylabel('Y values')

# Adding a legend

plt.legend()

# Displaying the plot

plt.show()

# Practical Name: Predict the Speed of a 5-Year-Old Car

# ------------------------------------------------------------

# Importing required libraries

import matplotlib.pyplot as plt

from scipy import stats

# Data: Age of cars (x) and their respective speeds (y)

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]

y = [99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86]

# Performing linear regression

slope, intercept, r, p, std\_err = stats.linregress(x, y)

# Function to predict speed based on car age

def predict\_speed(age):

return slope \* age + intercept

# Predicting the speed for a 5-year-old car

predicted\_speed = predict\_speed(5)

print(f"Predicted speed for a 5-year-old car: {predicted\_speed:.2f} units")

# Visualizing the data and regression line

# Scatter plot of the original data

plt.scatter(x, y, color='purple', label='Data Points')

# Plotting the regression line

mymodel = [predict\_speed(age) for age in x]

plt.plot(x, mymodel, color='blue', label='Regression Line')

# Adding labels and title

plt.title('Car Age vs Speed')

plt.xlabel('Age of Car (years)')

plt.ylabel('Speed (units)')

plt.legend()

# Displaying the plot

plt.show()

# Practical Name: Print the Coefficient Values of the Regression Object

# ------------------------------------------------------------

# Importing required libraries

import numpy as np

from sklearn.linear\_model import LinearRegression

# Input data: Features (x) and target (y)

x = np.array([[1], [2], [3], [4], [5]])

y = np.array([2, 4, 6, 8, 10])

# Creating and training the Linear Regression model

model = LinearRegression()

model.fit(x, y)

# Extracting the coefficients and intercept

coefficients = model.coef\_

intercept = model.intercept\_

# Printing the results

print("Coefficients:", coefficients)

print("Intercept:", intercept)

# Practical Name: Make Circles - Generate 2D Binary Classification Data with a Spherical Decision Boundary

# Import necessary libraries

from sklearn.datasets import make\_circles

import matplotlib.pyplot as plt

# Generate 2D classification dataset

x, y = make\_circles(

n\_samples=100, # Number of samples

shuffle=True, # Shuffle the dataset

noise=0.1, # Add some noise to the data

random\_state=10 # Random seed for reproducibility

)

# Plot the generated dataset

plt.scatter(

x[:, 0], # First feature

x[:, 1], # Second feature

c=y, # Color based on target labels

cmap='viridis' # Colormap for better visualization

)

# Add title and axis labels

plt.title('2D Binary Classification Data (Circles)')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

# Display the plot

plt.show()

# Practical Name: Display the Plot using plot() and show() from pyplot

# Importing the required library

import matplotlib.pyplot as plt

# Sample data

x = [1, 2, 3, 4, 5]

y = [2, 3, 5, 7, 11]

# Creating the plot

plt.plot(

x,

y,

marker='o', # Add markers at data points

linestyle='-', # Use a solid line

color='b' # Line color: blue

)

# Adding title and labels

plt.title('Sample Line Plot') # Plot title

plt.xlabel('X-axis label') # X-axis label

plt.ylabel('Y-axis label') # Y-axis label

# Displaying the plot

plt.show()

# Practical Name: Data Generated by the Function make\_blobs()

# ----------------------------------------------------------------------------------

# Import necessary libraries

from sklearn.datasets import make\_blobs

import matplotlib.pyplot as plt

# Generate 2D classification dataset

x, y = make\_blobs(

n\_samples=20, # Number of samples

centers=3, # Number of centers (clusters)

n\_features=2, # Number of features (dimensions)

random\_state=20 # Random seed for reproducibility

)

# Plot the generated dataset

plt.scatter(

x[:, 0], # First feature (x-axis)

x[:, 1], # Second feature (y-axis)

c=y, # Color points by cluster

cmap='viridis', # Colormap for better visualization

edgecolor='k' # Add black borders to points

)

# Add title and labels

plt.title('Blobs Dataset Visualization')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

# Display the plot

plt.show()

*#Practical Name:Random multi-label classification data is created by the function make make\_multi*

**from** sklearn.datasets **import** make\_multilabel\_classification *# Define dataset*X, y **=** make\_multilabel\_classification(n\_samples**=**10, n\_features**=**10, n\_classes**=**3, n\_labels**=**2, rando *# Summarize dataset shape*print(X.shape, y.shape) *# Summarize first few examples***for** i **in** range(10): print(X[i], y[i])

*#Practical Name:Random multi-label classification data is created by the function make make\_multi*

*Import necessary libraries***from** sklearn.datasets **import** make\_multilabel\_classification**import** pandas **as** pd**import** matplotlib.pyplot **as** plt *# Generate 2D classification dataset*X, y **=** make\_multilabel\_classification(n\_samples**=**50, n\_features**=**2, n\_classes**=**2, n\_labels**=**2, allow\_ *# Create pandas DataFrame from generated dataset*df **=** pd.concat([pd.DataFrame(X, columns**=**['X1', 'X2']), pd.DataFrame(y, columns**=**['Label1', 'Label2'])], axis**=**1) *# Display the first few rows of the DataFrame*display(df.head()) *# Plot the generated datasets*plt.scatter(df['X1'], df['X2'], c**=**df['Label1'])plt.show()

Write a python program to implement the KNN algorithm.

# Import necessary libraries

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

# Load the Iris dataset

irisData = load\_iris()

# Create feature (X) and target (y) arrays

X = irisData.data

y = irisData.target

# 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

)

# Initialize the KNeighborsClassifier with 7 neighbors

knn = KNeighborsClassifier(n\_neighbors=7)

# Train the model on the training data

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

# Predict on the test data (dataset not seen by the model)

predictions = knn.predict(X\_test)

# Print predictions

print("Predictions:", predictions)

Write a python program to creating a dataframe to implement one hot encoding from CSV file.

import pandas as pd

from sklearn.preprocessing import OneHotEncoder

# Create the DataFrame directly

data = {

'CustomerID': [1, 2, 3, 4, 5, 6],

'Age': [25, 30, 22, 35, 28, 18],

'Gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Male'],

}

df = pd.DataFrame(data)

# Display the first few rows of the DataFrame

print("Original DataFrame:")

print(df.head())

# Select the categorical columns to be one-hot encoded

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

# Apply one-hot encoding using pandas’ get\_dummies

df\_encoded = pd.get\_dummies(df, columns=categorical\_columns, drop\_first=True)

# Display the transformed DataFrame

print("\nDataFrame after One-Hot Encoding:")

print(df\_encoded.head())