SRI MANAKULA VINAYAGAR
ENGINEERING COLLEGE
(AN AUTONOMOUS INSTITUTION)
(APPROVED BY AICTE, NEW DELHI AND AFFILIATED TO PONDICHERRY UNIVERSITY)
(ACCREDITED BY NBA-AICTE, NEW DELHI, ACCREDITED BY NAAC WITH "A" GRADE)
MADAGADIPET, PUDUCHERRY - 605 107

Register Number :

Name :

Subject Name / Subject Code :

Branch :

Year / Semester :

Certificate

Certified that this is the bonafide record of Practical work done by the above student in the …… Laboratory during the academic year ……

Staff in-charge **Head of the Department**

Submitted for the End Semester practical Examination held on……

Internal Examiner **External Examiner**

TABLE OF CONTENTS

S.No.	Date	Title	Page No	Mark	Sign
1		Build a python program to implement Fibonacci series.	1		
2		Build a python program to get a range of numbers from user and to separate even numbers and odd numbers respectively.	4		
3		Build a function in Python to check duplicate letters. It must accept a string, i.e., a sentence. The function should return True if the sentence has any word with duplicate letters, else return False.	7		
4		Build a program to perform arithmetic operations using lambda function.	10		
5		Build a Python program that takes a list of numbers as input and returns a new list containing only the even numbers from the input list.	13		
6		Build a python program to create a class called Car with attributes Company, model, and year. Implement a method that returns the age of the car in years.	16		
7		Build a python program to create a base class called Shape that has a method called area which returns the area of the shape (set it to 0 for now). Then, create two derived classes Rectangle and Circle that inherit from the Shape class to calculate the area of derived classes.	20		
8		Build a python program to implement aggregation using Numpy.	23		
9		Build a python program to perform Indexing and Sorting.	26		

10		Build a python program to perform Handling of missing data.	29		
11		Build a python program to perform usage of Pivot table using Titanic datasets.	32		
12		Build a python program to perform use of eval() and query().	35		
13		Build a python program to perform Scatter Plot.	45		
14		Build a python program to perform 3D plotting.	52		
15		Implement of analytic application	60		
16		Implement an application to process a real time data.	67		

EX.NO: 1 **Fibonacci Series**

DATE:

Aim:

To write and implement a Python program to generate the Fibonacci series up to a given number of terms.

Algorithm:

1. Start the program.
2. Prompt the user to input the number of terms (n) to generate in the Fibonacci sequence.
3. Initialize two variables a = 0 and b = 1 to represent the first two numbers in the sequence.
4. Print the initial two numbers of the sequence.
5. Use a for loop starting from 2 up to n (since the first two terms are already printed).
6. Inside the loop, calculate the next number as c = a + b.
7. Print the calculated value of c.
8. Update the values of a and b for the next iteration (a = b, b = c).
9. Continue the loop until all n terms are generated.
10. End the program.

Program:

```
n = int(input("Enter the number of terms: "))
a, b = 0, 1
print("Fibonacci series:")
if n == 1:
    print(a)
else:
    print(a, b, end=" ")
    for _ in range(2, n):
        c = a + b
        print(c, end=" ")
        a, b = b, c
```

Output:

Sample1 :
Enter the number of terms: 5
Fibonacci series:
0 1 1 2 3

Sample2:
Enter the number of terms: 8
Fibonacci series:
0 1 1 2 3 5 8 13

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to Fibonacci series up to the given number of terms is executed and the output is verified.

EX.NO: 2 **Separate Even and Odd Numbers**

DATE:

Aim:

To write and implement a Python program that accepts a range of numbers from the user and separates even and odd numbers respectively.

Algorithm:

1. Start the program.
2. Prompt the user to input the starting value of the range and store it in variable start.
3. Prompt the user to input the ending value of the range and store it in variable end.
4. Initialize two empty lists: even_numbers and odd_numbers.
5. Use a for loop to iterate over all numbers from start to end (inclusive).
6. Inside the loop, check if the current number is divisible by 2.
7. If it is divisible by 2, append it to the even_numbers list.
8. If not, append it to the odd_numbers list.
9. After the loop ends, print the list of even and odd numbers.
10. End the program.

Program:

```
start = int(input("Enter the start of the range: "))
end = int(input("Enter the end of the range: "))

even_numbers = []
odd_numbers = []

for num in range(start, end + 1):
    if num % 2 == 0:
        even_numbers.append(num)
    else:
        odd_numbers.append(num)

print("Even numbers:", even_numbers)
print("Odd numbers:", odd_numbers)
```

Output:

Sample1 :
Enter the start of the range: 1
Enter the end of the range: 10
Even numbers: [2, 4, 6, 8, 10]
Odd numbers: [1, 3, 5, 7, 9]

Sample2 :
Enter the start of the range: 11
Enter the end of the range: 15
Even numbers: [12, 14]
Odd numbers: [11, 13, 15]

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program separates even and odd numbers from a given numeric range is executed and the output is verified.

EX.NO: 3 Check for Duplicate Letters

DATE:

Aim:

To write and implement a Python function that accepts a sentence as input and returns True if any word contains duplicate letters; otherwise, it returns False.

Algorithm:

1. Start the program.
2. Define a function has_duplicate_letters(sentence) that accepts a string as an argument.
3. Split the input sentence into a list of words using the split() method.
4. For each word in the list:
5. a. Convert the word to lowercase to ensure uniformity.
6. b. Create a set to store letters encountered.
7. c. Iterate through each letter in the word.
8. d. If the letter already exists in the set, return True.
9. e. If not, add the letter to the set.
10. If no word contains duplicate letters, return False.
11. Prompt the user to enter a sentence.
12. Call the function and display the result.
13. End the program.

Program:

```
def has_duplicate_letters(sentence):
    words = sentence.split()
    for word in words:
        seen_letters = set()
        for letter in word.lower():
            if letter in seen_letters:
                return True
            seen_letters.add(letter)
    return False

sentence = input("Enter a sentence: ")
result = has_duplicate_letters(sentence)
print("Contains duplicate letters in a word:", result)
```

Output:

Sample1:
Enter a sentence: Hello world
Contains duplicate letters in a word: True

Sample2:
Enter a sentence: Python is fun
Contains duplicate letters in a word: False

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program detects whether any word in the input sentence contains duplicate letters is executed and the output is verified.

EX.NO: 4 Arithmetic Operations using Lambda Functions

DATE:

Aim:

To write and implement a Python program that performs basic arithmetic operations (addition, subtraction, multiplication, division) using lambda functions.

Algorithm:

1. Start the program.
2. Define lambda functions for the four arithmetic operations:
   a. add = lambda x, y: x + y
   b. subtract = lambda x, y: x - y
   c. multiply = lambda x, y: x * y
   d. divide = lambda x, y: x / y
3. Prompt the user to input two numbers and store them in variables a and b.
4. Use the lambda functions to perform the operations on the input numbers.
5. Display the results of all operations.
6. Handle division by zero using a conditional check before division.
7. End the program.

Program:

```
add = lambda x, y: x + y
subtract = lambda x, y: x - y
multiply = lambda x, y: x * y
divide = lambda x, y: x / y if y != 0 else "Undefined (division by zero)"

a = float(input("Enter the first number: "))
b = float(input("Enter the second number: "))

print("Addition:", add(a, b))
print("Subtraction:", subtract(a, b))
print("Multiplication:", multiply(a, b))
print("Division:", divide(a, b))
```

Output:

```
Sample1:
Enter the first number: 12
Enter the second number: 4
Addition: 16.0
Subtraction: 8.0
Multiplication: 48.0
Division: 3.0

Sample2:
Enter the first number: 7
Enter the second number: 0
Addition: 7.0
Subtraction: 7.0
Multiplication: 0.0
Division: Undefined (division by zero)
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to performs arithmetic operations using lambda functions, including safe division handlingis executed and the output is verified.

EX.NO: 5 **Filter Even Numbers from a List**

DATE:

Aim:

To write and implement a Python program that accepts a list of numbers and returns a new list containing only the even numbers.

Algorithm:

1. Start the program.
2. Prompt the user to enter a list of numbers separated by spaces.
3. Convert the input string into a list of integers using map() and split().
4. Use a for loop or list comprehension to filter out even numbers from the list.
5. Alternatively, use the filter() function with a lambda expression to extract even numbers.
6. Store the result in a new list named even_numbers.
7. Print the list of even numbers.
8. Ensure the input list can be of any length.
9. Handle both positive and negative numbers if provided.
10. End the program.

Program:

```
#Use built in function

from functools import reduce

numbers = list(map(int, input("Enter numbers separated by space: ").split()))
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
even_count = reduce(lambda acc, x: acc + 1, even_numbers, 0)

print("Even numbers:", even_numbers)
print("Count of even numbers:", even_count)



# without use built function :

numbers = list(map(int, input("Enter numbers separated by space: ").split()))
even_numbers = []
count = 0

for num in numbers:
    if num % 2 == 0:
        even_numbers.append(num)
        count += 1

print("Even numbers:", even_numbers)
print("Count of even numbers:", count)
```

Output:

```
Sample 1:
Enter numbers separated by space: 2 5 8 11 14 17
Even numbers: [2, 8, 14].


Sample 2:
Enter numbers separated by space: 1 3 5 7
Even numbers: []
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the given program to filters and displays only the even numbers from the input list using a lambda function is executed and output is verified.

EX NO: 6 **Car Class with Age Calculation**

DATE:

Aim:

To write and implement a Python program to create a class called Car with attributes company, model, and year. Implement a method to return the age of the car in years.

Algorithm:

1. Start the program.
2. Import the datetime module to get the current year.
3. Define a class Car with an __init__method that accepts and stores company, model, and year.
4. Define a method get_age() in the class to calculate the age of the car.
5. In get_age(), subtract the car's manufacturing year from the current year.
6. Create an object of the Car class with the values entered by the user.
7. Call the get_age() method on the object and print the result.
8. Allow multiple test cases by creating different objects.
9. Ensure proper data types (e.g., convert year to integer).
10. End the program.

Program :

```
from datetime import datetime

class Car:

    def __init__(self, company, model, year):

        self.company = company

        self.model = model

        self.year = int(year)

    def get_age(self):

        current_year = datetime.now().year

        return current_year - self.year

# Test Case 1

company1 = input("Enter car company: ")

model1 = input("Enter car model: ")

year1 = input("Enter manufacturing year: ")

car1 = Car(company1, model1, year1)

print(f"The {car1.company} {car1.model} is {car1.get_age()} years old.")

# Test Case 2

company2 = input("\nEnter car company: ")
```

```
model2 = input("Enter car model: ")

year2 = input("Enter manufacturing year: ")

car2 = Car(company2, model2, year2)

print(f"The {car2.company} {car2.model} is {car2.get_age()} years old.")
```

Output :

Sample1 :

Enter car company: Toyota

Enter car model: Camry

Enter manufacturing year: 2015

The Toyota Camry is 10 years old.

Sample 2:

Enter car company: Honda

Enter car model: City

Enter manufacturing year: 2020

The Honda City is 5 years old.

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program accurately calculates and displays the age of the car using a class and instance method function was executed and output is verified.

EX.NO: 7 **Inheritance with Shape, Rectangle, and Circle Classes**

DATE:

Aim:

To write and implement a Python program that defines a base class Shape with an area() method, and two derived classes Rectangle and Circle that override the method to calculate and return their respective areas.

Algorithm:

1. Start the program.
2. Import the math module to access pi for circle area calculation.
3. Define a base class Shape with a method area() that returns 0.
4. Define a subclass Rectangle that inherits from Shape.
    a. Accept length and breadth as parameters.
    b. Override the area() method to return length * breadth.
5. Define another subclass Circle that also inherits from Shape.
    a. Accept radius as a parameter.
    b. Override the area() method to return π * radius^2.
6. In the main block, prompt the user for inputs for both Rectangle and Circle.
7. Create objects for both classes and call their area() methods.
8. Display the calculated areas.
9. Allow testing with two different sets of inputs.
10. End the program.

Program:

```
import math

class Shape:
    def area(self):
        return 0

class Rectangle(Shape):
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth

    def area(self):
        return self.length * self.breadth

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

# Test Case 1
length = float(input("Enter rectangle length: "))
breadth = float(input("Enter rectangle breadth: "))
rect = Rectangle(length, breadth)
print("Area of Rectangle:", rect.area())

radius = float(input("\nEnter circle radius: "))
circle = Circle(radius)
print("Area of Circle:", round(circle.area(), 2))

# Test Case 2
length2 = float(input("\nEnter another rectangle length: "))
breadth2 = float(input("Enter another rectangle breadth: "))
rect2 = Rectangle(length2, breadth2)
print("Area of Rectangle:", rect2.area())

radius2 = float(input("\nEnter another circle radius: "))
circle2 = Circle(radius2)
print("Area of Circle:", round(circle2.area(), 2))
```

Output :

```
Enter rectangle length: 10
Enter rectangle breadth: 5
Area of Rectangle: 50.0

Enter circle radius: 7
Area of Circle: 153.94
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to demonstrates inheritance by creating a base class and overriding its method in derived classes to calculate and return specific areas executed and output is verified.

EX.NO: 8 **Aggregation using NumPy**
DATE:

Aim:

To write and implement a Python program using NumPy to perform aggregation functions such as sum, mean, max, min, standard deviation, and variance on an array of numbers.

Algorithm:

1. Start the program.
2. Import the NumPy library as np.
3. Prompt the user to enter a sequence of numbers separated by spaces.
4. Convert the input into a NumPy array using np.array() and map().
5. Use NumPy functions to calculate:
    a. Sum → np.sum()
    b. Mean → np.mean()
    c. Maximum → np.max()
    d. Minimum → np.min()
    e. Standard Deviation → np.std()
    f. Variance → np.var()
6. Display all the aggregation results.
7. Repeat the process with a second test case.
8. End the program.

Program:

```
import numpy as np

# Test Case 1

data1 = np.array(list(map(float, input("Enter numbers (Test Case 1): ").split())))
print("Sum:", np.sum(data1))
print("Mean:", np.mean(data1))
print("Max:", np.max(data1))
print("Min:", np.min(data1))
print("Standard Deviation:", round(np.std(data1), 2))
print("Variance:", round(np.var(data1), 2))

# Test Case 2
data2 = np.array(list(map(float, input("\nEnter numbers (Test Case 2): ").split())))
print("Sum:", np.sum(data2))
print("Mean:", np.mean(data2))
print("Max:", np.max(data2))
print("Min:", np.min(data2))
print("Standard Deviation:", round(np.std(data2), 2))
print("Variance:", round(np.var(data2), 2))
```

Output test case 1:

```
Enter numbers (Test Case 1): 10 20 30 40 50
Sum: 150.0
Mean: 30.0
Max: 50.0
Min: 10.0
Standard Deviation: 14.14
Variance: 200.0
```

Output test case 2:

```
Enter numbers (Test Case 2): 5 5 5 5 5
Sum: 25.0
Mean: 5.0
Max: 5.0
Min: 5.0
Standard Deviation: 0.0
Variance: 0.0
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

The program effectively uses NumPy's aggregation functions to analyze numerical data arrays.

EX.NO: 9 INDEXING AND SORTING IN NUMPY

DATE:

Algorithm:

1. Start The Program
2. Import the NumPy library as np.
3. Accept a sequence of numbers from the user and convert it into a NumPy array.At first,we are using set_index().
4. Demonstrate **indexing** by accessing:
   a. First element
   b. Last element
   c. A slice of elements (e.g., first 3 or middle section)Using iloc() we are performing the indexing in the given table
5. Demonstrate **sorting** using np.sort() to sort the array in ascending order.
6. Repeat the above steps for a second test case.
7. Print both the original and sorted arrays.
8. End the program.

Program:

```
import numpy as np

# Test Case 1
arr1 = np.array(list(map(int, input("Enter numbers (Test Case 1): ").split())))
print("Original Array:", arr1)
print("First Element:", arr1[0])
print("Last Element:", arr1[-1])
print("Slice (First 3):", arr1[:3])
sorted_arr1 = np.sort(arr1)
print("Sorted Array:", sorted_arr1)

# Test Case 2
arr2 = np.array(list(map(int, input("\nEnter numbers (Test Case 2): ").split())))
print("Original Array:", arr2)
print("First Element:", arr2[0])
print("Last Element:", arr2[-1])
print("Slice (Middle):", arr2[1:-1])
sorted_arr2 = np.sort(arr2)
print("Sorted Array:", sorted_arr2)
```

Output test case 1:

```
Enter numbers (Test Case 1): 45 12 89 33 7
Original Array: [45 12 89 33  7]
First Element: 45
Last Element: 7
Slice (First 3): [45 12 89]
Sorted Array: [ 7 12 33 45 89]
```

```
Enter numbers (Test Case 2): 9 3 6 2 1
Original Array: [9 3 6 2 1]
First Element: 9
Last Element: 1
Slice (Middle): [3 6 2]
Sorted Array: [1 2 3 6 9]
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

The program demonstrates indexing (accessing elements and slices) and sorting using NumPy functions

EX.NO: 10 HANDLING MISSING DATA IN PYTHON

DATE:

Aim:

To write and implement a Python program using Pandas to detect, handle, and process missing data in a dataset.

Algorithm:

1. Start the program.
2. Import the pandas library as pd.
3. Create a DataFrame manually with some missing (NaN) values using np.nan.
4. Display the original DataFrame.
5. Detect missing data using isnull() and sum() functions.
6. Fill missing values using fillna() (e.g., replace with mean or a specific value).
7. Drop rows with missing values using dropna().
8. Repeat the process with a second dataset (Test Case 2).
9. Display the cleaned DataFrames.
10. End the program.

Program:

```
import pandas as pd
import numpy as np

# Test Case 1
data1 = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David'],
   'Age': [25, np.nan, 30, np.nan],
   'Score': [85, 90, np.nan, 88]
}
df1 = pd.DataFrame(data1)
print("Original DataFrame (Test Case 1):\n", df1)

# Handling missing data
print("\nMissing Values Count:\n", df1.isnull().sum())
df1_filled = df1.fillna(df1.mean(numeric_only=True))
print("\nAfter Filling Missing Values:\n", df1_filled)

df1_dropped = df1.dropna()
print("\nAfter Dropping Rows with Missing Values:\n", df1_dropped)

# Test Case 2
data2 = {
   'Product': ['A', 'B', 'C', 'D'],
   'Price': [100, np.nan, 150, 120],
   'Stock': [np.nan, 50, 70, np.nan]
}
df2 = pd.DataFrame(data2)
print("\n\nOriginal DataFrame (Test Case 2):\n", df2)

print("\nMissing Values Count:\n", df2.isnull().sum())
df2_filled = df2.fillna({'Price': df2['Price'].mean(), 'Stock': 0})
print("\nAfter Filling Missing Values:\n", df2_filled)

df2_dropped = df2.dropna()
print("\nAfter Dropping Rows with Missing Values:\n", df2_dropped)
```

Output:

```
Original DataFrame (Test Case 1):
      Name   Age  Score
0   Alice  25.0   85.0
1     Bob   NaN   90.0
2 Charlie  30.0    NaN
3   David   NaN   88.0
```

```
Missing Values Count:
 Name     0
Age      2
Score    1
dtype: int64

After Filling Missing Values:
      Name   Age  Score
0   Alice  25.0   85.0
1     Bob  27.5   90.0
2 Charlie  30.0   87.6667
3   David  27.5   88.0

After Dropping Rows with Missing Values:
    Name   Age  Score
0  Alice  25.0   85
```

Criteria	**Maximum Marks**	**Marks Obtained**
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to detects and handles missing data using both value filling and Row dropping techniques.was executed and output is verified.

EX.NO: 11 **Usage of Pivot Table using Titanic Dataset**

DATE:

Aim:

To write and implement a Python program that demonstrates the use of pivot tables using the Titanic dataset with the help of the Pandas library.

Algorithm:

1. Start the program.
2. Import pandas as pd and seaborn as sns (Titanic data is available via seaborn).
3. Load the Titanic dataset using sns.load_dataset("titanic").
4. Display the first few rows of the dataset using head().
5. Create pivot tables using pd.pivot_table() with various combinations:
    a. Average age by gender.
    b. Survival rate by class and gender.
    c. Average fare by class and embark_town.
6. Print the resulting pivot tables.
7. Repeat the pivot operation with different columns for a second test case.
8. End the program.

Program:

```
import pandas as pd
import seaborn as sns

# Load Titanic dataset
df = sns.load_dataset("titanic")
print("First 5 rows of Titanic Dataset:\n", df.head())

# Test Case 1 - Pivot Table Examples
print("\nAverage Age by Gender:")
print(pd.pivot_table(df, values='age', index='sex', aggfunc='mean'))

print("\nSurvival Rate by Class and Gender:")
print(pd.pivot_table(df, values='survived', index='class', columns='sex', aggfunc='mean'))

# Test Case 2 - Additional Pivot Examples
print("\nAverage Fare by Class and Embark Town:")
print(pd.pivot_table(df, values='fare', index='class', columns='embark_town', aggfunc='mean'))

print("\nCount of Passengers by Embark Town and Class:")
print(pd.pivot_table(df, values='survived', index='embark_town', columns='class', aggfunc='count'))
```

Output :

```
First 5 rows of Titanic Dataset:
   survived  pclass     sex   age  ...  who  deck  embark_town  alive
0         0       3    male  22.0  ...  man   NaN  Southampton     no
1         1       1  female  38.0  ...  woman   C    Cherbourg    yes



Average Age by Gender:
sex
female    27.915709
male      30.726645

Survival Rate by Class and Gender:
sex      female      male
class
First   0.968085  0.368852
Second  0.921053  0.157407
Third   0.500000  0.135447
```

Average Fare by Class and Embark Town:

class	embark_town Cherbourg	Queenstown	Southampton
First	104.7186	NaN	82.4084
Second	24.6873	13.2760	20.3271
Third	14.9503	9.0729	13.9145

Count of Passengers by Embark Town and Class:

embark_town	class First	Second	Third
Cherbourg	85	17	66
Queenstown	2	3	77
Southampton	127	164	353

Criteria	**Maximum Marks**	**Marks Obtained**
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to demonstrates how pivot tables can be used to summarize, analyze, and extract insights from the Titanic dataset effectively was executed and the output is Verified.

EX.NO: 12

DATE:

USE OF EVAL() AND QUERY()

Aim:

To create a user defined dataframe in pandas and to perform theeval() and query() operations in python.

Algorithm:

1. Start the program
2. install and import pandas Packages to python.

   !pip install pandas

   import pandas as pd
3. install and import numpy Packages to python.

   !pip install numpy

   import numpy as np
4. Create DataFrames of random values

   rng = np.random.RandomState(42)

   df1, df2, df3, df4 = (pd.DataFrame(rng.rand(nrows, ncols))for i in range(4))
5. Use and implement eval() function.EVAL()

   The eval() function is used to evaluate an expression in the context of the calling dataframe instance. The expression is evaluated over the columns of the dataframe.
6. Use and implement query() function QUERY()

   The query() function is used to query the columns of a DataFrame with aboolean expression.
7. Stop the program

Efficient Operations:

The eval() function in Pandas uses string expressions to efficiently compute operations using DataFrames. For example, consider the following DataFrames

```
import pandas as pd
import numpy as np
nrows, ncols = 100000, 100
rng = np.random.RandomState(42)
df1, df2, df3, df4 = (pd.DataFrame(rng.rand(nrows, ncols))for i in range(4))
```

To compute the sum of all four DataFrames using the typical Pandas approach, we can just write the sum:

```
%timeit df1 + df2 + df3 + df4
```

40.1 ms ± 484 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

The same result can be computed via pd.eval by constructing the expression as string

```
%timeit pd.eval('df1 + df2 + df3 + df4')
```

10 loops, best of 3: 42.2 ms per loop

The eval() version of this expression is about 50% faster (and uses muchless memory), while giving the same result:

```
np.allclose(df1 + df2 + df3 + df4,
pd.eval('df1 + df2 + df3 + df4'))
```

True

Operations supported by pd.eval():

As of Pandas v0.16, pd.eval() supports a wide range of operations. To demonstrate these, we'll use the following integer DataFrames

```
df1, df2, df3, df4, df5 = (pd.DataFrame(rng.randint(0, 1000, (100, 3)))
                           for i in range(5))
```

Arithmetic operators:

pd.eval() supports all arithmetic operators. For example:

```
result1 = -df1 * df2 / (df3 + df4) - df5
result2 = pd.eval('-df1 * df2 / (df3 + df4) - df5')
np.allclose(result1, result2)
```

True

Comparison operators:

pd.eval() supports all comparison operators, including chained expressions:

```
result1 = (df1 < df2) & (df2 <= df3) & (df3 != df4)
result2 = pd.eval('df1 < df2 <= df3 != df4')
np.allclose(result1, result2)
```

True

pd.eval() supports the & and | bitwise operators:

```
result1 = (df1 < 0.5) & (df2 < 0.5) | (df3 < df4)

result2 = pd.eval('(df1 < 0.5) & (df2 < 0.5) | (df3 < df4)')np.allclose(result1,

result2)
```

True

In addition, it supports the use of the literal and and or in Boolean expressions:

```
result3 = pd.eval('(df1 < 0.5) and (df2 < 0.5) or (df3 < df4)')

np.allclose(result1, result3)
```

True

Other operations:

Other operations such as function calls, conditional statements, loops, and other more involved constructs are currently *not* implemented in pd.eval(). If you'd like to execute these more complicated types of expressions, you can use the Numexpr library itself.

DataFrame.eval() for Column-Wise Operations:

Just as Pandas has a top-level pd.eval() function, DataFrames have an eval() method that works in similar ways. The benefit of the eval() method is that columns can be referred to *by* name. We'll use this labeled array as an example:

```
df = pd.DataFrame(rng.rand(1000, 3), columns=['A', 'B', 'C'])
df.head()
```

	A	B	C
0	0.375506	0.406939	0.069938
1	0.069087	0.235615	0.154374
2	0.677945	0.433839	0.652324
3	0.264038	0.808055	0.347197
4	0.589161	0.252418	0.557789

Using pd.eval() as above, we can compute expressions with the three columns like this:

True

```
result1 = (df['A'] + df['B']) / (df['C'] - 1)
result2 = pd.eval("(df.A + df.B) / (df.C - 1)")
np.allclose(result1, result2)
```

The DataFrame.eval() method allows much more succinct evaluation of expressions with thecolumns:

```
result3 = df.eval('(A + B) / (C - 1)')
np.allclose(result1, result3)
```

True

Assignment in DataFrame.eval():

In addition to the options just discussed, DataFrame.eval() also allows assignment to any column. Let's use the DataFrame from before,which has columns 'A', 'B', and 'C':

```
df.head()
```

	A	B	C
0	0.375506	0.406939	0.069938
1	0.069087	0.235615	0.154374
2	0.677945	0.433839	0.652324
3	0.264038	0.808055	0.347197
4	0.589161	0.252418	0.557789

We can use df.eval() to create a new column 'D' and assign to it a value computed from the other columns:

```
df.eval('D = (A + B) / C', inplace=True)
df.head()
```

	A	B	C	D
0	0.375506	0.406939	0.069938	11.187620
1	0.069087	0.235615	0.154374	1.973796
2	0.677945	0.433839	0.652324	1.704344
3	0.264038	0.808055	0.347197	3.087857
4	0.589161	0.252418	0.557789	1.508776

In the same way, any existing column can be modified:

```
df.eval('D = (A - B) / C', inplace=True)
df.head()
```

	A	B	C	D
0	0.375506	0.406939	0.069938	-0.449425
1	0.069087	0.235615	0.154374	-1.078728
2	0.677945	0.433839	0.652324	0.374209
3	0.264038	0.808055	0.347197	-1.566886
4	0.589161	0.252418	0.557789	0.603708

Local variables in DataFrame.eval():

The DataFrame.eval() method supports an additional syntax that lets it work with localPython variables. Consider the following:

```
column_mean = df.mean(1)

result1 = df['A'] + column_mean result2 =

df.eval('A + @column_mean')np.allclose(result1,

result2)
```

True

The @ character here marks a *variable name* rather than a *column name*,and lets you efficiently evaluate expressions involving the two "namespaces": the namespace of columns, and the namespace of Python objects. Notice that this @ character is only supported by the DataFrame.eval() *method*, not by the pandas.eval() *function*, because the pandas.eval() function only has access to the one (Python) namespace.

DataFrame.query() Method:

The DataFrame has another method based on evaluated strings,called the query() method. Consider the following:

```
result1 = df[(df.A < 0.5) & (df.B < 0.5)]

result2 = pd.eval('df[(df.A < 0.5) & (df.B < 0.5)]')

np.allclose(result1, result2)
```

True

As with the example used in our discussion of DataFrame.eval(), this is an expression involving columns of the DataFrame. It cannot be expressed using the DataFrame.eval() syntax, however! Instead, for this type of filtering operation, you can use the query() method:

```
result2 = df.query('A < 0.5 and B < 0.5')

np.allclose(result1, result2)
```

True

In addition to being a more efficient computation, compared to the masking expression this is much easier to read and understand. Note that the query() method also accepts the @ flag to mark local variables:

```
Cmean = df['C'].mean()

result1 = df[(df.A < Cmean) & (df.B < Cmean)] result2 = df.query('A

< @Cmean and B < @Cmean')np.allclose(result1, result2)
```

True

Performance: When to Use These Functions:

When considering whether to use these functions, there are two considerations: *computation time* and *memory use*. Memory use is the most predictable aspect. As already mentioned, every compound expression involving NumPy arrays or Pandas DataFrames will result in implicit creation of temporary arrays: For example, this:

```
x = df[(df.A < 0.5) & (df.B < 0.5)]
```

Is roughly equivalent to this:

```
tmp1 = df.A < 0.5

tmp2 = df.B < 0.5

tmp3 = tmp1 & tmp2

x = df[tmp3]
```

Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to implement eval() and query() was executed and the output is Verified.

EX.NO: 13

SCATTER PLOT

DATE:

Aim:

To create a user defined numpy array and visualize the data in scatter using matplotlib in python.

Algorithm:

1. Start the program
2. import matplot library package using

   matplotlib.pyplot as plt
3. import numpy package using

   import numpy as np
4. To work with backend we use magic comment that is %matplotlib inline
5. Then create a numpy array and then generate random values for x and y with the help of the numpy array.
6. With these generated values,

   We can plot in graph using plt.plot(x,y)
7. We can label the x and y axis using plt.xlabel() .
8. We can also limit the x and y axis values to such limits using

   plt.xlim(0,1.8)
9. Some of the properties we used here are:

   legend, colors, markersize, linewidth
10. Use plt.colorbar() to show color scale
11. Stop the program.

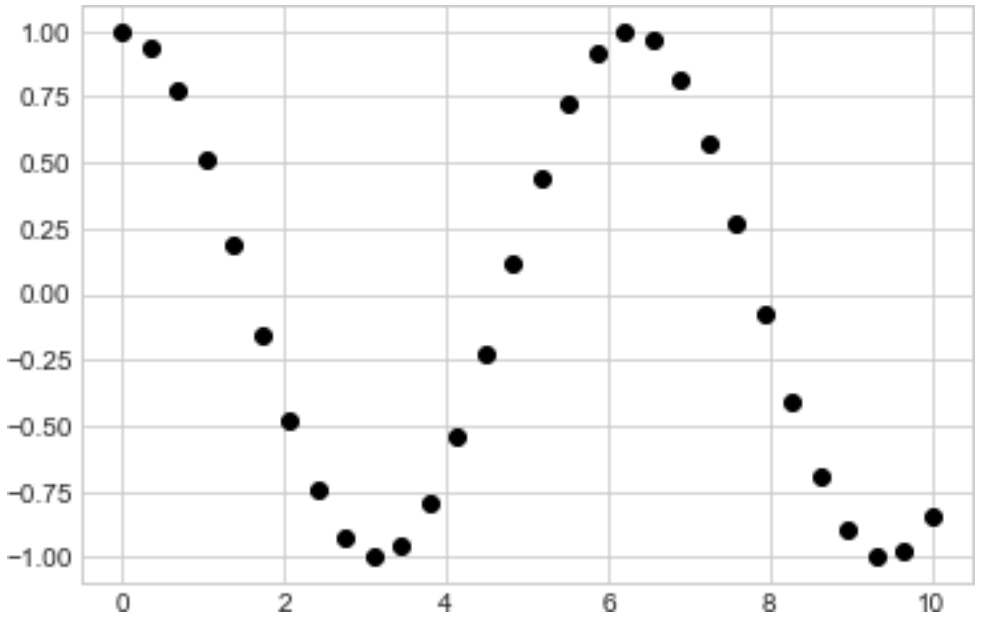
Simple scatter Plot:

The simple scatter plot, a close cousin of the line plot. Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape. We'll start by setting up the notebook for plotting and importing the functions we will use:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

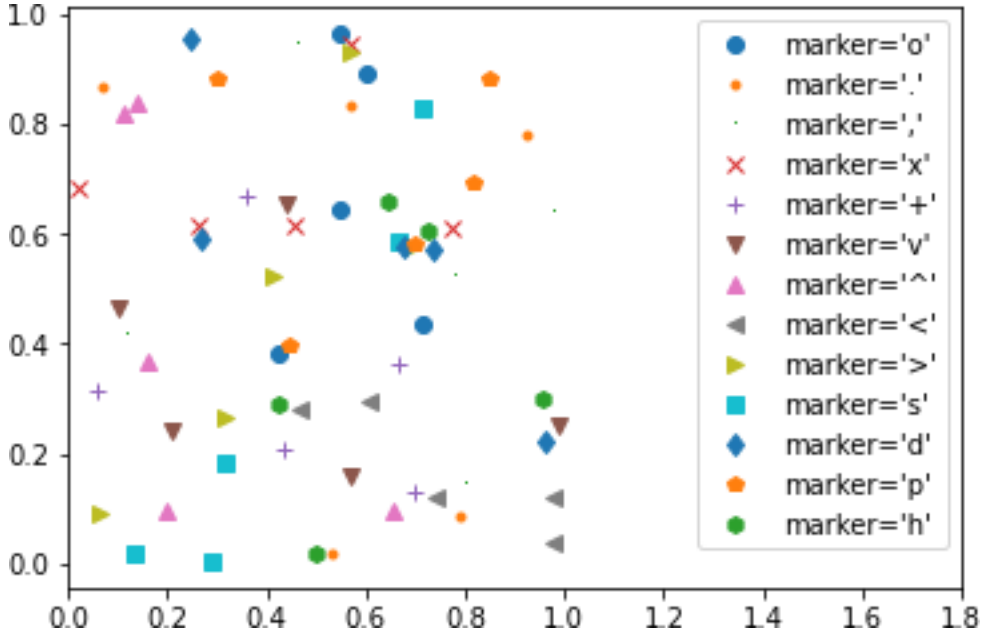
Scatter plot with *plt.plot*:

```
x = np.linspace(0, 10, 30)
y = np.cos(x)
plt.plot(x, y, 'o', color = 'black');
```



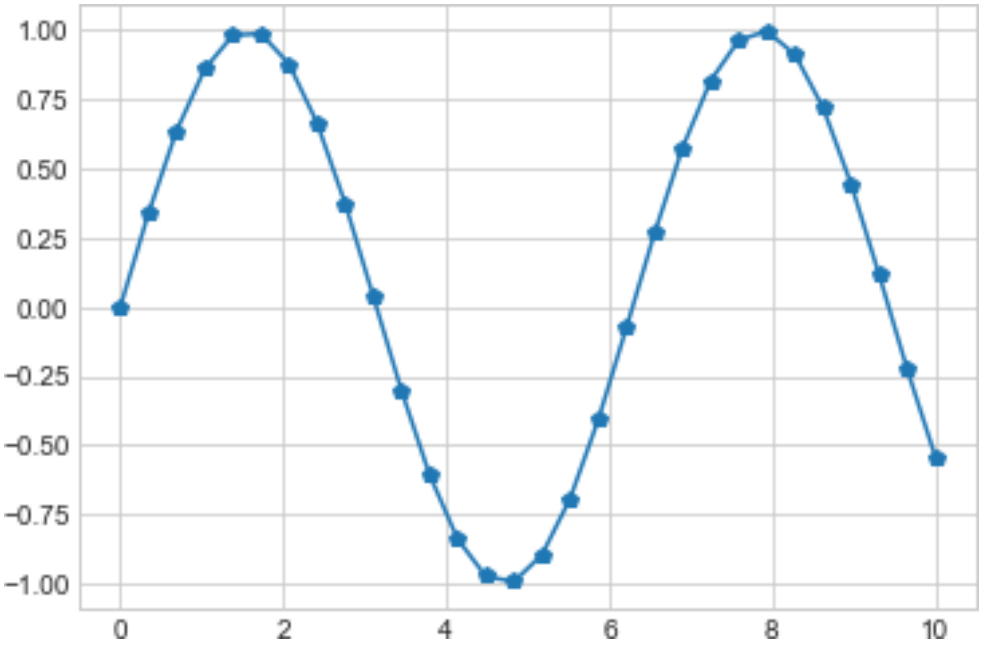
The third argument in the function call is a character that represents the type of symbol used for the plotting. Just as you can specify options such as '-', '--' to control the line style, the marker style has its own set of short string codes. The full list of available symbols can be seen in the documentation of plt.plot

```
rng = np.random.RandomState(0)
for marker in [‘o’,’.’,’,’,’x’,’+’,’^’,’<’,’>’,’s’,’d’,’p’,’h’]:
    plt.plot(rng.rand(5)rng.rand(5),marker, label=”marker=’{0}’”.fomat(marker))
    plt.legend(numpoints = 1)
plt.xlim(0, 1.8);
```



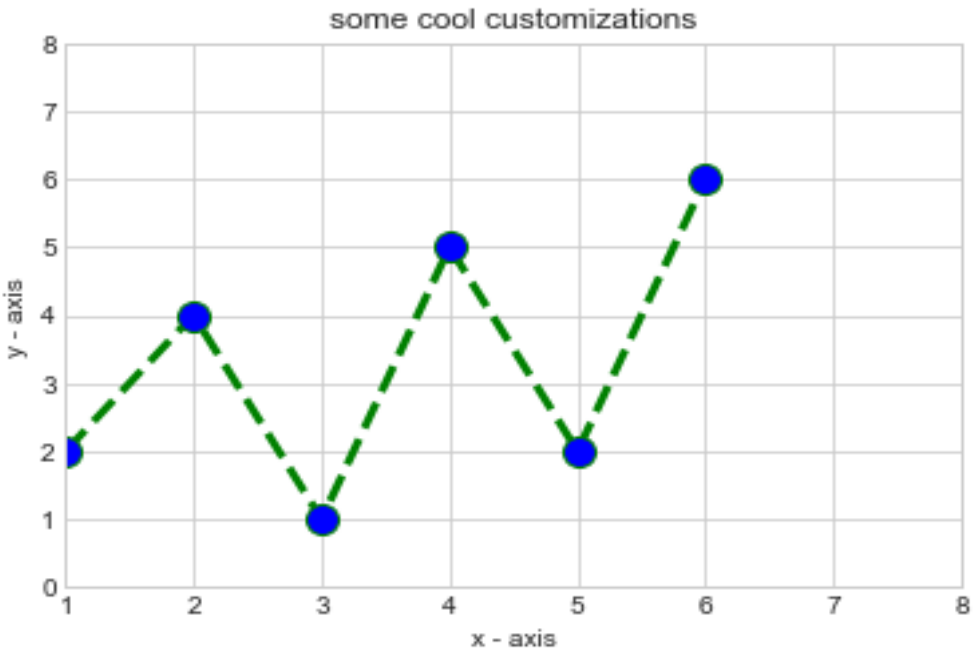
For even more possibilities, these character codes can be used together with line and color codes to plot points along with a line connecting them:

```
plt.plot(x, y, '-p');
```

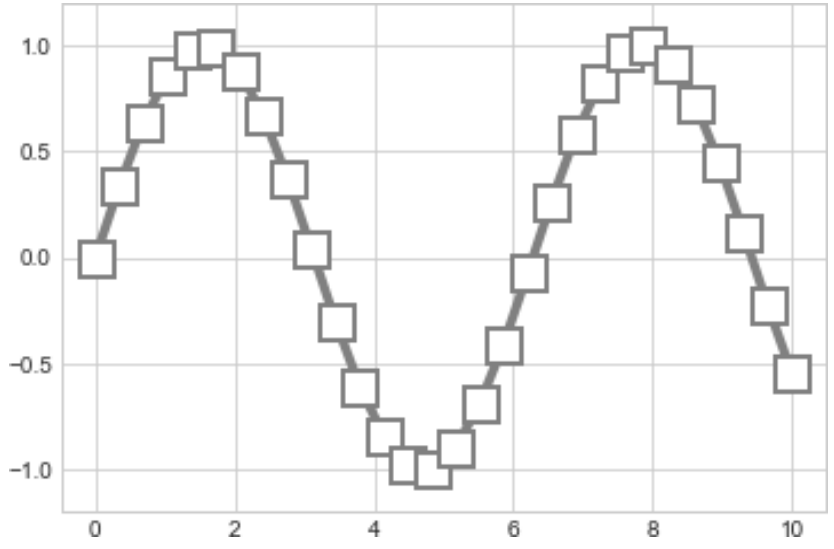


Additional keyword arguments to plt.plot specify a wide range of properties of the lines and markers. This type of flexibility in the plt.plotfunction allows for a wide variety of possible visualization options.

```
import matplotlib.pyplot as plt
  # x axis values
x = [1,2,3,4,5,6]
# corresponding y axis values
y = [2,4,1,5,2,6]

# plotting the points
plt.plot(x, y, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)

plt.ylim(1,8)
plt.xlim(1,8)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Some cool customizations!')

# function to show the plot
plt.show()
```




```
plt.plot(x, y, '-s', color = 'gray',
         markersize = 15, linewidth = 4,
         markerfacecolor = 'white',
         markeredgecolor = 'gray',
         markeredgewidth = 2)
plt.ylim(-1.2, 1.2);
```

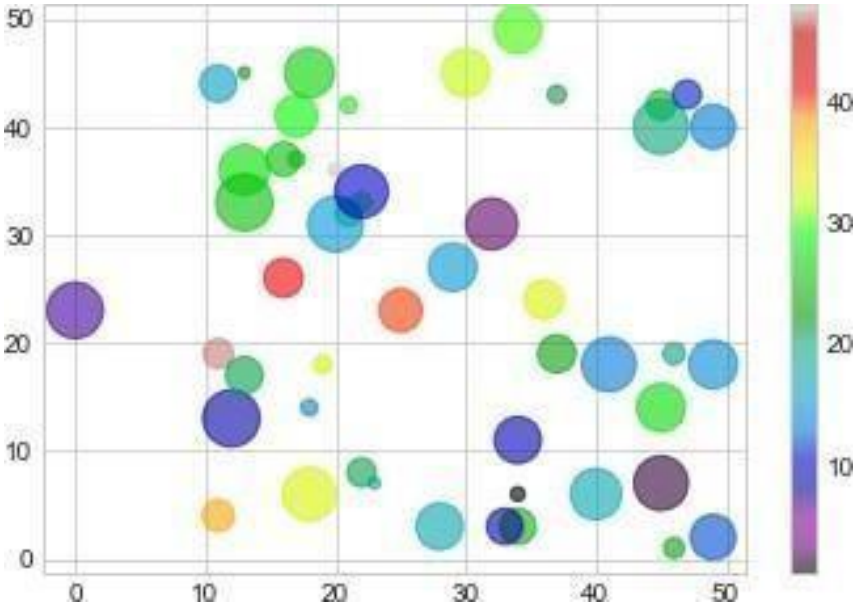


Scatter plots with *plt.scatter*:

plt.scatter function, which can be used very similarly to the plt.plotfunction.

The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, facecolor, edge color, etc.) can be individually controlled or mapped to data. the alpha keyword to adjust the transparency level:

```
import matplotlib.pyplot as plt
import numpy as np
x_axis_val = np.random.randint(50, size = (50))
y_axis_val = np.random.randint(50, size = (50))
colors_value = np.random.randint(50, size = (50))
sizes_val = 10 * np.random.randint(50, size = (50))
plt.scatter(x_axis_val, y_axis_val, c = colors_value, s = sizes_val, alpha = 0.6, cmap =
'nipy_spectral')
plt.colorbar()
 plt.show()
```



Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to implement scatter plots was executed and output is verified.

EX.NO: 14 **THREE-DIMENSIONAL PLOTTING IN MATPLOTLIB**

DATE

Aim:

To create a user defined numpy array and visualize the data in three-++dimensional plotting using matplotlib in python.

Algorithm:

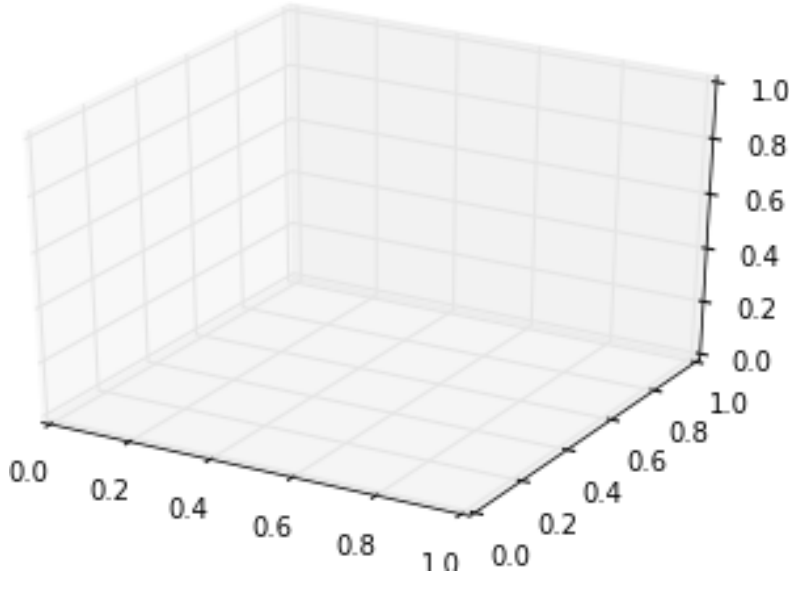
1. Start the program.
2. Import matplotlib.pyplot and mpl_toolkits.mplot3d for 3D plotting.
3. Import numpy for generating sample data.
4. Create figure and a 3D axis using projection='3d'.
5. Generate X, Y, and Z coordinates using numpy arrays.
6. Plot the 3D data using ax.scatter() or ax.plot_surface().
7. Label the X, Y, and Z axes.
8. Add a title and customize styles if needed.
9. Display the plot using plt.show().
10. End the program.

Three-dimensional plotting is one of the functionalities that benefits immensely from viewing figures interactively rather than statically. Three-dimensional plots are enabled by importing the mplot3d toolkit, included with the main Matplotlib installation:

```
from mpl_toolkits import mplot3d
```

Once this submodule is imported, a three-dimensional axes can be created by passing the keyword projection='3d' to any of the normal axes creation routines:

```
%matplotlib inline
 import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
```

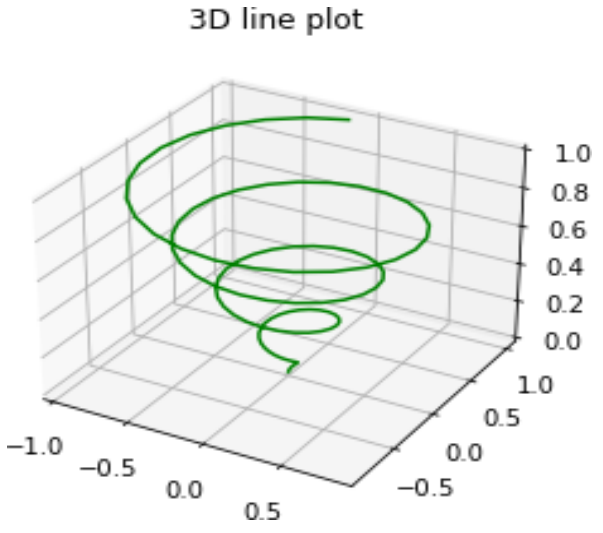


With this three-dimensional axes enabled, we can now plot a variety of three-dimensional plottypes.

Three-dimensional Points and Lines:

The most basic three-dimensional plot is a line or collection of scatter plot created from sets of (x, y, z) triples. These can be created using the ax.plot3D and ax.scatter3D functions. Here we'll plot a trigonometric spiral, along with some points drawn randomly near the line:

```
# syntax for 3-D projection
ax = plt.axes(projection ='3d')

# defining all 3 axis
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)

# plotting
ax.plot3D(x, y, z, 'green')
ax.set_title('3D line plot')
plt.show()
```

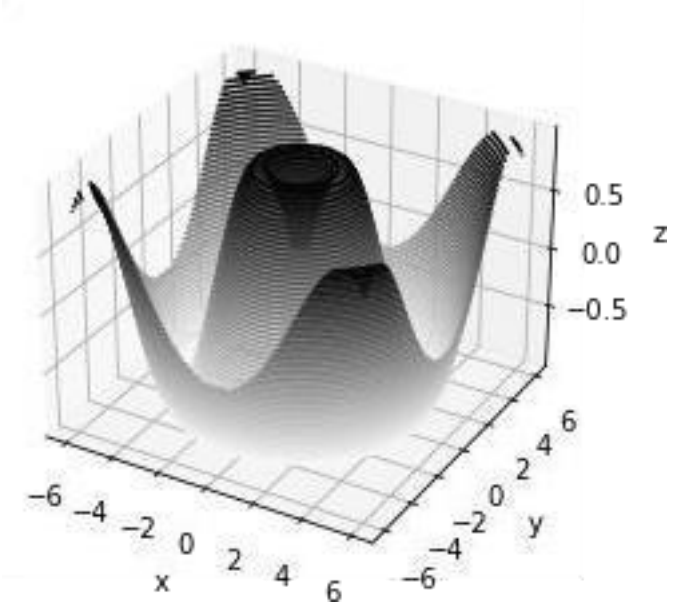


Notice that by default, the scatter points have their transparency adjusted to give a sense of depth on the page. While the three-dimensional effect is sometimes difficult to see within a static image, an interactive view can lead to some nice intuition about the layout of the points.

Three-dimensional Contour Plots:

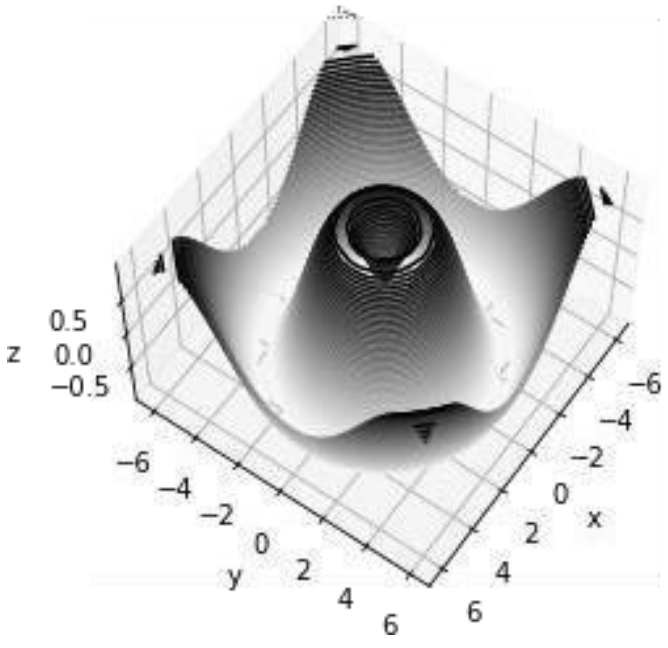
mplot3d contains tools to create three-dimensional relief plots using the same inputs. Like two-dimensional ax.contour plots, ax.contour3D requires all the input data to be in the form of two-dimensional regular grids, with the Z data evaluated at each point. Here we'll show a three-dimensional contour diagram of a three-dimensional sinusoidal function:

```
def f(x, y):
 return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```



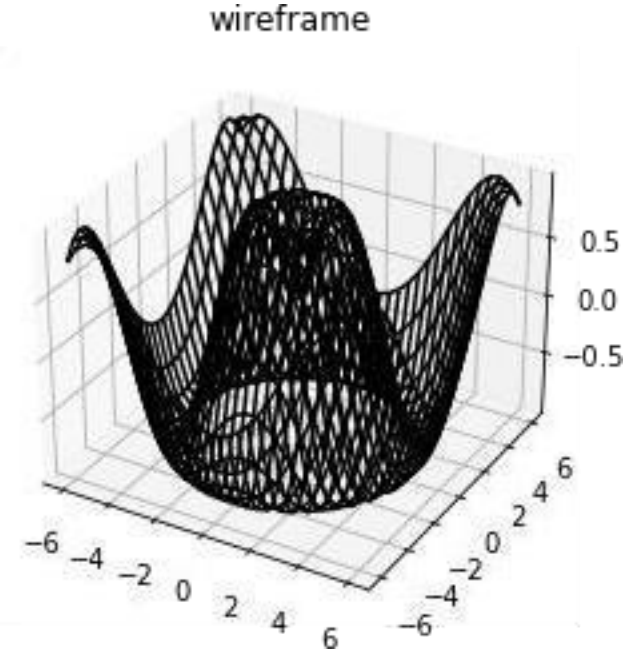
If the default viewing angle is not optimal, we can use the view_init method to set the elevation and azimuthalangles. In the following example, we'll use an elevation of 60 degrees (that is, 60 degrees above the x-y plane)and an azimuth of 35 degrees (that is, rotated35 degrees counter-clockwise about the z-axis):

```
ax.view_init(60, 35)
fig
```



Wireframes and Surface Plots:

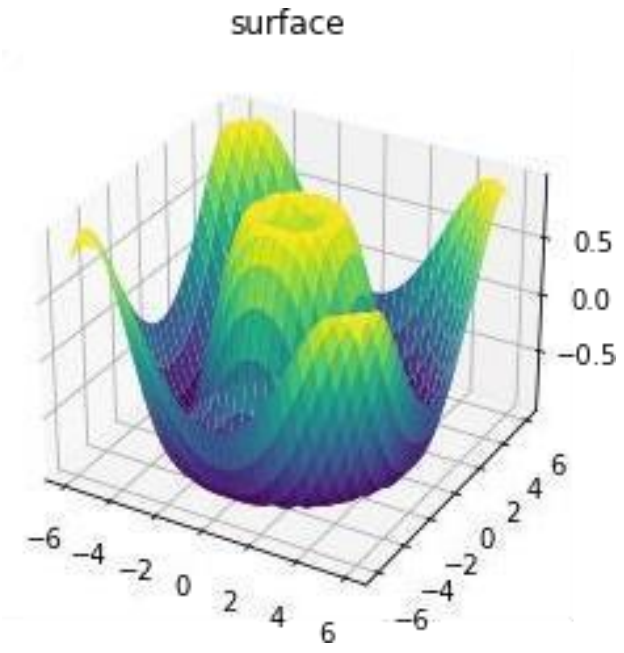
Two other types of three-dimensional plots that work on gridded data are wireframes andsurface plots. These take a grid of values and project it onto the specified three-dimensional surface and can make the resulting three-dimensional forms quite easy to visualize. Here's an example of using a wireframe:




```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black')
ax.set_title('wireframe');
```

A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon. Adding a colormap to the filled polygons can aid perception of the topology of the surface being visualized:

```
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,cmap='viridis',
                edgecolor='none')
ax.set_title('surface');
```



Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to create a user defined numpy array and visualize the data in three dimensional plotting using matplotlib.

EX.NO: 15 IMPLEMENT OF ANALYTIC APPLICATION

DATE:

Aim:

To import a real time data and implement an analytic application using numpy, pandas and matplotlib in python

Algorithm:

1. Start the Program.
2. install and import pandas Packages to python.

   !pip install pandas

   import pandas as pd
3. install and import numpy Packages to python.

   !pip install numpy

   import numpy as np
4. install and import matplotlib Packages to python.

   !pip install matplotlib

   import matplotlib.pyplot as plt
5. Import the dataset using pd.read_csv.
6. Clean the dataset by removing unknown and NULL values.
7. Visualize the results.
8. Stop the program.

```
 import numpy as np
  import pandas as pd
import matplotlib as plt
%matplotlib inline
```

Loading the data into the data frame:

```
df = pd.read_csv('world_cup_matches.csv')
#To display top 5 rows
print(df.head())

#head default value is 5
```

	ID	Year	Date	Stage	Home Team	Home Goals	Away Goals	Away Team	Win Conditions	Host Team
0	1	1930	1930-07-13	Group stage	France	4	1	Mexico	NaN	False
1	2	1930	1930-07-13	Group stage	United States	3	0	Belgium	NaN	False
2	3	1930	1930-07-14	Group stage	Yugoslavia	2	1	Brazil	NaN	False
3	4	1930	1930-07-14	Group stage	Romania	3	1	Peru	NaN	False
4	5	1930	1930-07-15	Group stage	Argentina	1	0	France	NaN	False

```
#To display last 5 rows
print(df.tail())
```

	ID	Year	Date	Stage	Home Team	Home Goals	Away Goals	Away Team	Win Conditions	Host Team
895	896	2018	2018-07-07	Quarter-finals	Russia	2	2	Croatia	Croatia win on penalties (3 - 4)	True
896	897	2018	2018-07-10	Semi-finals	France	1	0	Belgium	NaN	False
897	898	2018	2018-07-11	Semi-finals	Croatia	2	1	England	Extra time	False
898	899	2018	2018-07-14	Third place	Belgium	2	0	England	NaN	False
899	900	2018	2018-07-15	Final	France	4	2	Croatia	NaN	False

Checking the types of data:

```
df.dtypes
```

```
ID              int64
Year            int64
Date           object
Stage          object
Home Team      object
Home Goals      int64
Away Goals      int64
Away Team      object
Win Conditions object
Host Team        bool
dtype: object
```

Dropping irrelevant columns:

```
df = df.drop(['Host Team'],axis=1)
#To Remove unwanted columns
#To verify of dropping columns
df.columns
```

```
Index(['ID', 'Year', 'Date', 'Stage', 'Home Team', 'Home Goals', 'Away Goals',
       'Away Team', 'Win Conditions'],
      dtype='object')
```

Dropping the duplicate rows:

```
df = df.drop_duplicates()#To Remove duplicate rows
df.head()
```

	ID	Year	Date	Stage	Home Team	Home Goals	Away Goals	Away Team	Win Conditions
0	1	1930	1930-07-13	Group stage	France	4	1	Mexico	NaN
1	2	1930	1930-07-13	Group stage	United States	3	0	Belgium	NaN
2	3	1930	1930-07-14	Group stage	Yugoslavia	2	1	Brazil	NaN
3	4	1930	1930-07-14	Group stage	Romania	3	1	Peru	NaN
4	5	1930	1930-07-15	Group stage	Argentina	1	0	France	NaN

Dropping the missing or null values:

```
print(df.isnull().sum()) #returns the sum of null values
```

```
ID                0
Year              0
Date              0
Stage             0
Home Team         0
Home Goals        0
Away Goals        0
Away Team         0
Win Conditions    838
dtype: int64
```

```
df=df.dropna()#Dropping the missing values.
df.count()#To count the No. of values.
```

```
ID                62
Year              62
Date              62
Stage             62
Home Team         62
Home Goals        62
Away Goals        62
Away Team         62
Win Conditions    62
dtype: int64
```

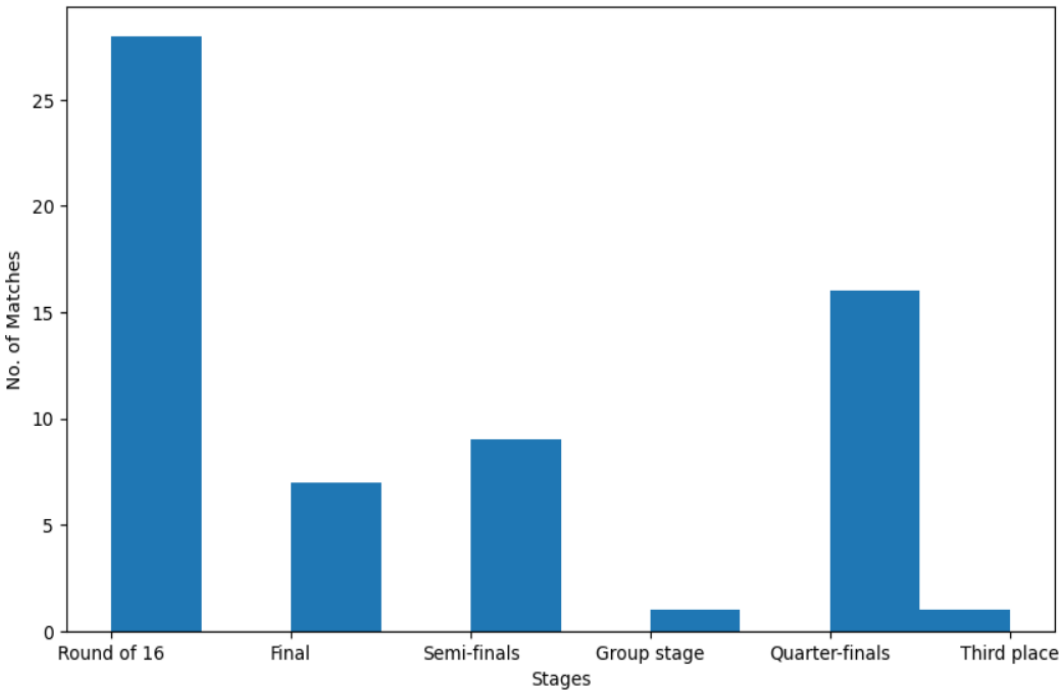
```
print(df.isnull().sum())# After dropping the values
# The dataset is clean now
```

```
ID                0
Year              0
Date              0
Stage             0
Home Team         0
Home Goals        0
Away Goals        0
Away Team         0
Win Conditions    0
dtype: int64
```

Data Visualization:

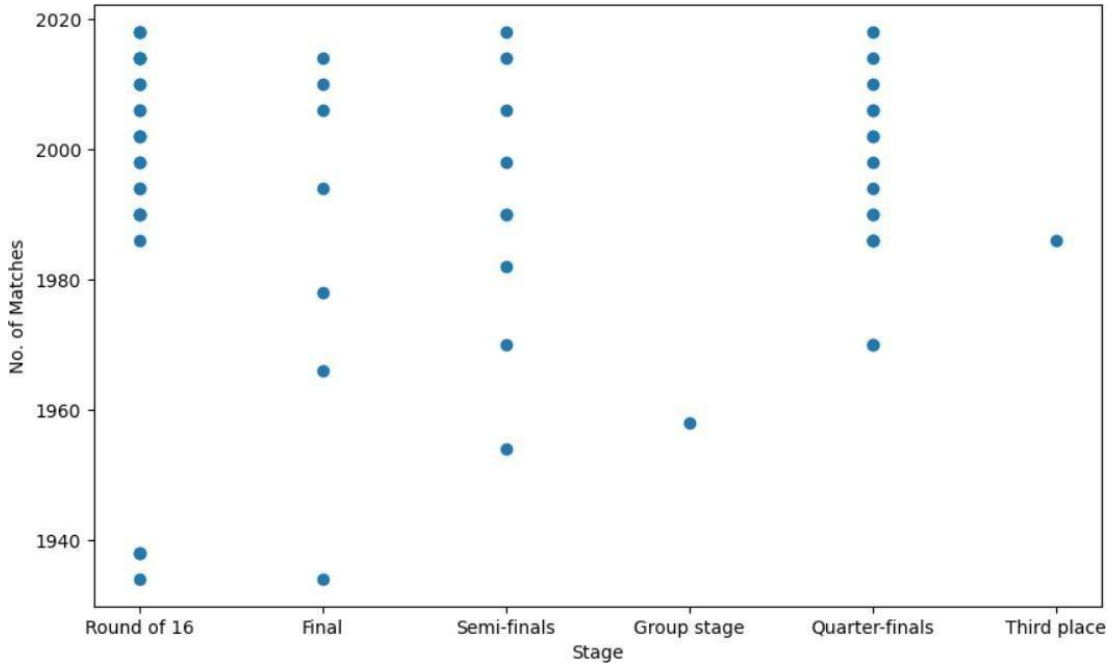
Histogram

```
plt.subplots(figsize=(10,6))
plt.hist(df['Stage'])
plt.xlabel('Stages')
plt.ylabel('No. of Matches')
plt.show()
```



Scatterplot:

```
plt.subplots(figsize=(10,6))
plt.scatter(df['Stage'],df['Year'])
plt.xlabel('Stage')
plt.ylabel('No. of Matches')
plt.show()
```



Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to create an analytic application was implemented and the analysis is verified.

EX.NO: 16 IMPLEMENT AN APPLICATION TO PROCESS A REAL TIME DATA

DATE:

Aim:

To import a real time data and implement an application to process the data using numpy, Pandas and matplotlib in python

Algorithm:

1. Start the Program.
2. install and import pandas Packages to python.
   !pip install pandas

   import pandas as pd
3. install and import numpy Packages to python.
   !pip install numpy

   import numpy as np
4. install and import matplotlib Packages to python.
   !pip install matplotlib

   import matplotlib.pyplot as plt
5. Import the dataset using pd.read_csv.
6. Clean the dataset by removing unwanted and NULL values.
7. Visualize the data.
8. Stop the program.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading the data into the data frame:

```
df = pd.read_csv("titanic.csv")
#to display the five rows
df.head(5)
```

	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

```
df.tail(5)
```

	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
886	886	0	2	male	27.0	0	0	13.00	S	Second	man	True	NaN	Southampton	no	True
887	887	1	1	female	19.0	0	0	30.00	S	First	woman	False	B	Southampton	yes	True
888	888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	NaN	Southampton	no	False
889	889	1	1	male	26.0	0	0	30.00	C	First	man	True	C	Cherbourg	yes	True
890	890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	NaN	Queenstown	no	True

Checking the types of data:

```
df.types()
```

```
Unnamed: 0        int64
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class            object
who              object
adult_male         bool
deck             object
embark_town      object
alive            object
alone              bool
dtype: object
```

Dropping irrelevant columns:

```
df = df.drop(['embarked', 'deck', 'alone'], axis=1)
df.head(5)
```

	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	class	who	adult_male	embark_town	alive
0	0	0	3	male	22.0	1	0	7.2500	Third	man	True	Southampton	no
1	1	1	1	female	38.0	1	0	71.2833	First	woman	False	Cherbourg	yes
2	2	1	3	female	26.0	0	0	7.9250	Third	woman	False	Southampton	yes
3	3	1	1	female	35.0	1	0	53.1000	First	woman	False	Southampton	yes
4	4	0	3	male	35.0	0	0	8.0500	Third	man	True	Southampton	no

Dropping the duplicate rows:

```
df = df.drop_duplicates()
df.head(5)
```

	Unnamed: 0	survived	pclass	sex	age	sibsp	parch	fare	class	who	adult_male	embark_town	alive
0	0	0	3	male	22.0	1	0	7.2500	Third	man	True	Southampton	no
1	1	1	1	female	38.0	1	0	71.2833	First	woman	False	Cherbourg	yes
2	2	1	3	female	26.0	0	0	7.9250	Third	woman	False	Southampton	yes
3	3	1	1	female	35.0	1	0	53.1000	First	woman	False	Southampton	yes
4	4	0	3	male	35.0	0	0	8.0500	Third	man	True	Southampton	no

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

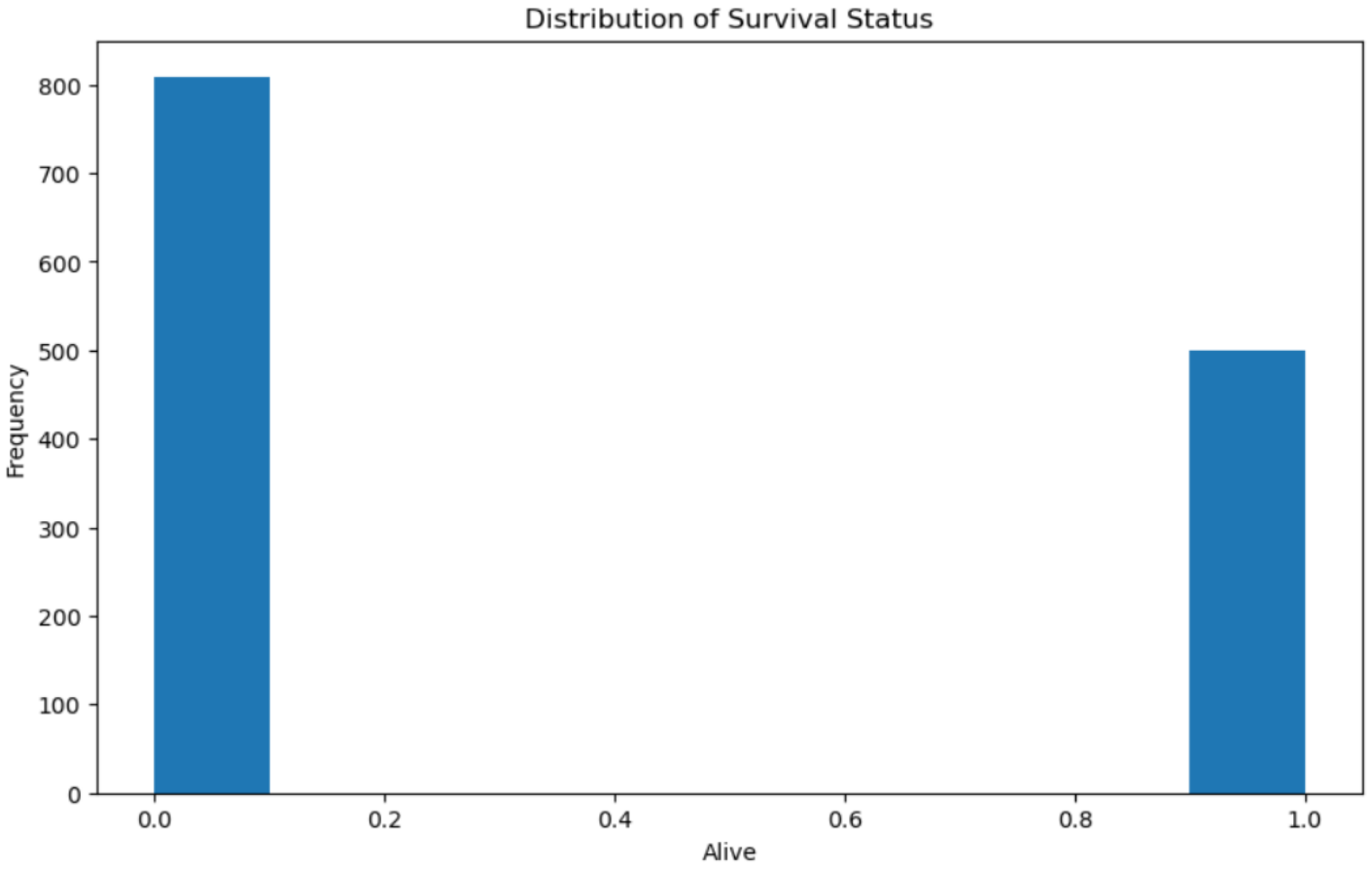
```
Unnamed: 0        0
survived          0
pclass            0
sex               0
age             177
sibsp             0
parch             0
fare              0
class             0
who               0
adult_male        0
embark_town       2
alive             0
dtype: int64
```

```
df=df.dropna()
```

```
Unnamed: 0     712
survived       712
pclass         712
sex            712
age            712
sibsp          712
parch          712
fare           712
class          712
who            712
adult_male     712
embark_town    712
alive          712
dtype: int64
```

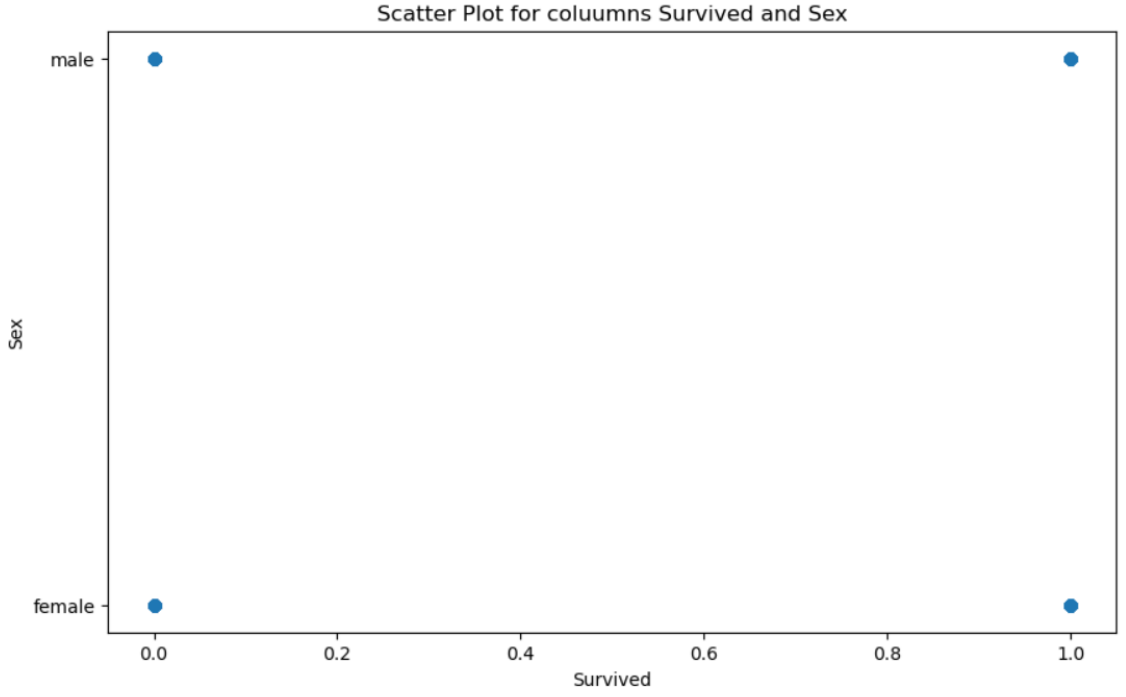
Data Visualization:
Histogram:

```
plt.figure(figsize=(10, 6))
plt.hist(df['survived'])
plt.xlabel('Alive')
plt.ylabel('Frequency')
plt.title('Distribution of Survival Status')
plt.show()
```



Scatterplot:

```
plt.figure(figsize=(10, 6))
plt.scatter(df['survived'], df['sex'])
plt.xlabel('Survived')
plt.ylabel('Sex')
plt.title('Scatter Plot for columns Survived and Sex')
plt.show()
```



Criteria	Maximum Marks	Marks Obtained
Aim and Algorithm	**05**	
Program and Output	**15**	
Viva	**05**	
Total	**25**	

Result:

Thus, the python program to create an analytic application was implemented and the analysis is verified.