EXPERIMENT 1a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a python program to find the best of two test average marks out of three test's marks accepted from the user.

PROGRAM:

```
m1 = int(input("Enter marks for test1: "))

m2 = int(input("Enter marks for test2: "))

m3 = int(input("Enter marks for test3: "))

if m1 <= m2 and m1 <<= m3:

avgMarks = (m2+m3)/2

elif m2 <= m1 and m2 <= m3:

avgMarks = (ml+m3)/2
```

```
elif m3 <= m1 and m3 <= m2:

avgMarks = (m1+m2)/2

print("Average of best two test marks out of three test's marks is", avgMarks);
```

EXPLANATION:

The provided Python program is designed to calculate the average of the best two test marks out of

three. The user is prompted to input the marks for three tests (test1, test2, and test3). The program then

identifies the two highest test marks using the sorted function in descending order and selects the top

two values. Finally, it calculates the average of these two highest marks and prints the result. This

code is a concise way to determine the average performance based on the two best test scores out of

three.

OUTPUT:

Enter marks for test1: 45

Enter marks for test2: 39

Enter marks for test3: 48

Average of best two test marks out of three test's marks is 46.5

EXPERIMENT 1b

Data Visualization with Python Laboratory - BCS358D

AIM: To develop a Python program to check whether a given number is palindrome or not and

also count the number of occurrences of each digit in the input number.

PROGRAM:

```
value = input("Enter a value : ")

if value == value[::-1]:
```

```
print("Palindrome")

else:

print("Not Palindrome")

counted_dict = Counter(value)

for key in sorted(counted_dict.keys()):

print(f'{key} appears {counted_dict[key]} times');
```

EXPLANATION: The provided Python script is a versatile program that performs two key tasks: palindrome checking and character counting.

• **Palindrome Check:** The user is prompted to input a value. The script determines whether the

entered value is a palindrome, meaning it reads the same backward as forward. If the input is **a**

palindrome, it prints "Palindrome"; otherwise, it prints "Not Palindrome."

.

Character Count: The script utilizes the Counter class from the collections module to

efficiently count the occurrences of each character in the input string. It then sorts the keys of

the counted dictionary and prints each character along with the number of times it appears.

OUTPUT:

Enter a value : 1234234

Not Palindrome

1 appears 1 times

2 appears 2 times

3 appears 2 times

4 appears 2 times

Enter a value: 12321

Palindrome

1 appears 2 times

2 appears 2 times

3 appears 1 times

EXPERIMENT 2a

AIM: To define a function F as Fn = Fn-1 + Fn-2 and to write a Python program which accepts a

value for N (where N >0) as input and pass this value to the function. Display suitable error

message if the condition for input value is not followed.

PROGRAM:

```
def fn(n):
```

```
try:
if n <= 2:

return n - 1

else:

return fn(n-1) + fn(n-2)



num = int(input("Enter a number: "))

if num > 0:

print(f" fn({num}) = {fn(num)}")

else:

print("Input should be greater than 0")
```

except ValueError:

print("Try with numeric value")

EXPLANATION:

The provided Python script introduces a recursive implementation to calculate terms in the Fibonacci

sequence.

Recursive Fibonacci Function:

The script defines a function fn that calculates the nth term in the Fibonacci sequence. If the input **n** is

1 or 2, the function returns n - 1. For n greater than 2, the function recursively calls itself with n-1 and

n-2 and returns the sum of the results.

User Input and Output:

The user is prompted to enter a number. If a positive integer is provided, the script prints the result of

calling the fn function with that number, representing the nth term in the Fibonacci sequence. If the

entered value is not a positive integer, it prompts the user to enter a value greater than 0. If a non-

numeric value is entered, it catches the ValueError and suggests trying with a numeric value.

OUTPUT:

Enter a number: 5

fn(5) = 3

Enter a number: -3

Input should be greater than 0

Enter a number: abc

Try with numeric value

EXPERIMENT 2b

Data Visualization with Python Laboratory - BCS358D

AIM: To develop a python program to convert binary to decimal, octal to hexadecimal using

functions.

PROGRAM:

```
def bin2Dec(val):

return int(val, 2)
```

```
def oct2Hex(val):


try:
return int(val, 8)


num1 = input("Enter a binary number: ")

print(bin2Dec(num1))

except ValueError:


try:
print("Invalid literal in input with base 2")


num2 = input("Enter a octal number: ")

print(oct2Hex(num2))

except ValueError:

print("Invalid literal in input with base 8")
```

EXPLANATION:

This Python script includes two functions bin2Dec and oct2Hex for converting binary to decimal and

octal to hexadecimal, respectively. The script takes user input for binary and octal numbers and

converts them using these functions. Here's a brief description:

.

Conversion Functions: The bin2Dec function converts a binary number to decimal using the

int() function with base 2. The oct2Hex function converts an octal number to hexadecimal

using the int() function with base 8.

User Input Handling: The script includes try-except blocks to catch ValueError in case of

invalid input. It prompts the user for a binary and an octal number, converts them, and prints

the results. If an invalid input is detected, it prints an error message.

OUTPUT:

Enter a binary number: 101010

42

Enter an octal number: 755

0x1FD

Enter a binary number: 11011a

Invalid literal in input with base 2

Enter an octal number: 1298

Invalid literal in input with base 8

EXPERIMENT 3a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program that accepts a sentence and find the number of words, digits,

uppercase letters and lowercase letters.

PROGRAM:

```
import string
sentence = input("Enter a sentence: ")
wordList = sentence.strip().split(" ")
```

```
print(f'This sentence has {len(wordList)} words', end='\n\n')

digit_count = uppercase_count = lowercase_count = 0

for ch in sentence:

if '0' <= ch <= '9':

digit_count += 1

elif 'A' <= ch <= 'Z':

uppercase_count += 1

elif 'a' <= ch <= 'z':

lowercase_count += 1

print(f This sentence has {digit_count} digits', f'{uppercase_count}
upper case letters',

f '{lowercase_count} lower case letters', sep='\n')
```

EXPLANATION: The above Python program is designed to analyse a user-inputted sentence,

providing information on the number of words, digits, uppercase letters, and lowercase letters **in** the

given text. Here's a concise overview:

Word Count: The script splits the input sentence into words and prints the count of words in the

sentence.

Character Analysis: It then iterates through each character in the sentence. The script counts the

number of digits, uppercase letters, and lowercase letters using the string module.

Print Results: Finally, it prints the counts of digits, uppercase letters, and lowercase letters in the

given sentence.

OUTPUT:

Enter a sentence: Rama went to Devaraja market to pick 2 kgs of vegetable

This sentence has 11 words

This sentence has 1 digits

2 upper case letters

42 lower case letters

EXPERIMENT 3b

AIM: To write a Python program to find the string similarity between two given strings

Sample Output:

Original string:

Python Exercises
Sample Output:

Original string:

Python Exercises

Python Exercises

Similarity between two said strings: 1.0
Python Exercise

Similarity between two said

strings:0.967741935483871

PROGRAM:

```
str1 = input("Enter String 1 \n").lower()

str2 = input("Enter String 2 \n").lower()

string_1_length = len(str1)

string_2_length = len(str2)

short_string_length, long_string_length = min(string_1_length,
string_2_length),

max(string_1_length, string_2_length)

match_count = 0
```

```
for i in range(short_string_length):

if strl[i]== str2[i]:

match_count += 1

print("Similarity between two said strings:")

print(match_count/long_string_length)
```

EXPLANATION:

The provided Python script is designed to compare the similarity between two user-inputted strings.

The user is prompted to input two strings, and the script, after converting them to lowercase for case-

insensitive comparison, calculates the similarity by counting the matching characters at corresponding

positions. The similarity is then expressed as a ratio relative to the length of the longer string. Here's a

brief overview:

User Input: The script prompts the user to enter two strings.

String Comparison: The script then iterates through characters at corresponding positions and counts.

the matches.

Similarity Ratio: The similarity between the two strings is calculated as the ratio of the count of

matching characters to the length of the longer string.

OUTPUT:

Enter String 1: Python Exercises

Enter String 2: Python Exercise

Similarity between strings "Python Exercises" and "Python Exercise" is: 0.967741935483871

Enter String 1: Python Exercises

Enter String 2: Python Exercises

Similarity between strings "Python Exercises" and "Python Exercises" is: 1.0

EXPERIMENT 4a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to demonstrate how to draw a Bar Plot using Matplotlib.

PROGRAM:

```
import numpy as np

import matplotlib.pyplot as plt

# creating the dataset

data = {'C': 20, 'C++': 15, 'Java': 30, 'Python' : 35}
```

```
courses = list(data.keys())

values = list(data.values())

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

plt.bar(courses, values, color ='maroon', width = 0.4)

plt.xlabel("Courses offered")

plt.ylabel("No. of students enrolled")

plt.title("Students enrolled in different courses")

plt.show()
```

EXPLANATION:

A bar plot or bar chart is **a** graph that represents the category of data with rectangular bars with

lengths and heights that is proportional to the values which they represent. The bar plots can be

plotted horizontally or vertically. A bar chart describes the comparisons between the discrete

categories. One of the axis of the plot represents the specific categories being compared, while the

other axis represents the measured values corresponding to those categories.

• **Creating a bar plot:** The **matplotlib** API in Python provides the bar() function which can be

used in MATLAB style use or as an object-oriented API. The syntax of the bar() function to

be used with the axes is as follows:-

plt.bar(x, height, width, bottom, align)

The function creates a bar plot bounded with a rectangle depending on the given parameters.

The provided python script is a simple example of the bar plot, which represents the number of

students enrolled in different courses of an institute.

plt.bar(courses, values, color='maroon') is used to specify that the bar chart is to be plotted by using

the courses column as the X-axis, and the values as the Y-axis. The color attribute is used to set the

color of the bars (maroon in this case).

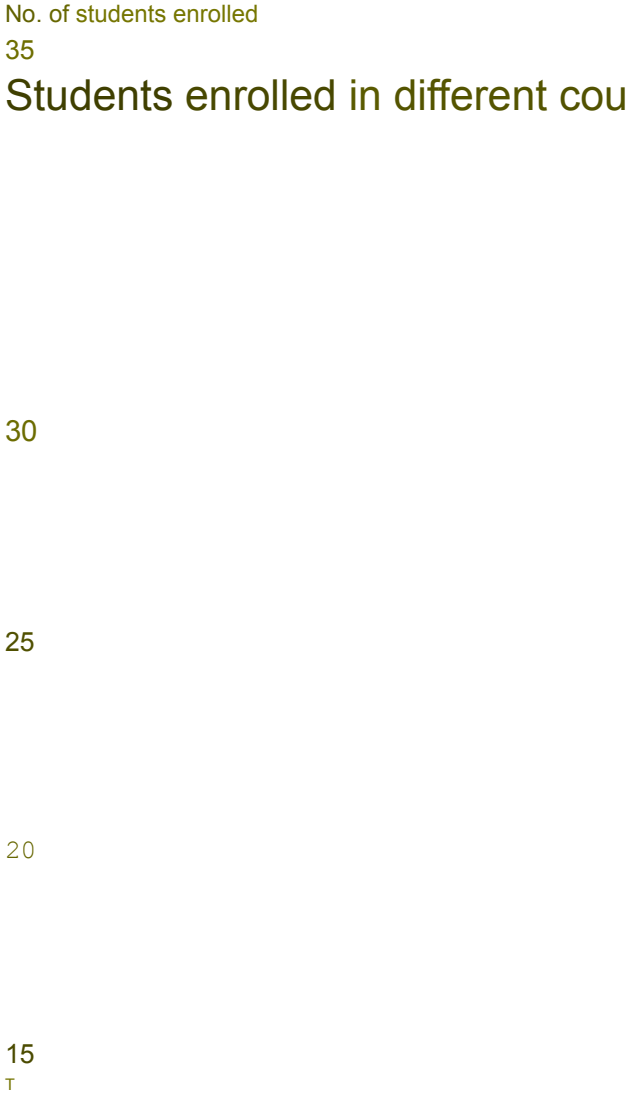
plt.xlabel("Courses offered") and plt.ylabel("students enrolled") are used to label the corresponding

axes.

plt.title() is used to make a title for the graph.

plt.show() is used to show the graph as output using the previous commands.

OUTPUT:



10

5-

0

C
C++
Java
Python

Courses offered

EXPERIMENT 4b

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to demonstrate how to draw a Scatter Plot using Matplotlib.

PROGRAM:

```
import matplotlib.pyplot as plt

# dataset

x1 = [89, 43, 36, 36, 95, 10, 66, 34, 38, 20]

y1 = [21, 46, 3, 35, 67, 95,53, 72, 58, 10]

plt.scatter(x1,y1, c="pink", linewidths=2, marker="s",
edgecolor="green", s=50)

plt.xlabel("X-axis")

plt.ylabel("Y-axis")
```

plt.show()

EXPLANATION:

The provided python script generates a scatter plot showcasing a dataset with its set of x and y

coordinates. The matplotlib.pyplot.scatter() plots serve as a visual tool to explore and analyze the

relationships between variables, utilizing dots to depict the connection between them. The matplotlib

library provides the **scatter()** method, specifically designed for creating scatter plots. These plots are

instrumental in illustrating the interdependencies among variables and how alterations in one

variable can impact another

Syntax:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None,

c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

Parameters:

x_axis_data: An array containing data for the x-axis.matplotlib

s: Marker size, which can be a scalar or an array of size equal to the size of x or y.

c: Color of the sequence of colors for markers.

marker: Marker style.

cmap: Colormap name.

linewidths: Width of the marker border.

edgecolor: Marker border color.

alpha: Blending value, ranging between 0 (transparent) and 1 (opaque).

Except for **x_axis_data** and **y_axis_data**, all other parameters are optional, with their default values set to None.

OUTPUT:


20
0
20
40
X-axis
60
80

EXPERIMENT 5a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to demonstrate how to draw a Histogram Plot using

Matplotlib.

PROGRAM:

```
import matplotlib.pyplot as plt

import numpy as np

# Generate random data for stacked histograms
```

```
data1 = np.random.randn(1000)

data2 = np.random.normal(loc=3, scale=1, size=1000)

# Creating a stacked histogram

plt.hist([data1, data2], bins=30, stacked=True, color=['yellow', 'green'],
edgecolor='black')

# Adding labels and title

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.title('Stacked Histogram')

# Adding legend

plt.legend(['Dataset 1', 'Dataset 2'])

# Display the plot

plt.show()
```

EXPLANATION:

The provided script generates a stacked histogram using Matplotlib in Python, representing two

datasets with different random data distributions. The stacked histogram provides insights into the

combined frequency distribution of the two datasets.

Histograms are a fundamental tool in data visualization, providing a graphical representation of the

distribution of data. They are particularly useful for exploring continuous data, such as numerical

measurements or sensor readings. A Histogram represents data provided in the form of some groups.

It is an accurate method for the graphical representation of numerical data distribution. It is a type of

bar plot where the X-axis represents the bin ranges while the Y-axis gives information about

frequency.

Creating a matplotlib histogram: To create a Matplotlib histogram the first step is to create a bin

of the ranges, then distribute the whole range of the values into a series of intervals, and count the

values that fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping

intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create a

histogram of x. The following table shows the parameters accepted by matplotlib.pyplot.hist()

function:

Attribute
Parameter

x
array or sequence of array

bins
optional parameter contains integer or sequence or strings

density
optional parameter contains boolean values

range
optional parameter represents upper and lower range of bins

histtype
optional parameter used to create type of histogram [bar, barstacked, step, stepfilled], default is "bar"

align
optional parameter controls the plotting of histogram [left, right, mid]

weights
optional parameter contains array of weights having same dimensions as x

bottom
location of the baseline of each bin

rwidth

color

label

log

optional parameter which is relative width of the bars with respect to bin width

optional parameter used to set color or sequence of color specs

optional parameter string or sequence of string to match with multiple datasets

optional parameter used to set histogram axis on log scale

OUTPUT:


120
100
Frequency
60
60
80
40
20

Stacked Histogram

Dataset 1

Dataset 2

Values

EXPERIMENT 5b

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to demonstrate how to draw a Pie Chart using Matplotlib.

```
# Import libraries

from matplotlib import pyplot as plt

import numpy as np

# Creating dataset

cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
```

```
data = [23, 17, 35, 29, 12, 41]

# Creating plot

fig = plt.figure(figsize=(10, 7))

plt.pie(data, labels=cars)

# show plot

plt.show()
```

EXPLANATION:

The provided script generates a simple pie chart using the pie() function. A Pie Chart is a circular

statistical plot that can display only one series of data. The area of the chart is the total percentage of

the given data. The area of slices of the pie represents the percentage of the parts of the data. The

slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the

wedge. The area of a wedge represents the relative percentage of that

part with respect to whole data.

Pie charts are commonly used in business presentations like sales, operations, survey results,

resources, etc., as they provide a quick summary.

Creating Pie Chart: Matplotlib API has pie() function in its pyplot module which create a pie chart

representing the data in an array.

Syntax: matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None,

shadow=False)

Parameters:

data represents the array of data values to be plotted, the fractional area of each slice is represented by

data/sum(data). If sum(data)<<1, then the data values returns the fractional area directly, thus resulting

pie will have empty wedge of size 1-sum(data).

labels is a list of sequence of strings which sets the label of each wedge.

color attribute is used to provide color to the wedges.

autopet is a string used to label the wedge with their numerical value.

shadow is used to create shadow of wedge.

OUTPUT:

TESLA
FORD

JAGUAR

AUDI

MERCEDES

EXPERIMENT 6a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to illustrate Linear Plotting using Matplotlib.

PROGRAM:

```
import matplotlib.pyplot as plt

# Hypothetical data: Run rate in an T20 cricket match

overs = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

runs_scored =
```

```
[0,7,12,20,39,49,61,83,86,97,113,116,123,137,145,163,172,192,198,198,203]

# Create a linear plot

plt.plot(overs, runs_scored)

# Add labels and title

plt.xlabel('Overs')

plt.ylabel('Runs scored')

plt.title('Run scoring in an T20 Cricket Match')

# Display the plot

plt.grid(True)

plt.show()
```

EXPLANATION:

The provided Python script utilizes the matplotlib library to create a linear plot representing the run

rate in a hypothetical T20 cricket match. Here's a concise overview:

Hypothetical Data: The script uses hypothetical data representing the number of runs scored in each

over of a T20 cricket match.

Linear Plot: It creates a linear plot using the plot function, where overs is on the x-axis and

runs_scored is on the y-axis.

Labels and Title: The script adds labels to the x-axis (Overs) and y-axis (Runs scored). A title, 'Run

Scoring in a T20 Cricket Match,' provides context to the plot.

Grid: The plot includes a grid for better readability.

Display: The show function is called to display the generated linear plot.

This script serves as a straightforward example of using matplotlib to visualize run scoring trends in a

T20 cricket match, making it suitable for readers interested in representing time-dependent data in a

graphical format.

OUTPUT:

Runs scored

200

175

Run scoring in an T20 Cricket Match

0

0.0

2.5
5.0
7.5
10.0

Overs
12.5 15.0 17.5
20.0

EXPERIMENT 6b

AIM: To write a Python program to illustrate liner plotting with line formatting using

Matplotlib.

PROGRAM:

```
import matplotlib.pyplot as plt

# Hypothetical data: Run rate in an T20 cricket match

overs = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]

runs_scored = [0,7,12,20,39,49,61,83,86,97,113,116, 123,
137,145,163,172,192,198,198,203]

# Create a linear plot
```

```
plt.plot(overs, runs_scored, marker='X', linestyle='dashed',color='red',
linewidth=2,

markerfacecolor='blue', markersize=8)

# Add labels and title

plt.xlabel('Overs', color = 'green')

plt.ylabel('Runs scored')

plt.title('Run scoring in an T20 Cricket Match')

# Display the plot

plt.grid(True)

plt.show()
```

EXPLANATION:

The provided Python script, an extension of the previous T20 cricket match run rate plot, customizes

the appearance of the plot with specific markers, line styles, colors, and label styles. Here's a concise

overview:

Customized Plot Appearance: The plot function is customized with parameters such as marker,

linestyle, color, linewidth, markerfacecolor, and markersize to control the appearance of the plot.

Labels and Title Styling: The script adds labels to the x-axis (Overs) and y-axis (Runs scored) with

specific color styling. The title, 'Run Scoring in a T20 Cricket Match,' maintains clarity.

Grid: The plot includes a grid for better readability.

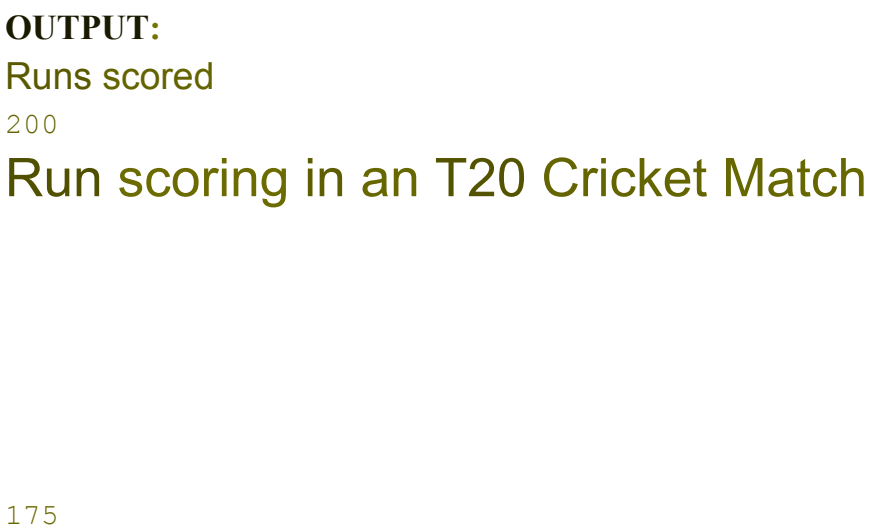
Display: The show function is called to display the generated customized linear plot.

The provided script is an excellent example for those looking to customize plot aesthetics in matplotlib

for a more visually appealing representation of data. It's especially helpful for those interested in

enhancing the clarity and style of their data visualizations.

OUTPUT:


Runs scored
200
Run scoring in an T20 Cricket Match
175
150
125
100
75
*
50

25
*

0

0.0
2.5
5.0
7.5
10.0

Overs
12.5
15.0 17.5 20.0

EXPERIMENT 7

Data Visualization with Python Laboratory - BCS358D

AIM: Write a Python program which explains uses of customizing seaborn plots with Aesthetic

functions.

PROGRAM:

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

def sinplot(n=10):
```

```
x = np.linspace(0, 14, 100)

for i in range(1, n + 1):

plt.plot(x, np.sin(x + i* .5)* (n+2-i))

sns.set_theme()

#sns.set_context("talk")

sns.set_context("notebook", font_scale=1.5, rc={"lines. linewidth": 2.5})

sinplot()

plt.title('Seaborn plots with Aesthetic functions')

plt.show()
```

EXPLANATION:

The provided Python script utilizes the seaborn library, in conjunction with numpy and matplotlib, to

create a series of sine wave plots with customized aesthetics. Here's a concise overview:

Data Generation:

The script uses numpy to generate a series of sine wave plots.

Seaborn Integration:

seaborn is imported and configured with a default theme (set_theme). The context is set to "notebook"

with customized font scaling and line width (set_context).

Customized Aesthetics:

The sinplot function generates multiple sine wave plots with varying frequencies and amplitudes.

Title and Display:

The script adds a title to the plot, 'Seaborn Plots with Aesthetic Functions. The show function is called

to display the generated plots.

This script serves as an illustrative example of how 'seaborn' can be used to enhance the aesthetics of

data visualizations, providing readers with insights into customizing plot styles and themes for more

visually appealing results. It's particularly useful for those looking to leverage 'seaborn' for improved

aesthetics in their data analysis and visualization workflows.

OUTPUT:

10

Seaborn plots with Aesthetic functions

LO

5

0

-5

-10

0.0
2.5
5.0
7.5
10.0
12.5

EXPERIMENT 8

AIM: To write a Python program for plotting different types of plots using Bokeh.

PROGRAM:

```
import numpy as np

from bokeh.layouts import gridplot

from bokeh.plotting import figure, show

x = np.linspace(0, 4* np.pi, 100)

y = np.sin(x)

TOOLS = "pan, wheel_zoom,box_zoom,reset,save, box_select"

pl = figure(title="Example 1", tools=TOOLS)

pl.circle(x, y, legend_label="sin(x)")
```

```
pl.circle(x, 2*y, legend_label="2*sin(x)", color="orange")

pl.circle(x, 3*y, legend_label="3*sin(x)", color="green")

pl.legend.title = 'Markers'

p2 = figure(title="Example 2", tools-TOOLS)

p2.circle(x, y, legend_label="sin(x)")

p2.line(x, y, legend_label="sin(x)")

p2.line(x, 2*y, legend_label="2*sin(x)",

line_dash=(4, 4), line_color="orange", line_width=2)

p2.square(x, 3*y, legend_label="3*sin(x)", fill_color=None,
line_color="green")

p2.line(x, 3*y, legend_label="3*sin(x)", line_color="green")

p2.legend.title = 'Lines'
```

```
show(gridplot([p1, p2], ncols=2, width=400, height=400))
```

EXPLANATION:

The provided Python script demonstrates the use of the Bokeh library to create interactive data

visualizations with multiple plots. Here's a concise overview:

Data Generation: The script generates example data (x and y) using NumPy to represent sine waves.

Interactive Tools: Bokeh's interactive tools (TOOLS) are enabled for features like pan, zoom, reset,

and save.

Multiple Plots: Two separate plots (p1 and p2) are created with different visualizations, including

circles, lines, and markers.

Legend and Titles: Legends are added to distinguish between different elements in the plots. Titles

are provided for each plot.

Grid Layout: The gridplot function is used to arrange the plots in a grid layout.

Interactive Display: The show function is called to display the grid layout, enabling interactive

exploration.

This script serves as an introduction to using Bokeh for creating interactive visualizations with

multiple plots, making it suitable for readers interested in interactive data exploration and visualization

techniques.

OUTPUT:

3

Example 1

2

T

-3

Markers

⚫ sin(x)

2*sin(x)

3*sin(x)

Example 2

3

2

Lines

sin(**x**)

2*sin(x)

3*sin(x)

-3

0
2
6
8
10
12

EXPERIMENT 9

AIM: To write a Python program to draw 3D Plots using Plotly Libraries.

PROGRAM 1:

```
import plotly.graph_objects as go

import numpy as np

x = np.linspace(-5, 5, 100)

y = np.linspace(-5, 5, 100)

x, y = np.meshgrid(x, y)

z = np.sin(np.sqrt(x**2 + y**2))
```

```
fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])

fig.update_layout(scene=dict(









fig.show()
xaxis_title='X Axis',

yaxis_title='Y Axis',

zaxis_title="Z Axis'),

margin=dict(1=0, r=0, b=0, t=40),

title='3D Surface Plot of sin(sqrt(x^2 + y^2))')
```

OR

PROGRAM 2:

```
import plotly.express as px

df = px.data.gapminder().query("continent=='Asia")

fig = px.line_3d(df, x="gdpPercap", y="pop", z="year",
color='country', title='Economic Evolution of

Asian Countries Over Time')

fig.show()
```

EXPLANATION OF PROGRAM 1: Program1 generates a 3D surface plot of the function z =

sin(sqrt(x2+y2)). We can modify the function or provide our own data to create different types of 3D

plots. The visualization will be interactive, allowing **us** to rotate and explore the plot.

EXPLANATION OF PROGRAM 2: In program 2, we leverage the power of Plotly Express to

visualize the economic evolution of Asian countries over time. The dataset used is Gapminder, a

comprehensive collection of global development indicators. The focus is specifically on the Asian

continent.

Import Libraries: We start by importing the necessary libraries, including plotly.express for

interactive visualizations.

Data Loading: We load the Gapminder dataset and filter it to

include only Asian countries.

3D Line Plot: The key visualization is a 3D line plot created using px.line_3d. The x-axis represents

the GDP per capita (gdpPercap), the y-axis represents the population (pop), and the z-axis represents

the year (year). Each line corresponds to a different country, differentiated by color.

Interactive Exploration: The resulting plot is interactive, allowing users to zoom, pan, and hover

over data points to explore specific details.

Users can observe how GDP per capita and population have changed over the years for various Asian

countries. The color-coded lines help distinguish between different nations.

OUTPUT OF PROGRAM 1:

3D Surface Plot of sin(sqrt(x^2+y^2))

4202

XAxis
10
0.5

-0.5

-1

OUTPUT OF PROGRAM 2:

Economic Evolution of Asian Countries Over Time

year

pop
dpPercap
country

Afghanistan Bahrain

-Bangladesh

Cambodia

China

Hong Kong, China

India

Indonesia

EXPERIMENT 10a

Data Visualization with Python Laboratory - BCS358D

AIM: To write a Python program to draw Time Series using Plotly Libraries.

PROGRAM:

```
import pandas as p

import plotly.express as px

dollar_conv = pd.read_csv('CUR_DLR_INR.csv')
```

```
fig = px.line(dollar_conv, x='DATE', y='RATE', title='Dollar vs Rupee')

fig.show()
```

EXPLANATION:

The provided Python script showcases the use of the Plotly Express library to create an interactive line

plot depicting the exchange rate between the US Dollar and the Indian Rupee over time. Here's a

concise overview:

Data Import:

The script uses the Pandas library to read currency conversion data from a CSV file

('CUR_DLR_INR.csv'). You can download the csv file given above.

Plotly Express:

Plotly Express (px) is employed to create an interactive line plot with the exchange rate data.

Line Plot:

The line function from Plotly Express is used to generate a line plot. The x-axis represents dates

('DATE'), and the y-axis represents exchange rates ('RATE').

Title:

The plot is given a title, 'Dollar vs Rupee,' for context.

Interactive Display:

The show method is called on the figure (fig) to display the interactive plot.

This script provides a quick and effective demonstration of using Plotly Express to visualize time-

series data, making it suitable for readers interested in creating interactive and visually appealing line

plots for financial or currency-related datasets.

OUTPUT:

50
70

60
Data Visualization with Python Laboratory - BCS358D

2000
2002
2004
2006

2008
2010
2012
2014
2016

DATE

EXPERIMENT 10b

AIM: To write a Python program for creating Maps using Plotly Libraries.

```
import plotly.express as px

import pandas as pd

data=pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/

gapminder_with_codes.csv')
```

```
fig = px.choropleth(data, locations='iso_alpha', color='gdpPercap',
hover_name='country',
```

```
fig.show()
projection='natural earth', title='GDP per Capita by Country')
```

EXPLANATION:

In this Python program, we utilize Plotly Express to create an interactive choropleth map visualizing

GDP per capita by country. The dataset used is sourced from Gapminder, providing a comprehensive

view of economic indicators globally.

Import Libraries: We start by importing the necessary libraries,

including plotly.express for easy and

interactive visualizations.

Data Loading: The program fetches data from a CSV file hosted on GitHub using pd.read_csv. The

dataset includes information about countries, their ISO codes, and GDP per Capita.

Choropleth Map: The choropleth map is created using px.choropleth.

Key parameters include:

locations: ISO codes of countries.

.

color: GDP per capita, determining the color intensity on the map.

.
hover_name: Country names appearing on hover.

projection: 'natural earth' projection for a global view.

• title: The title of the map.

Interactive Exploration: The resulting choropleth map is interactive, enabling users to hover over

countries to see GDP per Capita values.

Users can explore and compare GDP per Capita across different countries. Darker colors indicate

higher GDP per Capita. This program demonstrates the simplicity and power of Plotly Express for

creating data-driven visualizations. The choropleth map offers an intuitive way to understand global

economic disparities.

OUTPUT:

SAMPLE VIVA QUESTION AND ANSWERS

1. What is data visualization, and why is it important in data analysis?

Data visualization is the graphical representation of data to provide insights and facilitate better

understanding. It is important in data analysis because it allows analysts to identify patterns,

trends, and outliers more easily than through raw data alone.

2. Which Python library is commonly used for data visualization, and what are its main features?

Matplotlib is a commonly used Python library for data visualization. It provides a wide variety

of charts and plots, customization options, and is highly extensible. Other libraries like Seaborn

and Plotly are built on top of Matplotlib, offering additional features.

3. How do you install Matplotlib in Python?

Matplotlib can be installed using the following command: pip install matplotlib

4. Explain the difference between Matplotlib's pyplot and matplotlib.pyplot modules.

pyplot is a collection of functions that make Matplotlib work like MATLAB. matplotlib.pyplot

is the module where most of the Matplotlib functions are defined. The pyplot module is

commonly imported as plt for convenience.

5. What is the purpose of the seaborn library in data visualization?

Seaborn is a statistical data visualization library based on Matplotlib. It provides a high-level

interface for drawing attractive and informative statistical graphics.

Seaborn comes with

several built-in themes and color palettes, making it easy to create aesthetically pleasing

visualizations.

6. What is the purpose of the plotly library, and how is it different from Matplotlib and Seaborn?

Plotly is a Python graphing library that is interactive and suitable for creating web-based

visualizations. Unlike Matplotlib and Seaborn, Plotly produces interactive charts that can be

embedded in web applications, providing a richer user experience.

7. What is Bokeh, and how does it differ from other data visualization libraries like Matplotlib or

Seaborn?

Bokeh is a Python interactive visualization library that targets modern web browsers for

presentation. Unlike Matplotlib or Seaborn, Bokeh is designed for creating interactive plots

that can be embedded in web applications, allowing users to interact with the data dynamically.

8. How can you install Bokeh in Python?

Bokeh can be installed using the following command: pip install bokeh

9. What are the key features of Bokeh that make it suitable for interactive visualizations?

Bokeh provides high-performance interactivity over large or streaming datasets, supports a

wide variety of charts and visualizations, and can create interactive dashboards. It also allows

for easy integration with web frameworks like Flask or Django.

10. How does Bokeh support interactivity in plots?

Bokeh supports interactivity through tools and widgets. Tools allow users to interact with the

plot, such as zooming, panning, or saving the plot. Widgets, on the other hand, are UI

components like sliders or buttons that can be used to update the data or properties of the plot

dynamically.