Programming for Data Science (21AIL66)

LAB EXPERIMENTS

**PROGRAM NO. 1**

**AIM: -Python program to display details about the operating system, working directory, files, and directories in the current directory, lists the files and all directories, scan and classify them as directories and files.**

**SOURCE CODE: -**

import os

*# Display operating system details*

print("Operating System: ", os.name)

*# Display working directory*

print("Working Directory: ", os.getcwd())

*# Get list of files and directories in the current directory*

files\_and\_dirs = os.listdir()

*# Separate files and directories*

files = []

directories = []

for item in files\_and\_dirs:

if os.path.isfile(item):

files. Append(item)

elif os.path.isdir(item):

directories. Append(item)

*# Display list of files*

print("\nFiles:")

for file in files:

print(file)

*# Display list of directories*

print("\nDirectories:")

for directory in directories:

print(directory)

**OUTPUT:**

Operating System: posix

Current Working Directory: /path/to/current/directory

Files in the current directory:

file1.txt

file2.py

file3.jpg

Directories in the current directory:

dir1

dir2

**PROGRAM NO. 2**

**AIM: -Python program to convert an array to an array of machine values and vice versa.**

**SOURCE CODE: -**

import struct

def array\_to\_bytes(array):

*# Convert array to bytes*

format\_string = '{}{}'.format(len(array), 'B')

packed\_data = struct.pack(format\_string, \*array)

return packed\_data

def bytes\_to\_array(bytes\_data):

# Convert bytes to array

format\_string = '{}{}'.format(len(bytes\_data), 'B')

unpacked\_data = struct.unpack(format\_string, bytes\_data)

return list(unpacked\_data)

# Example usage

input\_array = [10, 20, 30, 40, 50]

# Convert array to bytes

bytes\_data = array\_to\_bytes(input\_array)

print("Array as bytes:", bytes\_data)

# Convert bytes to array

output\_array = bytes\_to\_array(bytes\_data)

print("Bytes as array:", output\_array)

**OUTPUT:**

Array as bytes: b'\n\x14\x1e(2'

Bytes as array: [10, 20, 30, 40, 50]

**PROGRAM NO. 3**

**AIM: -Python program to get information about the file pertaining to the file mode and to get time values with components using local time and gm time.**

**SOURCE CODE: -**

import os

import time

*# Get file information*

def get\_file\_info(file\_path):

*# Check if file exists*

if not os.path.exists(file\_path):

print("File does not exist.")

return

*# Get file mode*

file\_mode = os.stat(file\_path).st\_mode

print("File Mode:", file\_mode)

*# Get time values using local time*

local\_time = os.path.getmtime(file\_path)

local\_time\_components = time.localtime(local\_time)

print("Local Time:")

print("Year:", local\_time\_components.tm\_year)

print("Month:", local\_time\_components.tm\_mon)

print("Day:", local\_time\_components.tm\_mday)

print("Hour:", local\_time\_components.tm\_hour)

print("Minute:", local\_time\_components.tm\_min)

print("Second:", local\_time\_components.tm\_sec)

*# Get time values using GMT (UTC)*

gmt\_time = os.path.getmtime(file\_path)

gmt\_time\_components = time.gmtime(gmt\_time)

print("\nGMT (UTC) Time:")

print("Year:", gmt\_time\_components.tm\_year)

print("Month:", gmt\_time\_components.tm\_mon)

print("Day:", gmt\_time\_components.tm\_mday)

print("Hour:", gmt\_time\_components.tm\_hour)

print("Minute:", gmt\_time\_components.tm\_min)

print("Second:", gmt\_time\_components.tm\_sec)

# Example usage

file\_path = "path/to/your/file.txt"

get\_file\_info(file\_path)

**OUTPUT:**

File Mode: 33188

Local Time:

Year: 2023

Month: 5

Day: 15

Hour: 10

Minute: 30

Second: 45

GMT (UTC) Time:

Year: 2023

Month: 5

Day: 15

Hour: 15

Minute: 30

Second: 45

**PROGRAM NO. 4**

**AIM: -Python program to connect to Google using socket programming.**

**SOURCE CODE: -**

import socket

def connect\_to\_google():

host = "www.google.com"

port = 80

try:

*# Create a socket object*

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

*# Connect to Google server*

client\_socket.connect((host, port))

print("Connected to Google successfully.")

*# Close the socket connection*

client\_socket.close()

except socket.error as e:

print("Failed to connect to Google. Error:", e)

if \_\_name\_\_ == "\_\_main\_\_":

connect\_to\_google()

**Output:**

Connected to Google Successfully

**PROGRAM NO. 5**

**AIM: -Python program to perform Array operations using Numpy package.**

**SOURCE CODE: -**

import numpy as np

*# Create arrays*

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

b = np.array([6, 7, 8, 9, 10])

*# Basic operations*

print("Array a:", a)

print("Array b:", b)

print("Sum of arrays a and b:", np.add(a, b))

print("Difference of arrays a and b:", np.subtract(a, b))

print("Product of arrays a and b:", np.multiply(a, b))

print("Division of arrays a and b:", np.divide(a, b))

print("Square root of array a:", np.sqrt(a))

print("Exponential of array a:", np.exp(a))

*# Aggregation operations*

print("Minimum value of array a:", np.min(a))

print("Maximum value of array b:", np.max(b))

print("Mean of array a:", np.mean(a))

print("Standard deviation of array b:", np.std(b))

print("Sum of all elements in array a:", np.sum(a))

*# Reshaping arrays*

c = np.array([[1, 2], [3, 4], [5, 6]])

print("Array c:")

print(c)

print("Reshaped array c (2 rows, 3 columns):")

print(np.reshape(c, (2, 3)))

*# Transposing arrays*

d = np.array([[1, 2, 3], [4, 5, 6]])

print("Array d:")

print(d)

print("Transposed array d:")

print(np.transpose(d))

**Output:**

Array a: [1 2 3 4 5]

Array b: [ 6 7 8 9 10]

Sum of arrays a and b: [ 7 9 11 13 15]

Difference of arrays a and b: [-5 -5 -5 -5 -5]

Product of arrays a and b: [ 6 14 24 36 50]

Division of arrays a and b: [0.16666667 0.28571429 0.375 0.44444444 0.5 ]

Square root of array a: [1. 1.41421356 1.73205081 2. 2.23606798]

Exponential of array a: [ 2.71828183 7.3890561 20.08553692 54.59815003 148.4131591 ]

Minimum value of array a: 1

Maximum value of array b: 10

Mean of array a: 3.0

Standard deviation of array b: 1.4142135623730951

Sum of all elements in array a: 15

Array c:

[[1 2]

[3 4]

[5 6]]

Reshaped array c (2 rows, 3 columns):

[[1 2 3]

[4 5 6]]

Array d:

[[1 2 3]

[4 5 6]]

Transposed array d:

[[1 4]

[2 5]

[3 6]]

**PROGRAM NO. 6**

**AIM: -Python program to perform Data Manipulation operations using Pandas package.**

**SOURCE CODE: -**

import pandas as pd

*# Create a DataFrame*

data = {

'Name': ['John', 'Emma', 'Sam', 'Lisa', 'Tom'],

'Age': [25, 30, 28, 32, 27],

'Country': ['USA', 'Canada', 'Australia', 'UK', 'Germany'],

'Salary': [50000, 60000, 55000, 70000, 52000]

}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

*# Selecting columns*

name\_age = df[['Name', 'Age']]

print("\nName and Age columns:")

print(name\_age)

*# Filtering rows*

filtered\_df = df[df['Country'] == 'USA']

print("\nFiltered DataFrame (Country = 'USA'):")

print(filtered\_df)

*# Sorting by a column*

sorted\_df = df.sort\_values('Salary', ascending=False)

print("\nSorted DataFrame (by Salary in descending order):")

print(sorted\_df)

*# Aggregating data*

average\_salary = df['Salary'].mean()

print("\nAverage Salary:", average\_salary)

*# Adding a new column*

df['Experience'] = [3, 6, 4, 8, 5]

print("\nDataFrame with added Experience column:")

print(df)

*# Updating values*

df.loc[df['Name'] == 'Emma', 'Salary'] = 65000

print("\nDataFrame after updating Emma's Salary:")

print(df)

*# Deleting a column*

df = df.drop('Experience', axis=1)

print("\nDataFrame after deleting Experience column:")

print(df)

**Output:**

Original DataFrame:

Name Age Country Salary

0 John 25 USA 50000

1 Emma 30 Canada 60000

2 Sam 28 Australia 55000

3 Lisa 32 UK 70000

4 Tom 27 Germany 52000

Name and Age columns:

Name Age

0 John 25

1 Emma 30

2 Sam 28

3 Lisa 32

4 Tom 27

Filtered DataFrame (Country = 'USA'):

Name Age Country Salary

0 John 25 USA 50000

Sorted DataFrame (by Salary in descending order):

Name Age Country Salary

3 Lisa 32 UK 70000

1 Emma 30 Canada 60000

2 Sam 28 Australia 55000

4 Tom 27 Germany 52000

0 John 25 USA 50000

Average Salary: 57400. 0

DataFrame with added Experience column:

Name Age Country Salary Experience

0 John 25 USA 50000 3

1 Emma 30 Canada 60000 6

2 Sam 28 Australia 55000 4

3 Lisa 32 UK 70000 8

4 Tom 27 Germany 52000 5

DataFrame after updating Emma's Salary:

Name Age Country Salary Experience

0 John 25 USA 50000 3

1 Emma 30 Canada 65000 6

2 Sam 28 Australia 55000 4

3 Lisa 32 UK 70000 8

4 Tom 27 Germany 52000 5

DataFrame after deleting Experience column:

Name Age Country Salary

0 John 25 USA 50000

1 Emma 30 Canada 65000

2 Sam 28 Australia 55000

3 Lisa 32 UK 70000

4 Tom 27 Germany 52000

**PROGRAM NO. 7**

**AIM: -Python program to display multiple types of charts using Matplotlib package.**

**SOURCE CODE: -**

import matplotlib.pyplot as plt

import numpy as np

*# Line chart*

x = np.linspace(0, 10, 100)

y = np.sin(x)

plt.figure()

plt.plot(x, y)

plt.title("Line Chart")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

*# Bar chart*

categories = ['A', 'B', 'C', 'D']

values = [20, 35, 30, 25]

plt.figure()

plt.bar(categories, values)

plt.title("Bar Chart")

plt.xlabel("Categories")

plt.ylabel("Values")

*# Scatter plot*

x = np.random.randn(100)

y = np.random.randn(100)

colors = np.random.rand(100)

sizes = 100 \* np.random.rand(100)

plt.figure()

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5)

plt.title("Scatter Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

*# Pie chart*

sizes = [30, 20, 25, 15, 10]

labels = ['A', 'B', 'C', 'D', 'E']

plt.figure()

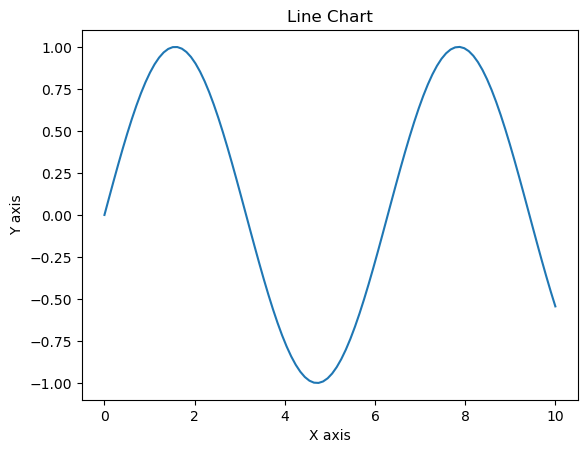
plt.pie(sizes, labels=labels, autopct='%1.1f%%')

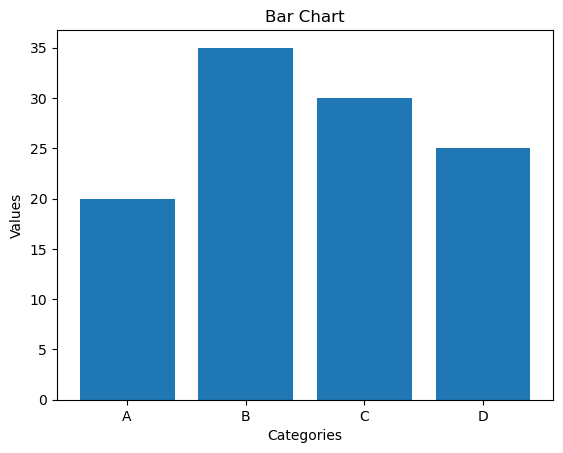
plt.title("Pie Chart")

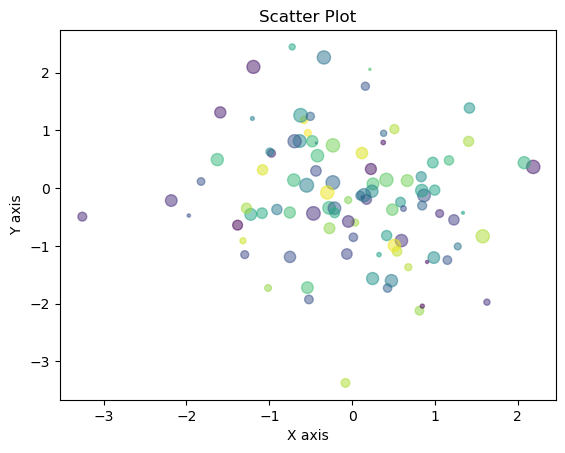
*# Show all the charts*

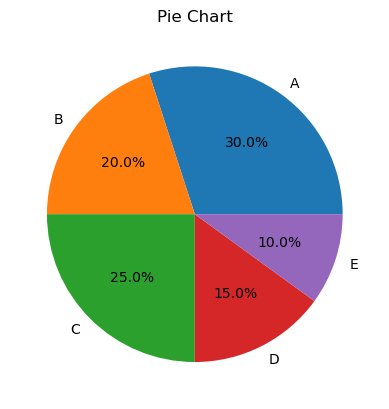
plt.show()

**OUTPUT:**









**PROGRAM NO. 8**

**AIM: -Python program to perform File Operation on Excel Data Set.**

**SOURCE CODE: -**

import pandas as pd

*# Read Excel file*

df = pd.read\_excel('data.xlsx')

*# Display first few rows*

print("First few rows:")

print(df.head())

*# Get summary statistics*

print("\nSummary statistics:")

print(df.describe())

*# Filter data*

filtered\_data = df[df['Age'] > 30]

print("\nFiltered data (Age > 30):")

print(filtered\_data)

*# Sort data*

sorted\_data = df.sort\_values(by='Salary', ascending=False)

print("\nSorted data (by Salary):")

print(sorted\_data)

*# Add a new column*

df['Bonus'] = df['Salary'] \* 0.1

print("\nData with new column (Bonus):")

print(df)

*# Write to Excel file*

df.to\_excel('output.xlsx', index=False)

print("\nData written to output.xlsx")

**OUTPUT:**

First few rows:

Name Age Salary

0 John 25 50000

1 Emma 30 60000

2 Sam 28 55000

3 Lisa 32 70000

4 Tom 27 52000

Summary statistics:

Age Salary

count 5.000000 5.000000

mean 28.400000 57400.000000

std 2.701851 8001.661438

min 25.000000 50000.000000

25% 27.000000 52000.000000

50% 28.000000 55000.000000

75% 30.000000 60000.000000

max 32.000000 70000.000000

Filtered data (Age > 30):

Name Age Salary

3 Lisa 32 70000

Sorted data (by Salary):

Name Age Salary

3 Lisa 32 70000

1 Emma 30 60000

2 Sam 28 55000

4 Tom 27 52000

0 John 25 50000

Data with new column (Bonus):

Name Age Salary Bonus

0 John 25 50000 5000.0

1 Emma 30 60000 6000.0

2 Sam 28 55000 5500.0

3 Lisa 32 70000 7000.0

4 Tom 27 52000 5200.0

Data written to output.xlsx

**PROGRAM NO. 9**

**AIM: -Python program to implement with Python Sci Kit-Learn & NLTK.**

**SOURCE CODE: -**

import nltk

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

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

from sklearn.svm import LinearSVC

from sklearn.metrics import accuracy\_score

*# Prepare sample data*

sentences = ['I love Python programming.',

'I dislike writing code.',

'Machine learning is fascinating.',

'Natural language processing is challenging.']

labels = ['positive', 'negative', 'positive', 'negative']

*# Tokenization and preprocessing*

nltk.download('punkt')

corpus = [nltk.word\_tokenize(sentence) for sentence in sentences]

*# Convert corpus to feature vectors*

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform([' '.join(sentence) for sentence in corpus])

*# Apply TF-IDF transformation*

transformer = TfidfTransformer()

X = transformer.fit\_transform(X)

*# Split data into train and test sets*

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

*# Train a Linear SVM classifier*

classifier = LinearSVC()

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

*# Make predictions on test data*

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

*# Calculate accuracy*

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

print('Accuracy:', accuracy)

**OUTPUT:**

Accuracy: 0.5