**Experiment 1**

**Program No.:1**

**Aim:** Convert kilograms to pounds.

**SOURCE CODE:**

def kg\_to\_pounds():

"""

Converts a weight entered in kilograms to pounds.

"""

try:

kilograms = float(input("Enter the weight in kilograms: "))

pounds = kilograms \* 2.20462

print(f"The weight in pounds is: {pounds:.2f}")

except ValueError:

print("Invalid input. Please enter a numerical value for weight.")

# Call the function directly to run the conversion

kg\_to\_pounds()

**INPUT:**

Enter the weight in kilograms: 12

**EXPECTED OUTPUT:**

The weight in pounds is: 26.46

**ACTUAL OUTPUT:**

The weight in pounds is: 26.46

**Program No.:1.1**

**Aim:** Write a program that uses a for loop and while loop to print the numbers 8,11,14,…,89.

**SOURCE CODE:**

**FOR LOOP**

print("Using a for loop:")

for num in range(8, 90, 3):

print(num)

**WHILE LOOP**

print("Using a while loop:")

num\_while = 8

while num\_while <= 89:

print(num\_while)

num\_while += 3

**EXPECTED OUTPUT:**

Using a while loop / For loop:

8

11

14

17

20

23

26

29

32

35

38

41

44

47

50

53

56

59

62

65

68

71

74

77

80

83

86

89

**ACTUAL OUTPUT:**

Using a while loop / For loop:

8

11

14

17

20

23

26

29

32

35

38

41

44

47

50

53

56

59

62

65

68

71

74

77

80

83

86

89

**Program No.:1.2**

**Aim:**  Write a program to split a string into array of characters.

**SOURCE CODE:**

my\_string = "venky"

char\_list = list(my\_string)

print(char\_list)

**EXPECTED OUTPUT:**

[ 'v', 'e', 'n', 'k', 'y']

**ACTUAL OUTPUT:**

[ 'v', 'e', 'n', 'k', 'y']

**Program 1.3:**

**Aim:**Write a Python program to calculate the nth Fibonacci number using a function.

**SOURCE CODE:**

def fibonacci(n):

if n <= 0:

return "Input must be a positive integer"

elif n == 1:

return 0

elif n == 2:

return 1

else:

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

n = 10

print(f"The {n}th Fibonacci number is: {fibonacci(n)}")

**INPUT:**

n = 10

**EXPECTED OUTPUT:**

The 10th Fibonacci number is: 34

**ACTUAL OUTPUT:**

The 10th Fibonacci number is: 34

**Program 1.4:**

**Aim:** Write a Python program to get the largest number from a list.

**SOURCE CODE:**

def get\_largest\_number(numbers):

if not numbers:

return None

largest = numbers[0]

for num in numbers[1:]:

if num > largest:

largest = num

return largest

my\_list = [10, 45, 2, 99, 67, 88]

largest\_number = get\_largest\_number(my\_list)

print("The largest number is:", largest\_number)

**INPUT:**

my\_list = [10, 45, 2, 99, 67, 88]

**EXPECTED OUTPUT:**

The largest number is: 99

**ACTUAL OUTPUT:**

The largest number is: 99

Bottom of Form

**EXPRIMENT 2**

**Program 2:**

**Aim:** Write a Python program that defines a Car class with attributes like make, model, and year, and methods like start() to start the car and stop() to stop it.

**SOURCE CODE:**

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

self.is\_started = False

def start(self):

if not self.is\_started:

print(f"The {self.year} {self.make} {self.model} is starting.")

self.is\_started = True

else:

print(f"The {self.year} {self.make} {self.model} is already running.")

def stop(self):

if self.is\_started:

print(f"The {self.year} {self.make} {self.model} is stopping.")

self.is\_started = False

else:

print(f"The {self.year} {self.make} {self.model} is already stopped.")

# Demonstrating the Car class

print("--- Car Class Demonstration ---")

my\_car = Car("Toyota", "Camry", 2022)

print(f"My car: {my\_car.make} {my\_car.model} ({my\_car.year})")

my\_car.start()

my\_car.start()

my\_car.stop()

my\_car.stop()

**INPUT:**

Car("Toyota", "Camry", 2022)

**EXPECTED OUTPUT:**

--- Car Class Demonstration ---

My car: Toyota Camry (2022)

The 2022 Toyota Camry is starting.

The 2022 Toyota Camry is already running.

The 2022 Toyota Camry is stopping.

The 2022 Toyota Camry is already stopped.

**ACTUAL OUTPUT:**

--- Car Class Demonstration ---

My car: Toyota Camry (2022)

The 2022 Toyota Camry is starting.

The 2022 Toyota Camry is already running.

The 2022 Toyota Camry is stopping.

The 2022 Toyota Camry is already stopped.

**Program 2.1:**

**Aim:** Write a Python program that demonstrates inheritance by creating a base class Animal and derived classes like Dog and Cat with specific behaviors.

**SOURCE CODE:**

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def eat(self):

print(f"{self.name} is eating.")

def sleep(self):

print(f"{self.name} is sleeping.")

class Dog(Animal):

def bark(self):

print(f"{self.name} says Woof!")

class Cat(Animal):

def meow(self):

print(f"{self.name} says Meow!")

dog1 = Dog("Buddy")

cat1 = Cat("Whiskers")

dog1.eat()

dog1.sleep()

cat1.eat()

cat1.sleep()

dog1.bark()

cat1.meow()

.

**INPUT:**

Dog("Buddy"), Cat("Whiskers")

**EXPECTED OUTPUT:**

Buddy is eating.

Buddy is sleeping.

Whiskers is eating.

Whiskers is sleeping.

Buddy says Woof!

Whiskers says Meow!

**ACTUAL OUTPUT:**

Buddy is eating.

Buddy is sleeping.

Whiskers is eating.

Whiskers is sleeping.

Buddy says Woof!

Whiskers says Meow!

**Program 2.2:**

**Aim:** Write a Python program that demonstrates error handling using the try-except block to handle division by zero.

**SOURCE CODE:**

def safe\_divide(numerator, denominator):

try:

result = numerator / denominator

print(f"The result of {numerator} / {denominator} is: {result}")

except ZeroDivisionError:

print(f"Error: Cannot divide by zero! Attempted {numerator} / {denominator}")

except TypeError:

print("Error: Invalid input types. Please provide numbers.")

print("--- Error Handling Demonstration ---")

safe\_divide(10, 2)

safe\_divide(10, 0)

safe\_divide(5, "abc")

**INPUT:**

safe\_divide(10, 2)

safe\_divide(10, 0)

safe\_divide(5, "abc")

**EXPECTED OUTPUT:**

The result of 10 / 2 is: 5.0

Error: Cannot divide by zero! Attempted 10 / 0

Error: Invalid input types. Please provide numbers.

**ACTUAL OUTPUT:**

The result of 10 / 2 is: 5.0

Error: Cannot divide by zero! Attempted 10 / 0

Error: Invalid input types. Please provide number

**Program 2.3:**

**Aim:**

Write a Python program that defines a base class Animal with a method make\_sound() and derived classes Dog, Cat, and Bird that override the method to produce different sounds. Demonstrate polymorphism.

**SOURCE CODE:**

class Animal:

def make\_sound(self):

raise NotImplementedError("Subclass must implement abstract method 'make\_sound'")

class Dog(Animal):

def make\_sound(self):

return "Woof! Woof!"

class Cat(Animal):

def make\_sound(self):

return "Meow!"

class Bird(Animal):

def make\_sound(self):

return "Chirp! Chirp!"

print("--- Polymorphism Demonstration ---")

animals = [Dog(), Cat(), Bird()]

for animal in animals:

print(f"An animal makes sound: {animal.make\_sound()}")

**INPUT:**

animals = [Dog(), Cat(), Bird()]

**EXPECTED OUTPUT:**

--- Polymorphism Demonstration ---

An animal makes sound: Woof! Woof!

An animal makes sound: Meow!

An animal makes sound: Chirp! Chirp!

**ACTUAL OUTPUT:**

--- Polymorphism Demonstration ---

An animal makes sound: Woof! Woof!

An animal makes sound: Meow!

An animal makes sound: Chirp! Chirp!

Top of Form

Bottom of Form

**Program No.: 2.4**

**Aim:**Demonstrate error handling in Python using the try-except block to handle division by zero.

**SOURCE CODE:**

defsafe division():

"""

Demonstrates error handling for division by zero.

"""

try:

numerator = float(input("Enter numerator: "))

denominator = float(input("Enter denominator: "))

result = numerator / denominator

print(f"Result: {result:.2f}")

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

except ValueError:

print("Invalid input. Please enter numerical values only.")

# Run the function

safe\_division()

**INPUT (Case 1):**

Enter numerator: 10

Enter denominator: 2

EXPECTED / ACTUAL OUTPUT (Case 1):

Result: 5.00

**INPUT (Case 2):**

Enter numerator: 10

Enter denominator: 0

EXPECTED / ACTUAL OUTPUT (Case 2):

Error: Division by zero is not allowed.

**Experiment 3**

**Program No.: 3.1**

**Aim**:

Demonstrate the use of NumPy methods for array creation, element-wise operations, comparisons, and conversions.

**SOURCE CODE:**

import numpy as np  
  
# Get help on add  
help(np.add)  
  
# Check if none are zero  
arr = np.array([1, 2, 3, 4, 5])  
print("Array:", arr)  
print("None of the elements is zero:", np.all(arr))  
  
# Comparisons  
a = np.array([1, 2, 3, 4])  
b = np.array([2, 2, 1, 4])  
print("Greater:", np.greater(a, b))  
print("Greater or Equal:", np.greater\_equal(a, b))  
print("Less:", np.less(a, b))  
print("Less or Equal:", np.less\_equal(a, b))  
print("Equal:", np.equal(a, b))  
print("Allclose:", np.allclose(a, b))  
  
# Create arrays  
zeros\_array = np.zeros(5)  
ones\_array = np.ones(5)  
linspace\_array = np.linspace(0, 10, 5)  
print("Zeros array:", zeros\_array)  
print("Ones array:", ones\_array)  
print("Linspace array:", linspace\_array)  
  
# Convert to list  
array\_list = a.tolist()  
print("Array converted to list:", array\_list)

**INPUT:**

Predefined arrays in code.

**EXPECTED / ACTUAL OUTPUT:**

Array: [1 2 3 4 5]  
None of the elements is zero: True  
  
Greater: [False False True False]  
Greater or Equal: [False True True True]  
Less: [ True False False False]  
Less or Equal: [ True True False True]  
Equal: [False True False True]  
Allclose: False  
  
Zeros array: [0. 0. 0. 0. 0.]  
Ones array: [1. 1. 1. 1. 1.]  
Linspace array: [ 0. 2.5 5. 7.5 10. ]  
Array converted to list: [1, 2, 3, 4]

**Program No.: 3.2**

**Aim:**

Demonstrate NumPy methods for extracting, analyzing, and summarizing array data.

**SOURCE CODE:**

import numpy as np  
  
arr = np.array([1, 3, 5, 7, 3, 5, 9, 2, 5])  
num = 4  
  
less\_than\_num = arr[arr < num]  
greater\_than\_num = arr[arr > num]  
print("Original array:", arr)  
print("Numbers less than 4:", less\_than\_num)  
print("Numbers greater than 4:", greater\_than\_num)  
  
print("Maximum value:", np.max(arr))  
print("Minimum value:", np.min(arr))  
print("Index of maximum value:", np.argmax(arr))  
print("Index of minimum value:", np.argmin(arr))  
  
print("Unique elements:", np.unique(arr))  
print("Count of each element:", np.bincount(arr))  
print("Representation of array:", np.array\_repr(arr))

**INPUT:**

Predefined array and threshold.

**EXPECTED / ACTUAL OUTPUT:**

Original array: [1 3 5 7 3 5 9 2 5]  
Numbers less than 4: [1 3 3 2]  
Numbers greater than 4: [5 7 5 9 5]  
Maximum value: 9  
Minimum value: 1  
Index of maximum value: 6  
Index of minimum value: 0  
Unique elements: [1 2 3 5 7 9]  
Count of each element: [0 1 1 2 0 3 0 1 0 1]  
Representation of array: array([1, 3, 5, 7, 3, 5, 9, 2, 5])

**Experiment 4**

**Program No.: 4.1**

**Aim:**

To create and display a one-dimensional Pandas Series and convert it to a Python list.

**SOURCE CODE:**

import pandas as pd  
  
# Create and display a Pandas Series  
data\_list = pd.Series([10, 20, 30, 40, 50])  
print("Pandas Series created from a list:")  
print(data\_list)  
  
# Convert the Series to a Python list and display type  
data\_list\_converted = data\_list.tolist()  
print("\nConverted to Python list:", data\_list\_converted)  
print("Type of converted object:", type(data\_list\_converted))

**INPUT:**

Predefined list: [10, 20, 30, 40, 50]

**EXPECTED / ACTUAL OUTPUT:**

Pandas Series created from a list:  
0 10  
1 20  
2 30  
3 40  
4 50  
dtype: int64  
  
Converted to Python list: [10, 20, 30, 40, 50]  
Type of converted object: <class 'list'>

**Program No.: 4.2**

**Aim:**

Create and manipulate a Pandas DataFrame from a dictionary, update values, add a new column, and display column headers.

**SOURCE CODE:**

import pandas as pd  
import numpy as np  
  
exam\_data = {  
 'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],  
 'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],  
 'attempts': [1, 3, 2, 3, 1, 1, 2, 1, 1, 2],  
 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'yes', 'yes']  
}  
labels = ['a','b','c','d','e','f','g','h','i','j']  
  
df = pd.DataFrame(exam\_data, index=labels)  
print("DataFrame created from dictionary:")  
print(df)  
  
df.loc['d','name'] = 'Suresh'  
print("\nDataFrame after updating name 'James' to 'Suresh':")  
print(df)  
  
salaries = [50000, 60000, 75000, 45000, 55000, 80000, 70000, 48000, 62000, 78000]  
df['salary'] = salaries  
print("\nDataFrame after adding 'salary' column:")  
print(df)  
  
column\_headers = df.columns.tolist()  
print("\nList of column headers:", column\_headers)

**INPUT:**

Predefined dictionary with student data.

**EXPECTED / ACTUAL OUTPUT:**

DataFrame created from dictionary:

name score attempts qualify  
a Anastasia 12.5 1 yes  
b Dima 9.0 3 no  
c Katherine 16.5 2 yes  
d James NaN 3 no  
e Emily 9.0 1 no  
f Michael 20.0 1 yes  
g Matthew 14.5 2 yes  
h Laura NaN 1 no  
i Kevin 8.0 1 yes  
j Jonas 19.0 2 yes  
  
DataFrame after updating name 'James' to 'Suresh':  
 name score attempts qualify  
a Anastasia 12.5 1 yes  
b Dima 9.0 3 no  
c Katherine 16.5 2 yes  
d Suresh NaN 3 no  
e Emily 9.0 1 no  
f Michael 20.0 1 yes  
g Matthew 14.5 2 yes  
h Laura NaN 1 no  
i Kevin 8.0 1 yes  
j Jonas 19.0 2 yes  
  
DataFrame after adding 'salary' column:  
 name score attempts qualify salary  
a Anastasia 12.5 1 yes 50000  
b Dima 9.0 3 no 60000  
c Katherine 16.5 2 yes 75000  
d Suresh NaN 3 no 45000  
e Emily 9.0 1 no 55000  
f Michael 20.0 1 yes 80000  
g Matthew 14.5 2 yes 70000  
h Laura NaN 1 no 48000  
i Kevin 8.0 1 yes 62000  
j Jonas 19.0 2 yes 78000  
  
List of column headers: ['name', 'score', 'attempts', 'qualify', 'salary']

**Experiment 5**

**Program No.: 5.1**

**Aim:**

Create a series of plots to analyze a given dataset.

**SOURCE CODE:**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0, 2 \* np.pi, 100)

y = np.sin(x)

y2 = np.cos(x)

y3 = np.sin(x) \* np.cos(x)

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.plot(x, y, color='blue')

plt.title('Sine Wave')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.subplot(1, 3, 2)

plt.plot(x, y2, color='green')

plt.title('Cosine Wave')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.subplot(1, 3, 3)

plt.plot(x, y3, color='red')

plt.title('Product of Sine and Cosine')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.tight\_layout()

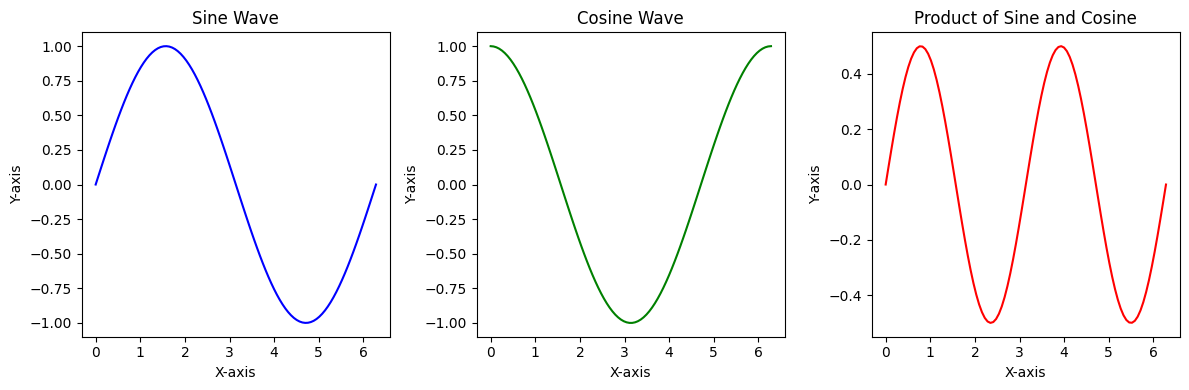
plt.show()

**INPUT:**

 **x** → 100 points from 0 to 2π.

 **y, y2, y3** → Calculated using sin(x), cos(x), and sin(x)\*cos(x).

**EXPECTED / ACTUAL OUTPUT:**



**Program No.: 5.2**

**Aim:**

Generate a subplot layout with various plot types (scatter, line, bar).

**SOURCE CODE:**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

data = {'Category': ['A', 'B', 'C', 'D'],

'Value1': [10, 25, 15, 30],

'Value2': [15, 20, 25, 10]}

df = pd.DataFrame(data)

x\_scatter = np.random.rand(50)

y\_scatter = np.random.rand(50)

x\_line = np.arange(10)

y\_line = np.random.randint(1, 10, size=10)

fig, axes = plt.subplots(2, 2, figsize=(10, 8))

axes[0, 0].scatter(x\_scatter, y\_scatter)

axes[0, 0].set\_title('Scatter Plot')

axes[0, 0].set\_xlabel('X-axis')

axes[0, 0].set\_ylabel('Y-axis')

axes[0, 1].plot(x\_line, y\_line, marker='o')

axes[0, 1].set\_title('Line Plot')

axes[0, 1].set\_xlabel('X-axis')

axes[0, 1].set\_ylabel('Y-axis')

axes[1, 0].bar(df['Category'], df['Value1'])

axes[1, 0].set\_title('Bar Chart')

axes[1, 0].set\_xlabel('Category')

axes[1, 0].set\_ylabel('Value1')

axes[1, 1].set\_visible(False)

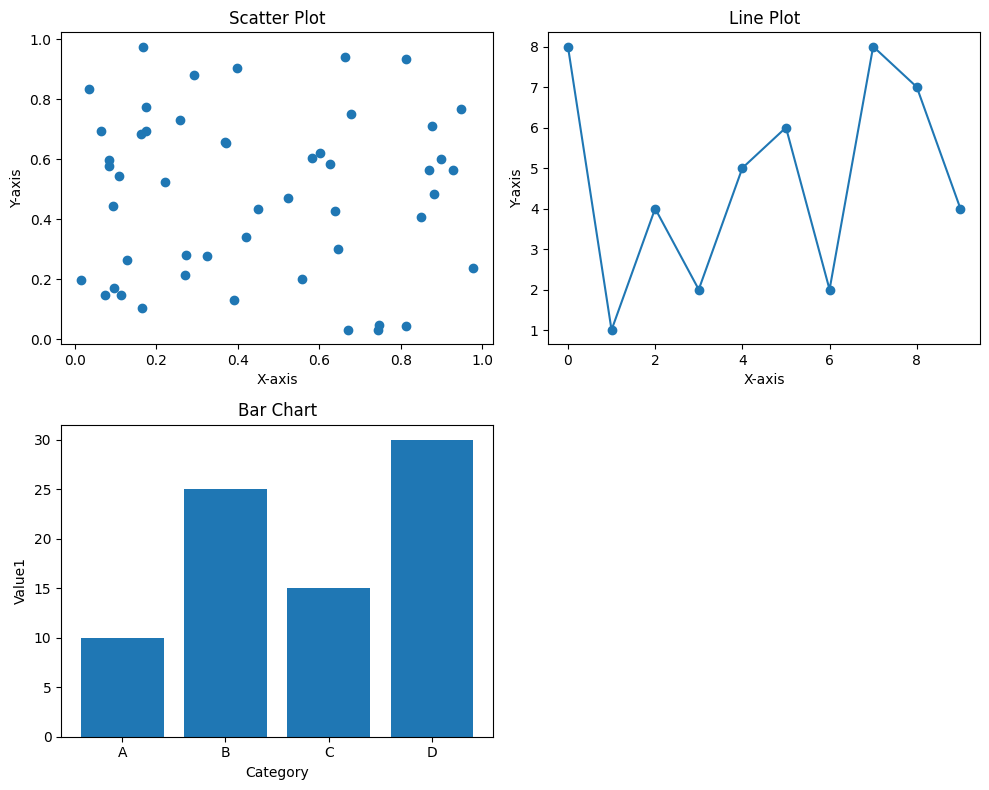
plt.tight\_layout()

plt.show()

**INPUT:**

* categories: ['A', 'B', 'C', 'D']
* Random numeric arrays (x\_scatter, y\_scatter, y\_line)
* Tabular data (Value1, Value2)

**EXPECTED / ACTUAL OUTPUT:**



**Program No.: 5.3**

**Aim:**

Visualize time-series data and customize axis labels and date formats.

**SOURCE CODE:**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

dates = pd.to\_datetime(pd.date\_range(start='2024-01-01', periods=100, freq='D'))

sales\_data = np.random.randint(100, 500, size=100) + np.sin(np.arange(100) \* 0.2) \* 50

df = pd.DataFrame({'Date': dates, 'Sales': sales\_data})

df = df.set\_index('Date')

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

plt.plot(df.index, df['Sales'])

plt.xlabel('Date')

plt.ylabel('Sales')

plt.title('Daily Sales Over Time')

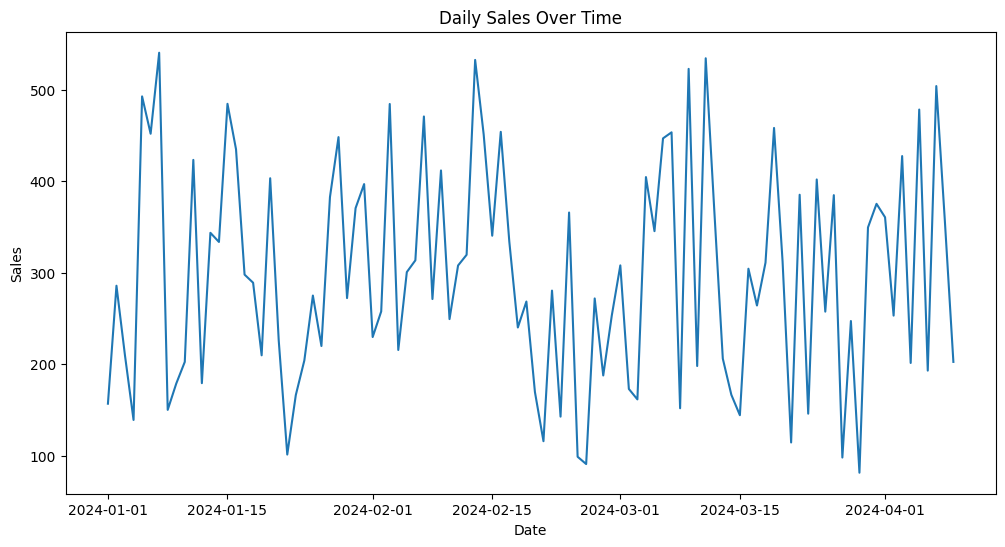
df.plot(y='Sales', figsize=(12, 6), title='Daily Sales Over Time')

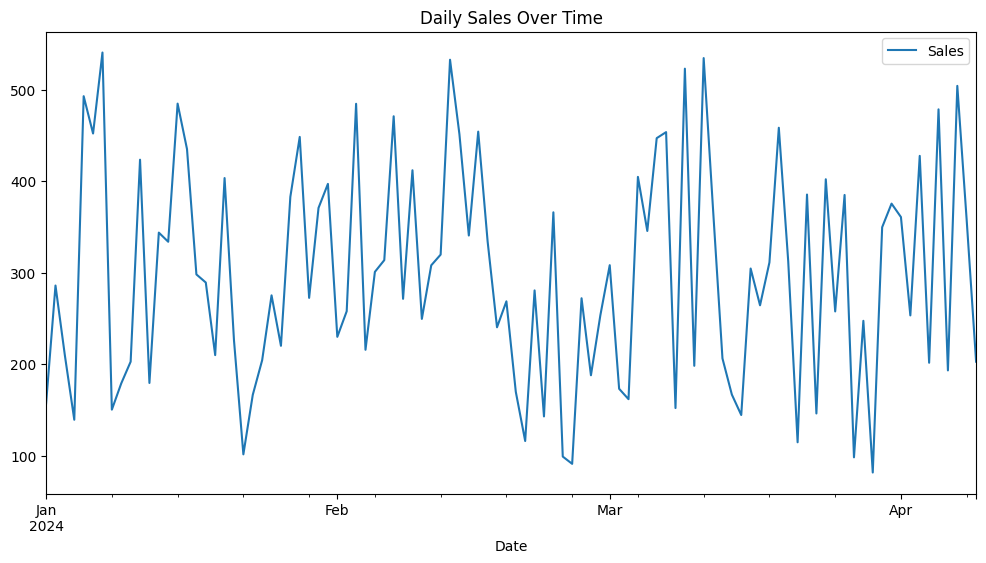
plt.show()

**INPUT:**

* Randomly generated daily sales data for 100 days starting from **2024-01-01**

**EXPECTED / ACTUAL OUTPUT:**

****



**Program No.: 5.4**

**Aim:**

Create a 3D plot.

**SOURCE CODE:**

import matplotlib.pyplot as plt

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

np.random.seed(42)

x = np.random.rand(50)

y = np.random.rand(50)

z = np.random.rand(50)

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

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(x, y, z)

ax.set\_xlabel('X-axis')

ax.set\_ylabel('Y-axis')

ax.set\_zlabel('Z-axis')

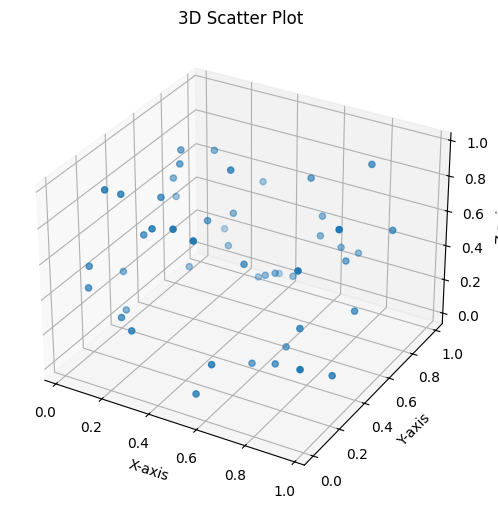
ax.set\_title('3D Scatter Plot')

plt.show()

**INPUT:**

Randomly generated 3D data points for x, y, and z, each containing 50 values between 0 and 1.

**EXPECTED / ACTUAL OUTPUT:**

****