|  |
| --- |
| **Program #1.1 Date :** |
| * 1. **Create a simple calculator in Python.** |

**SOURCE CODE :**

num1 = int(input("Enter Number 1: "))

operator = input("Enter the Operator: ")

num2 = int(input("Enter Number 2: "))

if operator == '+': print(num1 + num2)

elif operator == '-': print(num1 - num2)

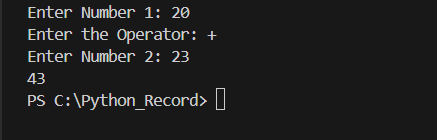
elif operator == '\*': print(num1 \* num2)

elif operator == '/' and num2 != 0:

print(num1/num2)

else: print("Error")

**OUTPUT:**



|  |
| --- |
| **Program #1.2 Date :** |
| An electric power distribution company charges domestic customers as follows: Consumption unit Rate of charge: **1.2.1. 0-200 Rs. 0.50 per unit** 201-400 Rs. 0.65 per unit in excess of 200  * + 1. **401-600 Rs 0.80 per unit excess of 400**  601 and above Rs 1.00per unit excess of 600  * + 1. **If the bill exceeds Rs. 400, then a surcharge of 15% will be charged, and the minimum bill should be Rs. 100/-**  Create a Python program based on the scenario mentioned above. |

**SOURCE CODE:**

charge=0

unit=int(input("enter the unit")) if unit <=200 :

charge=unit\*0.50

elif unit>200 and unit<=400: charge=(200\*0.50)+((unit-200)\*0.65)

elif unit>400 and unit<=600: sur\_charge=((400\*0.60)+((unit-400)\*0.8))\*1.5 charge+=sur\_charge

else:

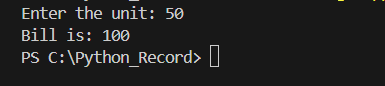
sur\_charge=((600\*0.80)+((unit-600)\*1))\*1.5 charge+=sur\_charge

if charge<100:

print("bill is:100") else:

print("bill is:"+str(charge))

**OUTPUT:**



|  |
| --- |
| **Program #1.3 Date :** |
| Print the pyramid of numbers using for loops. |

**SOURCE CODE :**

startRange = int(33/2)

endRange = int(33/2 -1)

for i in range(1,18):

for j in range(1,33):

if j in range(startRange,endRange):

print("\*",end="")

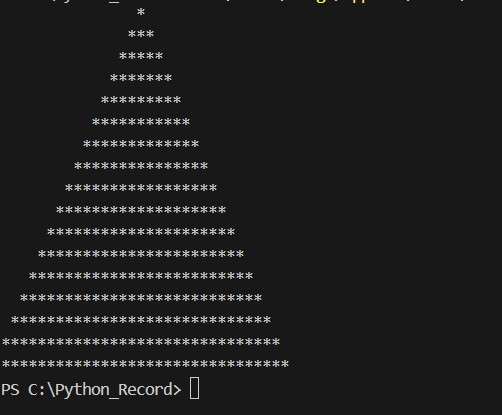
else: print(" ", end="")

startRange -= 1

endRange += 1

print()

**OUTPUT:**



|  |
| --- |
| **Program #1.4 Date :** |
| **Write a program to find the number and sum of all integers greater than 100 and less than 200 that are divisible by 7.** |

**SOURCE CODE :**

sum=0 count=0

for num in range(100,200):

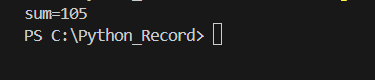
if num%7==0:

count+=1

sum+=count

print("sum="+str(sum))

**OUTPUT:**



|  |
| --- |
| **Program #1.5 Date :** |
| Write a recursive function to calculate the sum of numbers from 0 to 10 |

**SOURCE CODE :**

def sum(number):

if number > 0:

return number + sum(number - 1)

return 0

n = int(input("Number: "))

print("Sum:", sum(n))

**OUTPUT:**



|  |
| --- |
| **Program #1.6 Date :** |
| **Write a Python program to reverse the digits of a given number and add them to the original. If the sum is not a palindrome, repeat this procedure.** |

**SOURCE CODE :**

def reverseNum(num):

rev = 0

while num > 0:

rev \*= 10

rev += (num % 10)

num = num // 10

return rev

def isPalindrome(number):

isPal = True

if number != reverseNum(number):

isPal = False

return isPal

n = int(input("Enter Number: "))

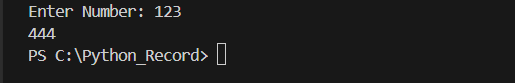
n += reverseNum(n)

while(not isPalindrome(n)):

n += reverseNum(n)

print(n)

**OUTPUT:**



|  |
| --- |
| **Program #1.7 Date :** |
| **Write a menu-driven program that performs the following operations on**  **strings**   * + 1. **Check if the String is a Substring of Another String**     2. **Count Occurrences of Character**     3. **Replace a substring with another substring**     4. **Convert to Capital Letters** |

**SOURCE CODE :**

import os

def checkSubstring():

str1 = input("Enter String 1: ")

str2 = input("Enter String 2: ")

if str2 in str1:

print(str2, "is a substring of", str1)

else:

print(str2, "is not a substring of", str1)

input("Press Enter to continue...")

def countOccurrence():

str\_input = input("Enter a String: ")

ch = input("Enter Character to check: ")

count = str\_input.count(ch)

print(ch, "found", count, "times in", str\_input)

input("Press Enter to continue...")

def replaceSubstring():

str1 = input("Enter String 1: ")

str2 = input("Enter Substring: ")

str3 = input("Enter Substring replacement: ")

if str2 in str1:

str1 = str1.replace(str2, str3)

print("Updated String:", str1)

else:

print(str2, "not found in", str1)

input("Press Enter to continue...")

def toCapital():

str\_input = input("Enter a String: ")

print(str\_input.upper())

input("Press Enter to continue...")

def initMenu():

choice = 0

while choice != 5:

os.system('cls' if os.name == 'nt' else 'clear')

print("1. Check if the String is a Substring of Another String")

print("2. Count Occurrences of Character")

print("3. Replace a Substring with Another Substring")

print("4. Convert to Capital Letters")

print("5. Exit")

choice = int(input("Enter Your Choice: "))

if choice in range(1, 6):

if choice == 1:

checkSubstring()

elif choice == 2:

countOccurrence()

elif choice == 3:

replaceSubstring()

elif choice == 4:

toCapital()

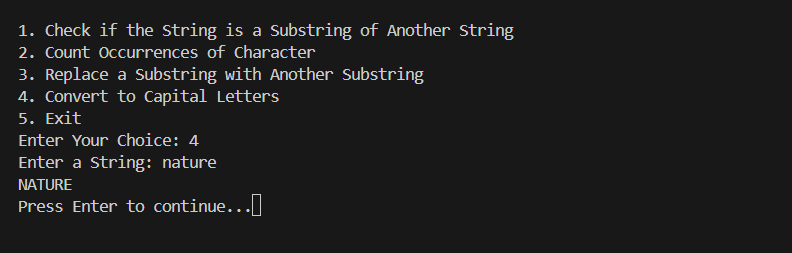
else:

print("Enter a Valid Choice")

input("Press Enter to continue...")

initMenu()

**OUTPUT:**



|  |
| --- |
| **Program #1.8 Date :** |
| **Write a function to find the factorial of a number but also store the factorials calculated in a dictionary.** |

**SOURCE CODE :**

def factorial(n, memo={}):

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

if n == 0 or n == 1:

return 1

if n in memo:

return memo[n]

else:

result = n \* factorial(n - 1, memo)

memo[n] = result

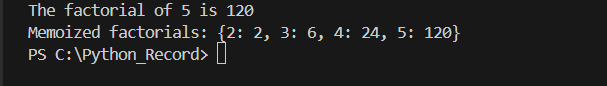
return result

num = 5

print(f"The factorial of {num} is {factorial(num)}")

print(f"Memoized factorials: {factorial.\_\_defaults\_\_[0]}")

**OUTPUT:**



|  |
| --- |
| **Program #1.9 Date :** |
| **Perform various set operations**  **1.9.1 Set Union**  **1.9.2 Set Intersection**  **1.9.3 Set Difference** |

**SOURCE CODE :**

set1 = {1, 2, 3, 4}

set2= {3, 4, 5, 6}

union\_set = set1.union(set2)

print("Union using union() method:", union\_set)

union\_set\_operator = set1 | set2

print("Union using | operator:", union\_set\_operator)

intersection\_set = set1.intersection(set2)

print("Intersection using intersection() method:",intersection\_set)

intersection\_set\_operator = set1 & set2

print("Intersection using & operator:", intersection\_set\_operator)

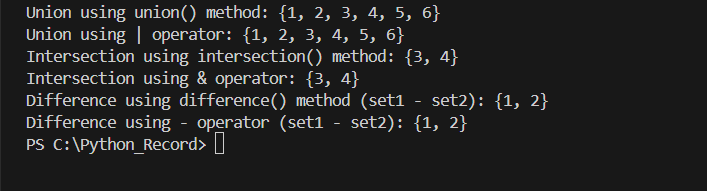
difference\_set = set1.difference(set2)

print("Difference using difference() method (set1 - set2):", difference\_set)

difference\_set\_operator = set1 - set2

print("Difference using - operator (set1 - set2):", difference\_set\_operator)

**OUTPUT:**



|  |
| --- |
| **Program #1.10 Date :** |
| **Create a dictionary to store the name, roll\_no, and total\_mark of N students. Now print the details of the student with the highest total\_mark.** |

**SOURCE CODE :**

students = [

{"name": "Alice", "roll\_no": 101, "total\_mark": 85},

{"name": "Bob", "roll\_no": 102, "total\_mark": 92},

{"name": "Charlie", "roll\_no": 103, "total\_mark": 78},

{"name": "David", "roll\_no": 104, "total\_mark": 95},

{"name": "Eve", "roll\_no": 105, "total\_mark": 88}

]

highest\_mark\_student = max(students, key=lambda student: student["total\_mark"])

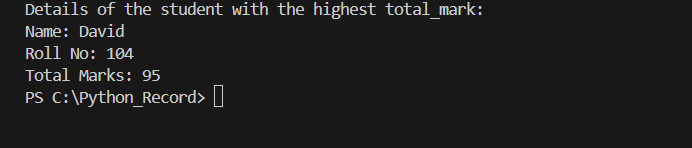
print("Details of the student with the highest total\_mark:")

print(f"Name: {highest\_mark\_student['name']}")

print(f"Roll No: {highest\_mark\_student['roll\_no']}")

print(f"Total Marks: {highest\_mark\_student['total\_mark']}")

**OUTPUT :**



|  |
| --- |
| **Program #1.11 Date :** |
| **Write a Python program to copy the contents of a file into another file, line by line.** |

**SOURCE CODE :**

def copy\_file(source\_file, destination\_file):

try:

with open(source\_file, 'r') as src:

with open(destination\_file, 'w') as dest:

for line in src:

dest.write(line)

print(f"Contents of '{source\_file}' copied to '{destination\_file}' successfully.")

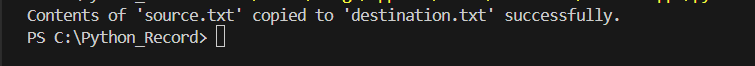
except IOError as e:

print(f"Error copying file: {e}")

source\_file = 'source.txt' destination\_file = 'destination.txt'

copy\_file(source\_file, destination\_file)

**OUTPUT :**



|  |
| --- |
| **Program #1.12 Date :** |
| **Use the OS module to perform**  **1.12.1 Create a directory**   * + 1. **Directory Listing**     2. **Search for “.py” files**     3. **Remove a particular file** |

**SOURCE CODE :**

import os

def create\_directory(directory):

try:

if not os.path.exists(directory):

os.makedirs(directory)

print(f"Directory '{directory}' created successfully.")

else:

print(f"Directory '{directory}' already exists.")

except OSError as e:

print(f"Error creating directory '{directory}': {e}")

def list\_directory(directory):

try:

files = os.listdir(directory)

print(f"Listing of directory '{directory}':")

for file in files:

print(file)

except OSError as e:

print(f"Error listing directory '{directory}': {e}")

def search\_files(directory, extension):

try:

print(f"Searching for files with extension '{extension}' in directory '{directory}':")

for root, \_, files in os.walk(directory):

for file in files:

if file.endswith(extension):

print(os.path.join(root, file))

except OSError as e:

print(f"Error searching files in directory '{directory}': {e}")

def remove\_file(file\_path):

try:

os.remove(file\_path)

print(f"File '{file\_path}' removed successfully.")

except OSError as e:

print(f"Error removing file '{file\_path}': {e}")

def create\_test\_file(file\_path):

try:

with open(file\_path, 'w') as f:

f.write("This is a test file to be removed.\n")

print(f"File '{file\_path}' created successfully.")

except OSError as e:

print(f"Error creating file '{file\_path}': {e}")

# Main script execution

directory\_name = "test\_directory"

file\_extension = ".py"

file\_to\_remove = "test\_directory/file\_to\_remove.txt"

create\_directory(directory\_name)

list\_directory(directory\_name)

search\_files(directory\_name, file\_extension)

# Create the test file

create\_test\_file(file\_to\_remove)

# Check if the file exists before attempting to remove it

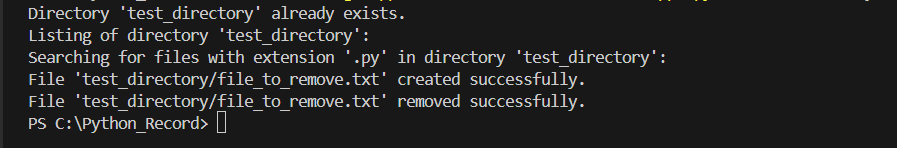
if os.path.exists(file\_to\_remove):

remove\_file(file\_to\_remove)

else:

print(f"File '{file\_to\_remove}' does not exist, so it cannot be removed.")

**OUTPUT :**



|  |
| --- |
| **Program #1.13 Date :** |
| **Create a simple banking application by using inheritance.** |

**SOURCE CODE :**

class Account:

def init (self, account\_number, balance=0):

self.account\_number = account\_number

self.balance = balance

def deposit(self, amount):

self.balance += amount

print(f"Deposited ${amount}. New balance is ${self.balance}.")

def withdraw(self, amount):

if self.balance >= amount:

self.balance -= amount

print(f"Withdrew ${amount}. New balance is ${self.balance}.")

else:

print("Insufficient funds.")

def display\_balance(self):

print(f"Account Number: {self.account\_number}")

print(f"Current Balance: ${self.balance}")

class SavingsAccount(Account):

def init (self, account\_number, balance=0, interest\_rate=0.01):

super(). init (account\_number, balance)

self.interest\_rate = interest\_rate

def add\_interest(self):

interest = self.balance \* self.interest\_rate

self.balance += interest

print(f"Interest added. New balance is ${self.balance}.")

class CheckingAccount(Account):

def init (self, account\_number, balance=0, transaction\_fee=1):

super(). init (account\_number, balance)

self.transaction\_fee = transaction\_fee

def deduct\_transaction\_fee(self):

self.balance -= self.transaction\_fee

print(f"Deducted transaction fee of ${self.transaction\_fee}. New balance is ${self.balance}.")

savings\_acc = SavingsAccount("SAV123", 1000, 0.02)

checking\_acc = CheckingAccount("CHK456", 500)

savings\_acc.display\_balance()

savings\_acc.deposit(500)

savings\_acc.add\_interest()

savings\_acc.withdraw(200)

print()

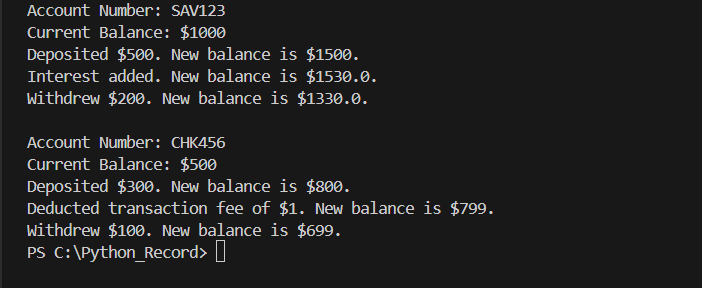
checking\_acc.display\_balance()

checking\_acc.deposit(300)

checking\_acc.deduct\_transaction\_fee()

checking\_acc.withdraw(100)

**OUTPUT :**



|  |
| --- |
| **Program #2.1 Date :** |
| **Implementation of MySQL connection using Python** |

**SOURCE CODE:**

#pip install mysql-connector-python "install myqsl using this command"

import sys

sys.path.append('C:\\Users\\vargh\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.12\_qbz5n2kfra8p0\\LocalCache\\local-packages\\python312\\site-packages')

import mysql.connector

from mysql.connector import Error

# Function to create a MySQL connection

def create\_connection(host\_name, user\_name, user\_password, db\_name):

connection = None

try:

connection = mysql.connector.connect(

host=host\_name,

user=user\_name,

passwd=user\_password,

database=db\_name

)

print("Connection to MySQL DB successful")

except Error as e:

print(f"The error '{e}' occurred")

return connection

# Function to execute a single query (e.g., creating a table)

def execute\_query(connection, query):

cursor = connection.cursor()

try:

cursor.execute(query)

connection.commit()

print("Query executed successfully")

except Error as e:

print(f"The error '{e}' occurred")

# Function to execute a read query (e.g., selecting data)

def execute\_read\_query(connection, query):

cursor = connection.cursor()

result = None

try:

cursor.execute(query)

result = cursor.fetchall()

return result

except Error as e:

print(f"The error '{e}' occurred")

# Step 3: Create Connection

connection = create\_connection("localhost", "root", "", "new\_mysql")

# Step 4: Create a Table

create\_table\_query = """

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

age INT,

gender VARCHAR(10),

nationality VARCHAR(50)

);

"""

execute\_query(connection, create\_table\_query)

# Step 5: Insert Data

insert\_user\_query = """

INSERT INTO users (name, age, gender, nationality) VALUES

('John Doe', 28, 'Male', 'USA'),

('Jane Smith', 25, 'Female', 'Canada');

"""

execute\_query(connection, insert\_user\_query)

# Step 6: Select Data

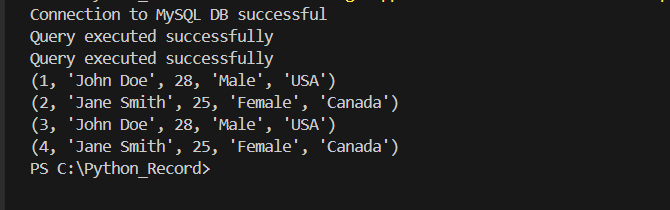
select\_users\_query = "SELECT \* FROM users"

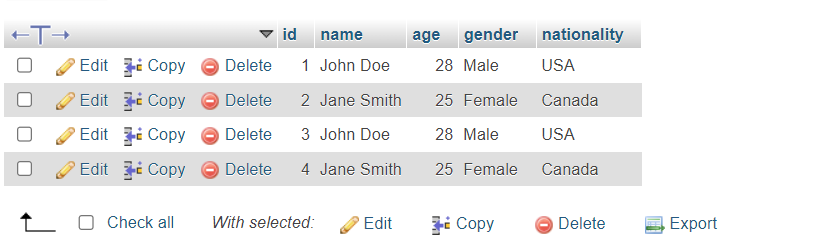
users = execute\_read\_query(connection, select\_users\_query)

for user in users:

print(user)

**OUTPUT :**





|  |
| --- |
| **Program #2.2 Date :** |
| **Implementation of SqLite3 connection using Python** |

**SOURCE CODE:**

import sqlite3

from sqlite3 import Error

# Function to create a connection to the SQLite database

def create\_connection(db\_file):

connection = None

try:

connection = sqlite3.connect(db\_file)

print("Connection to SQLite DB successful")

except Error as e:

print(f"The error '{e}' occurred")

return connection

# Function to execute a single query (CREATE, UPDATE, DELETE)

def execute\_query(connection, query, data=None):

cursor = connection.cursor()

try:

if data:

cursor.execute(query, data)

else:

cursor.execute(query)

connection.commit()

print("Query executed successfully")

except Error as e:

print(f"The error '{e}' occurred")

# Function to execute a read query (SELECT)

def execute\_read\_query(connection, query):

cursor = connection.cursor()

result = None

try:

cursor.execute(query)

result = cursor.fetchall()

return result

except Error as e:

print(f"The error '{e}' occurred")

# Step 2: Create Connection

connection = create\_connection("test\_database.sqlite")

# Step 3: Create a Table

create\_table\_query = """

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

age INTEGER,

gender TEXT,

nationality TEXT

);

"""

execute\_query(connection, create\_table\_query)

# Step 4: Insert Data (Create)

insert\_user\_query = """

INSERT INTO users (name, age, gender, nationality)

VALUES (?, ?, ?, ?)

"""

user\_data = ("John Doe", 28, "Male", "USA")

execute\_query(connection, insert\_user\_query, user\_data)

# Step 5: Select Data (Read)

select\_users\_query = "SELECT \* FROM users"

users = execute\_read\_query(connection, select\_users\_query)

print("Users in the database:")

for user in users:

print(user)

# Step 6: Update Data (Update)

update\_user\_query = """

UPDATE users

SET age = ?

WHERE name = ?

"""

updated\_data = (35, "John Doe")

execute\_query(connection, update\_user\_query, updated\_data)

# Step 7: Delete Data (Delete)

delete\_user\_query = "DELETE FROM users WHERE name = ?"

delete\_user\_data = ("John Doe",)

execute\_query(connection, delete\_user\_query, delete\_user\_data)

# Step 8: Confirm deletion

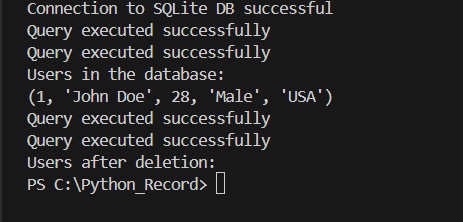
users = execute\_read\_query(connection, select\_users\_query)

print("Users after deletion:")

for user in users:

print(user)

**OUTPUT :**



|  |
| --- |
| **Program #2.3 Date :** |
| **Write a program to implement CRUD operations using Python** |

**SOURCE CODE:**

import sys

sys.path.append('C:\\Users\\vargh\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.12\_qbz5n2kfra8p0\\LocalCache\\local-packages\\python312\\site-packages')

import mysql.connector

from mysql.connector import Error

# Function to create a MySQL connection

def create\_connection(host\_name, user\_name, user\_password, db\_name):

connection = None

try:

connection = mysql.connector.connect(

host=host\_name,

user=user\_name,

passwd=user\_password,

database=db\_name

)

print("Connection to MySQL DB successful")

except Error as e:

print(f"The error '{e}' occurred")

return connection

# Function to execute a single query (CREATE, UPDATE, DELETE)

def execute\_query(connection, query, data=None):

cursor = connection.cursor()

try:

if data:

cursor.execute(query, data)

else:

cursor.execute(query)

connection.commit()

print("Query executed successfully")

except Error as e:

print(f"The error '{e}' occurred")

# Function to execute a read query (SELECT)

def execute\_read\_query(connection, query):

cursor = connection.cursor()

result = None

try:

cursor.execute(query)

result = cursor.fetchall()

return result

except Error as e:

print(f"The error '{e}' occurred")

# Establishing connection

connection = create\_connection("localhost", "root", "", "CRUD")

# Step 1: Create Table

create\_users\_table = """

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

age INT,

gender VARCHAR(10),

nationality VARCHAR(50)

);

"""

execute\_query(connection, create\_users\_table)

# Step 2: Create (Insert Data)

insert\_user\_query = """

INSERT INTO users (name, age, gender, nationality) VALUES (%s, %s, %s, %s)

"""

user\_data = ("Alice", 30, "Female", "USA")

execute\_query(connection, insert\_user\_query, user\_data)

# Step 3: Read (Select Data)

select\_users\_query = "SELECT \* FROM users"

users = execute\_read\_query(connection, select\_users\_query)

print("Users in the database:")

for user in users:

print(user)

# Step 4: Update (Modify Data)

update\_user\_query = """

UPDATE users

SET age = %s

WHERE name = %s

"""

updated\_data = (35, "Alice")

execute\_query(connection, update\_user\_query, updated\_data)

# Step 5: Read again to confirm update

users = execute\_read\_query(connection, select\_users\_query)

print("Users after update:")

for user in users:

print(user)

# Step 6: Delete (Remove Data)

delete\_user\_query = "DELETE FROM users WHERE name = %s"

delete\_user\_data = ("Alice",)

execute\_query(connection, delete\_user\_query, delete\_user\_data)

# Step 7: Read again to confirm deletion

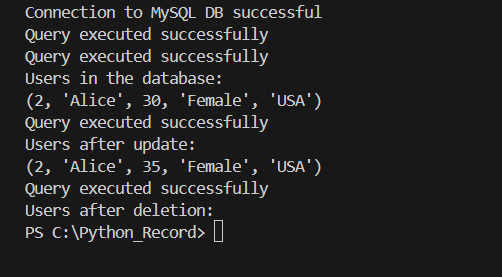
users = execute\_read\_query(connection, select\_users\_query)

print("Users after deletion:")

for user in users:

print(user)

**OUTPUT :**



|  |
| --- |
| **Program #3.1 Date :** |
| **Create a Login form using CGI and display the input on a different page.** |

**SOURCE CODE:**

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Login</title>

<style>

body{

display: flex;

flex-direction: column;

gap: 1rem;

align-items: center;

justify-content: center;

height: 88svh;

font-family: sans-serif;

}

form{

display: flex;

flex-direction: column;

gap: .725rem;

}

</style>

</head>

<body>

<h1>LOGIN</h1>

<form action="dbHandler.py" method="get">

<input type="text" id="username" name="username" placeholder="username">

<input type="password" id="password" name="password" placeholder="password">

<input type="submit" value="Submit">

</form>

</body>

</html>

Dbhandler.py

#!C:\wamp64\www\python-cgi\venv\Scripts\python.exe

import cgi

import cgitb

import MySQLdb

cgitb.enable()

try:

myDb = MySQLdb.connect(host="localhost",user="root",password="",db="test")

myCursor = myDb.cursor()

form = cgi.FieldStorage()

username = form.getvalue('username')

password = form.getvalue('password')

sql = "INSERT INTO user (`username`, `password`) VALUES(%s,%s)"

myCursor.execute(sql,(username,password))

myDb.commit()

print("Content-type:text/html")

print()

print(f'''<!DOCTYPE html>

<html lang='en'>

<head>

<meta charset='UTF-8'>

<meta name='viewport' content='width=device-width, initial-scale=1.0'>

<title>Welcome</title>

</head>

<body>

<center>

<h1>Welcome, {username}!</h1>

<p>Your login was successful.</p>

<a href='index.html'>Back to Login</a>

</center>

</body>

</html>''')

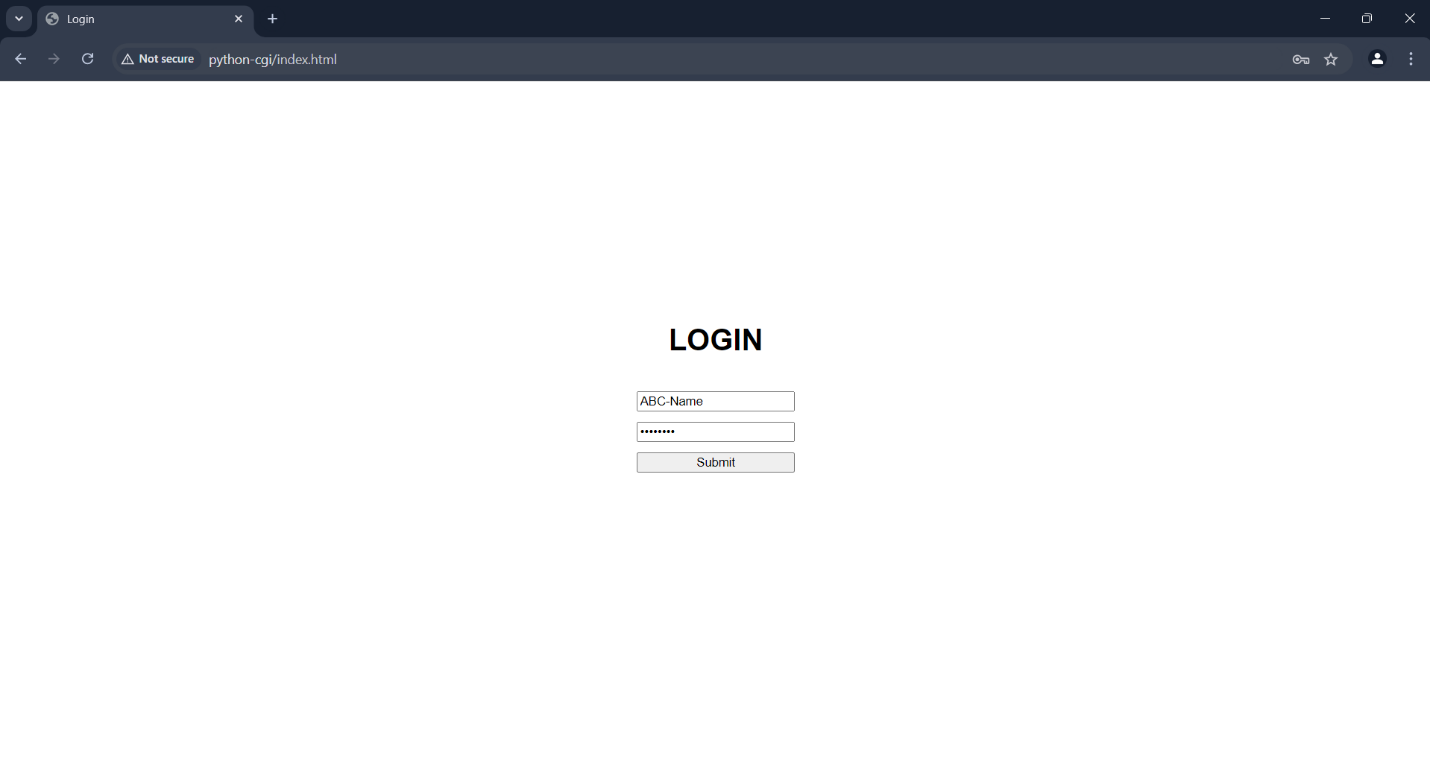
except Exception as e:

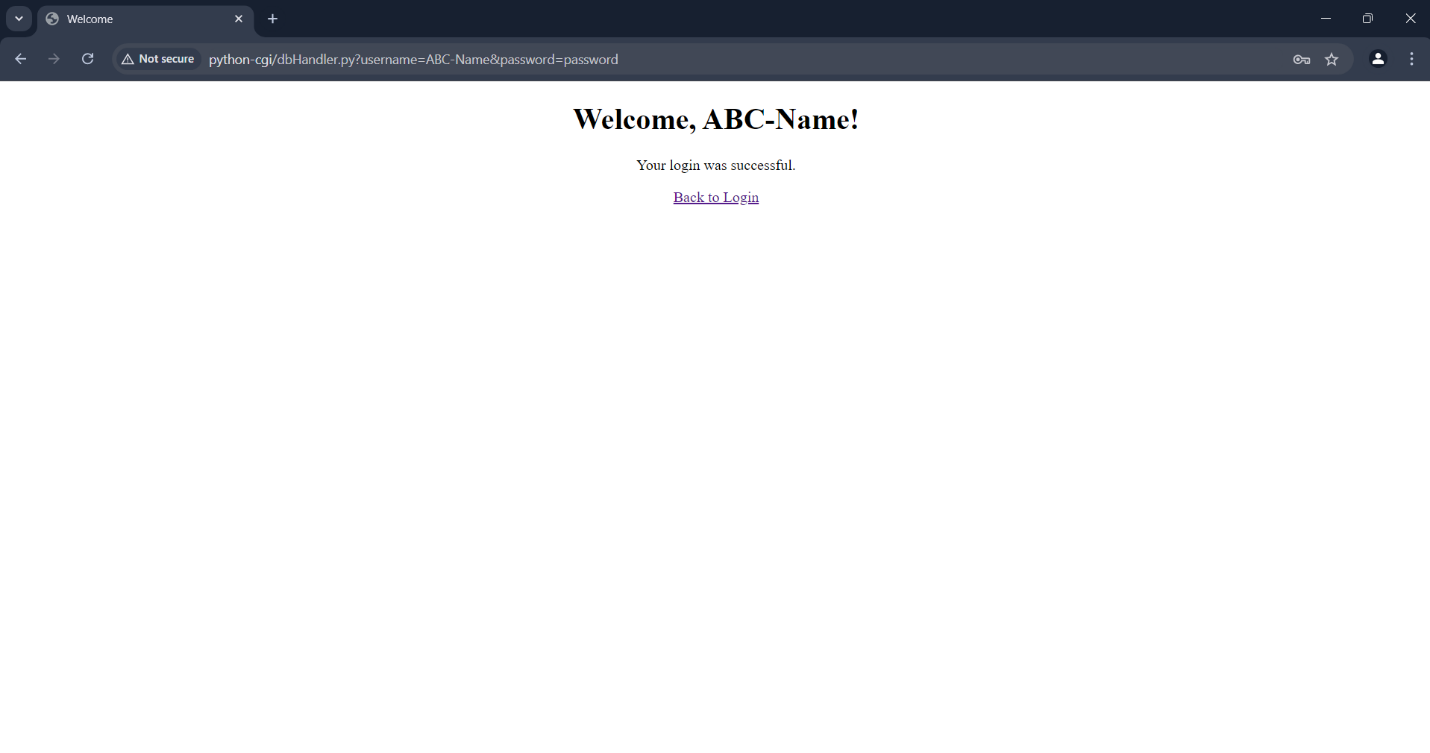
print(e)

finally:

myDb.close()

**OUTPUT :**





|  |
| --- |
| **Program #3.2 Date :** |
| **Create a registration form for MCA admission and display the inserted data on the web page.** |

**SOURCE CODE:**

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>MCA | ADMISSION</title>

<style>

body{

display: flex;

flex-direction: column;

gap: 1rem;

align-items: center;

justify-content: center;

height: 88svh;

font-family: sans-serif;

}

h1{

color: green;

}

form{

display: flex;

flex-direction: column;

gap: .725rem;

}

</style>

</head>

<body>

<h1>REGISTER - MCA admission 2024</h1>

<form action="mcaHandler.py" method="get">

<input type="email" id="email" name="email" placeholder="email">

<input type="text" id="username" name="username" placeholder="username">

<input type="password" id="password" name="password" placeholder="password">

<input type="submit" value="register">

</form>

</body>

</html>

Db-handler.py

#!C:\wamp64\www\python-cgi\venv\Scripts\python.exe

import cgi

import cgitb

import MySQLdb

cgitb.enable()

try:

myDb = MySQLdb.connect(host="localhost",user="root",password="",db="test")

myCursor = myDb.cursor()

form = cgi.FieldStorage()

username = form.getvalue('username')

password = form.getvalue('password')

email = form.getvalue("email")

sql = "INSERT INTO user (`username`, `password`) VALUES(%s,%s)"

myCursor.execute(sql,(username,password))

myDb.commit()

print("Content-type:text/html")

print()

print(f'''<!DOCTYPE html>

<html lang='en'>

<head>

<meta charset='UTF-8'>

<meta name='viewport' content='width=device-width, initial-scale=1.0'>

<title>Welcome</title>

</head>

<body>

<center>

<h1>Welcome, {username}!</h1>

<p>Your Registration was successful! check verification sent on {email}</p>

<p style='color:blue;'>Continue With registration</p>

</center>

</body>

</html>''')

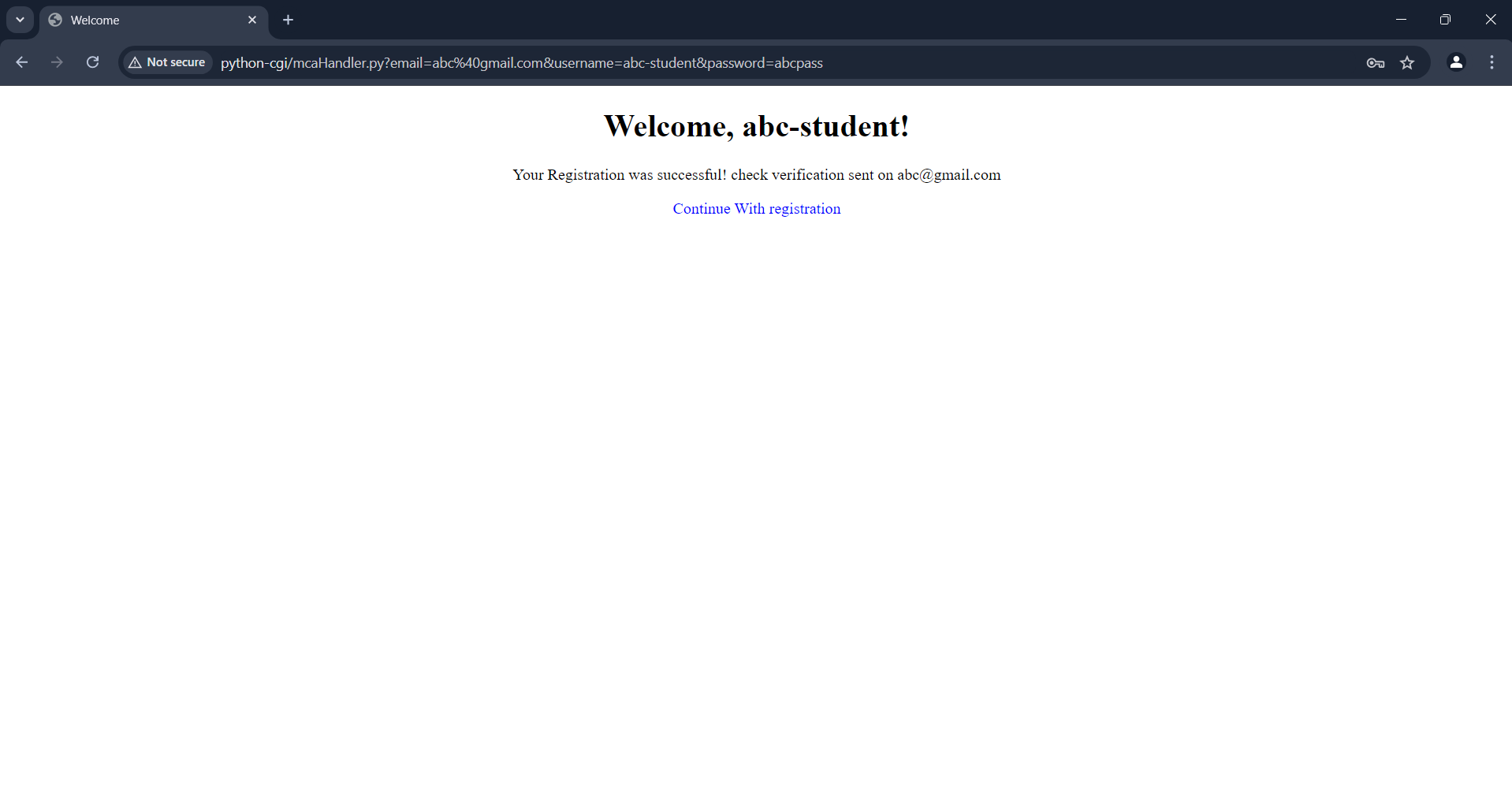
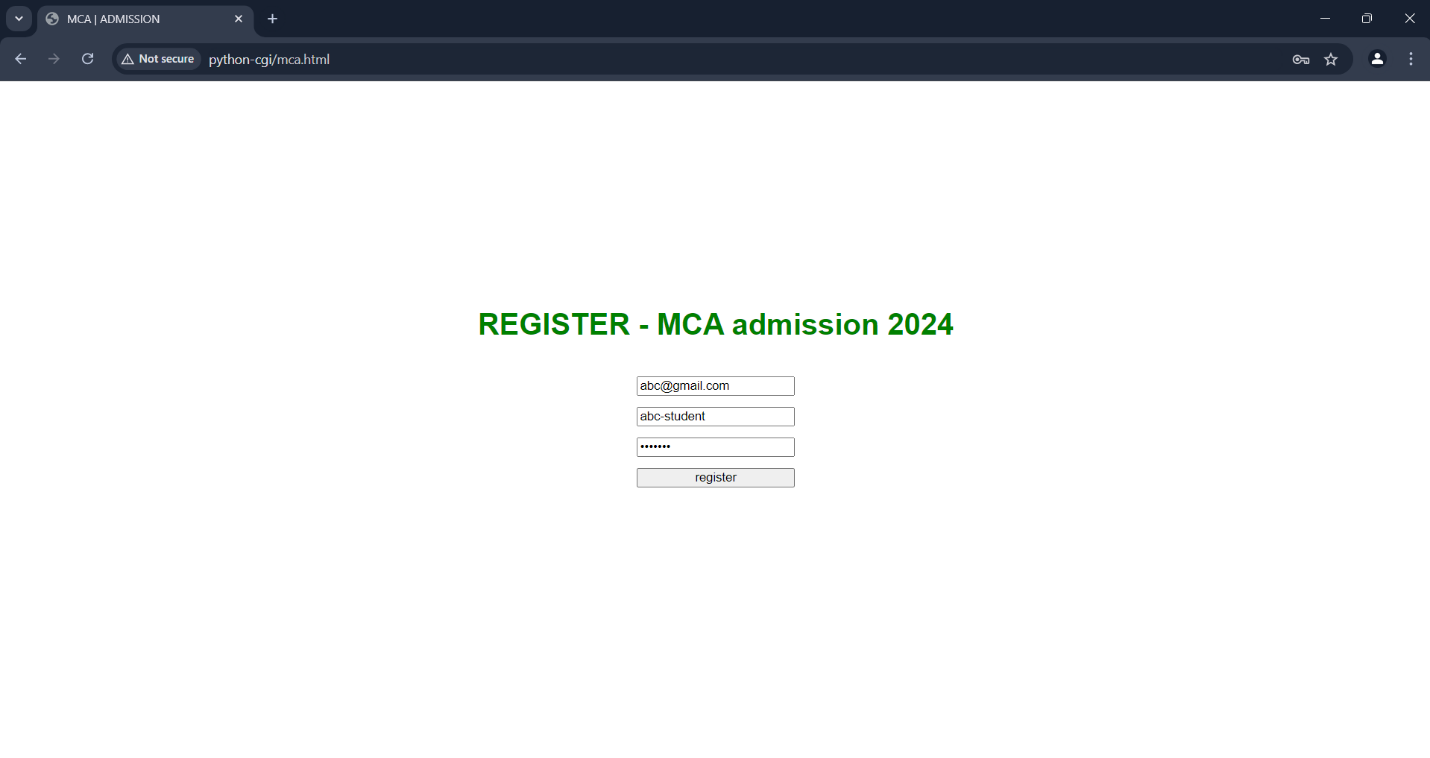
except Exception as e:

print(e)

finally:

myDb.close()

**OUTPUT:**



|  |
| --- |
| **Program #3.3 Date :** |
| **Create a MySQL database and perform INSERT, UPDATE, DESTROY, and SELECT (display) operations using the CGI interface.** |

**SOURCE CODE:**

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CRUD Operations</title>

<style>

body {

display: flex;

flex-direction: column;

gap: 1rem;

align-items: center;

justify-content: center;

height: 88svh;

font-family: sans-serif;

}

.content {

display: flex;

flex-wrap: wrap;

gap: 10rem;

align-items: center;

justify-content: center;

font-family: sans-serif;

}

form {

display: flex;

flex-direction: column;

gap: .725rem;

}

</style>

</head>

<body>

<h1>CRUD OPERATIONS

<hr>

</h1>

<div class="content">

<div>

<h2>Create</h2>

<form action="crud.py" method="post">

<input type="text" name="username" placeholder="Username" required>

<input type="password" name="password" placeholder="Password" required>

<input type="submit" name="action" value="Create">

</form>

</div>

<div>

<h2>Read</h2>

<form action="crud.py" method="post">

<input type="text" name="username" placeholder="Username to display" required>

<input type="submit" name="action" value="Read">

</form>

</div>

<div>

<h2>Update</h2>

<form action="crud.py" method="post">

<input type="text" name="old\_username" placeholder="Current Username" required>

<input type="text" name="new\_username" placeholder="New Username" required>

<input type="password" name="new\_password" placeholder="New Password" required>

<input type="submit" name="action" value="Update">

</form>

</div>

<div>

<h2>Delete</h2>

<form action="crud.py" method="post">

<input type="text" name="username" placeholder="Username to delete" required>

<input type="submit" name="action" value="Delete">

</form>

</div>

</div>

<div id="result"></div>

</body>

</html>

Db-handler.py

#!C:\wamp64\www\python-cgi\venv\Scripts\python.exe

import cgi

import cgitb

import MySQLdb

cgitb.enable()

def connect\_db():

return MySQLdb.connect(host="localhost", user="root", password="", db="test")

def create\_user(cursor, username, password):

sql = "INSERT INTO user (`username`, `password`) VALUES (%s, %s)"

cursor.execute(sql, (username, password))

def read\_user(cursor, username):

sql = "SELECT \* FROM user WHERE username = %s"

cursor.execute(sql, (username,))

return cursor.fetchall()

def update\_user(cursor, old\_username, new\_username, new\_password):

sql = "UPDATE user SET username = %s, password = %s WHERE username = %s"

cursor.execute(sql, (new\_username, new\_password, old\_username))

def delete\_user(cursor, username):

sql = "DELETE FROM user WHERE username = %s"

cursor.execute(sql, (username,))

print("Content-type:text/html")

print()

try:

form = cgi.FieldStorage()

myDb = connect\_db()

myCursor = myDb.cursor()

action = form.getvalue('action')

if action == 'Create':

username = form.getvalue('username')

password = form.getvalue('password')

create\_user(myCursor, username, password)

myDb.commit()

print(f"<p>User {username} created successfully!</p>")

elif action == 'Read':

username = form.getvalue('username')

user\_data = read\_user(myCursor, username)

if user\_data:

for row in user\_data:

print(f"<p>Username: {row[0]}, Password: {row[1]}</p>")

else:

print("<p>No user found.</p>")

elif action == 'Update':

old\_username = form.getvalue('old\_username')

new\_username = form.getvalue('new\_username')

new\_password = form.getvalue('new\_password')

update\_user(myCursor, old\_username, new\_username, new\_password)

myDb.commit()

print(f"<p>User {old\_username} updated successfully!</p>")

elif action == 'Delete':

username = form.getvalue('username')

delete\_user(myCursor, username)

myDb.commit()

print(f"<p>User {username} deleted successfully!</p>")

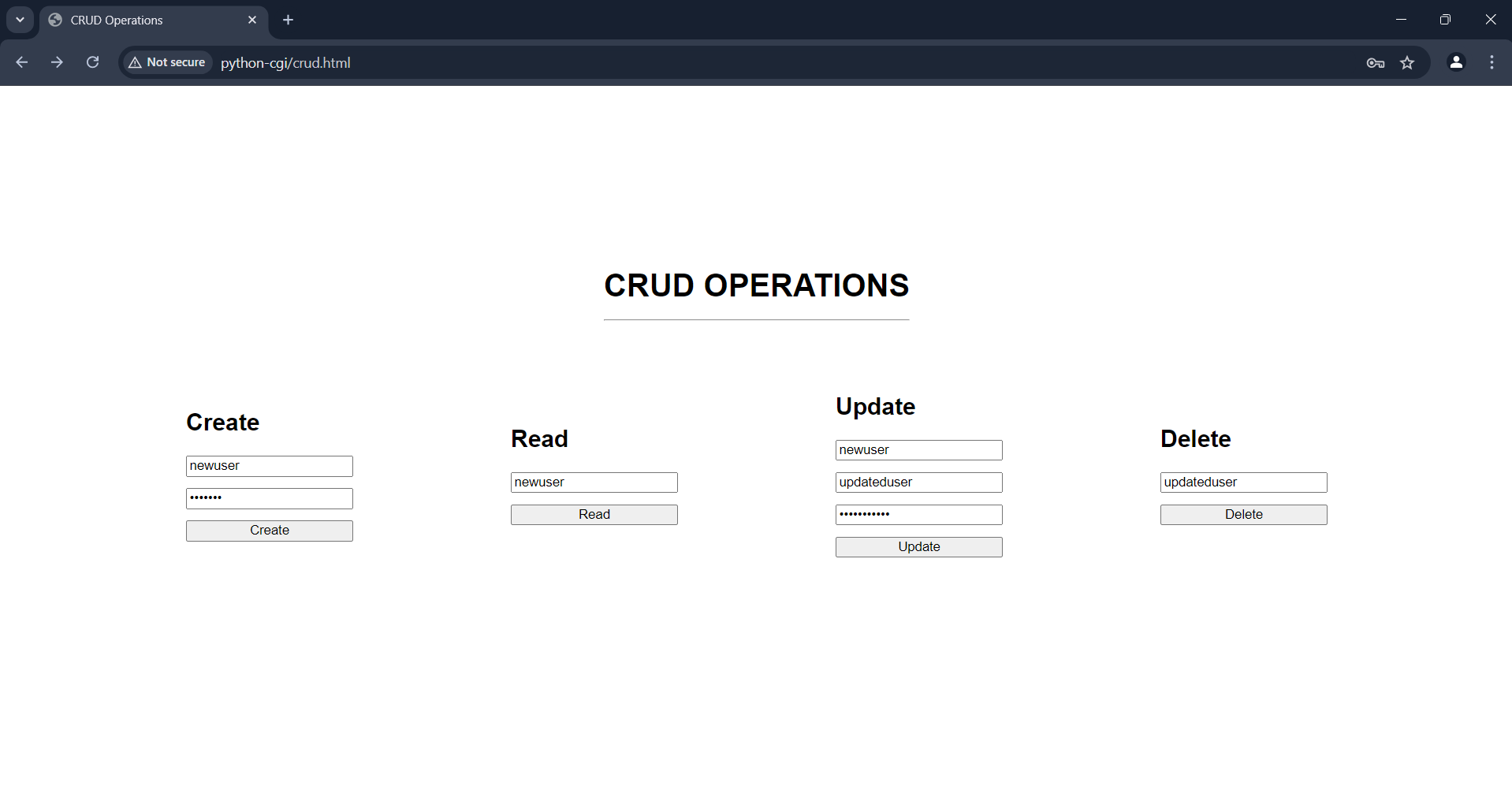
except Exception as e:

print(f"<p>Error: {e}</p>")

finally:

myDb.close()

**OUTPUT:**



|  |
| --- |
| **Program #4.1.1 Date :** |
| **Create a numpy array filled with all ones by defining its shape.** |

**SOURCE CODE:**

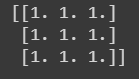
import numpy as np

shape=(3,3)

ones\_array=np.ones(shape)

print(ones\_array)

**OUTPUT­:**



|  |
| --- |
| **Program #4.1.2 Date :** |
| **How do you remove rows from a Numpy array that contains non-numeric values?** |

**SOURCE CODE:**

import numpy as np

arr=np.array([

[1,2,3,4],

['a','b',4,7],

[11,12,13,14]

],dtype=object)

def safe\_convert(value):

try:

return float(value)

except ValueError:

return np.nan

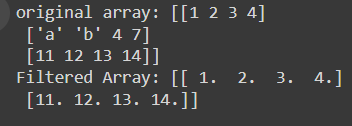
arr\_numeric =np.vectorize(safe\_convert)(arr)

filtered\_arr=arr\_numeric[~np.isnan(arr\_numeric).any(axis=1)]

print("original array:",arr)

print("Filtered Array:",filtered\_arr)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.3 Date :** |
| **Remove single-dimensional entries from the shape of an array** |

**SOURCE CODE:**

import numpy as np

# Example array with single-dimensional entries

arr = np.array([[[1], [2], [3]]])

print("Original array shape:", arr.shape)

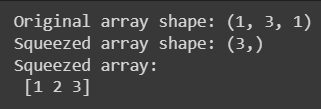
# Remove single-dimensional entries from the shape

squeezed\_arr = np.squeeze(arr)

print("Squeezed array shape:", squeezed\_arr.shape)

print("Squeezed array:\n", squeezed\_arr)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.4 Date :** |
| **How do you check whether specified values are present in the NumPy array?** |

**SOURCE CODE:**

import numpy as np

# Create a NumPy array

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Values to check

values\_to\_check = [3, 5, 10]

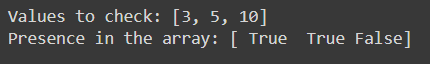
# Check for presence of specified values

presence = np.isin(values\_to\_check, arr)

print("Values to check:", values\_to\_check)

print("Presence in the array:", presence)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.5 Date :** |
| **Write a program to get all 2D diagonals of a 3D NumPy array?** |

**SOURCE CODE:**

import numpy as np

# Create a 3D NumPy array

array\_3d = np.array([

[[1, 2, 3],

[4, 5, 6],

[7, 8, 9]],

[[10, 11, 12],

[13, 14, 15],

[16, 17, 18]],

[[19, 20, 21],

[22, 23, 24],

[25, 26, 27]]

])

# Function to get all 2D diagonals of a 3D NumPy array

def get\_2d\_diagonals(arr):

diagonals = []

# Iterate over the first dimension of the 3D array

for i in range(arr.shape[0]):

# Get the diagonal of each 2D slice

diag = np.diagonal(arr[i], axis1=0, axis2=1)

diagonals.append(diag)

return diagonals

# Get all 2D diagonals

diagonals\_2d = get\_2d\_diagonals(array\_3d)

# Print the results

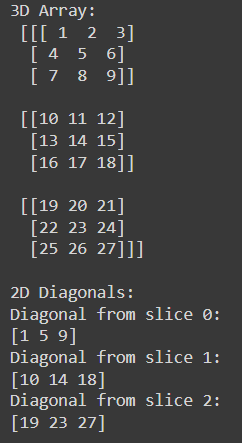
print("3D Array:\n", array\_3d)

print("\n2D Diagonals:")

for index, diag in enumerate(diagonals\_2d):

print(f"Diagonal from slice {index}:\n{diag}")

**OUTPUT:**



|  |
| --- |
| **Program #4.1.6 Date :** |
| **Write a NumPy program to sort a given array of shape 2 along the first axis, last axis, and flattened array.** |

**SOURCE CODE:**

import numpy as np

# Create a sample array of shape (2, 3)

array = np.array([[3, 1, 2],

[6, 4, 5]])

print("Original array:\n", array)

# Sort along the first axis (axis=0)

sorted\_first\_axis = np.sort(array, axis=0)

print("\nSorted along the first axis (axis=0):\n", sorted\_first\_axis)

# Sort along the last axis (axis=1)

sorted\_last\_axis = np.sort(array, axis=1)

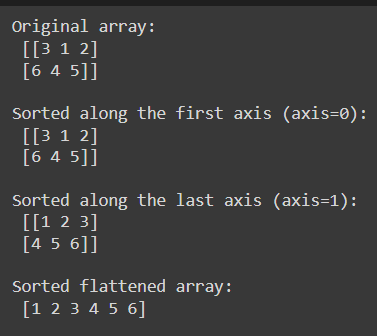
print("\nSorted along the last axis (axis=1):\n", sorted\_last\_axis)

# Flatten the array and sort

flattened\_sorted = np.sort(array.flatten())

print("\nSorted flattened array:\n", flattened\_sorted)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.7 Date :** |
| **Write a NumPy program to create a structured array from a given student name, height, class, and data type. Now sort by class, then height if the classes are equal.** |

**SOURCE CODE:**

import numpy as np

# Define the data type for the structured array

dtype = [('name', 'U20'), ('height', 'f4'), ('class', 'i4')]

# Create a structured array with student data

student\_data = np.array([

('Alice', 5.5, 2),

('Bob', 6.0, 1),

('Charlie', 5.7, 2),

('David', 5.9, 1),

('Eve', 5.4, 2)

], dtype=dtype)

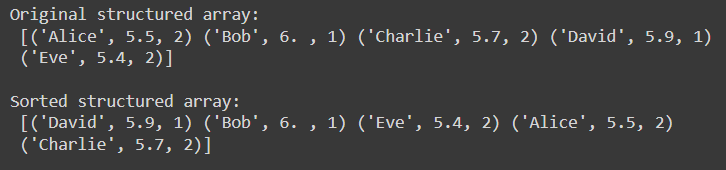
print("Original structured array:\n", student\_data)

# Sort the structured array by 'class' and then by 'height'

sorted\_students = np.sort(student\_data, order=['class', 'height'])

print("\nSorted structured array:\n", sorted\_students)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.8 Date :** |
| **Write a NumPy program to sort a given complex array using the real part first, then the imaginary part.** |

**SOURCE CODE:**

import numpy as np

# Create a complex NumPy array

complex\_array = np.array([3 + 2j, 1 + 4j, 2 + 3j, 1 + 2j, 2 + 1j])

print("Original complex array:\n", complex\_array)

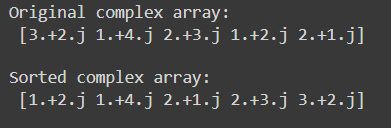
# Sort by real part, then by imaginary part

sorted\_indices = np.lexsort((complex\_array.imag, complex\_array.real))

sorted\_complex\_array = complex\_array[sorted\_indices]

print("\nSorted complex array:\n", sorted\_complex\_array)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.9 Date :** |
| **Write a NumPy program to sort a given array by the nth column.** |

**SOURCE CODE:**

import numpy as np

# Create a sample 2D NumPy array

array = np.array([[3, 1, 2],

[6, 4, 5],

[1, 7, 8],

[9, 0, 3]])

print("Original array:\n", array)

# Specify the column index to sort by (0-indexed)

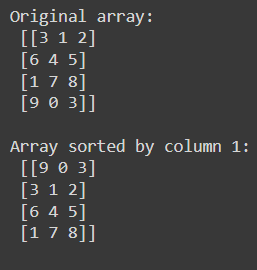
n = 1 # Sort by the second column

# Sort the array by the nth column

sorted\_array = array[array[:, n].argsort()]

print(f"\nArray sorted by column {n}:\n", sorted\_array)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.10 Date :** |
| **Calculate the sum of the diagonal elements of a NumPy array** |

**SOURCE CODE:**

import numpy as np

# Create a sample 2D NumPy array

array = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

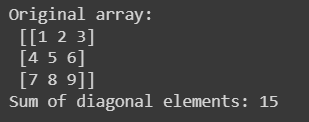
print("Original array:\n", array)

# Calculate the sum of the diagonal elements

diagonal\_sum = np.sum(np.diagonal(array))

print("Sum of diagonal elements:", diagonal\_sum)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.11 Date :** |
| **Write a program for Matrix Multiplication in NumPy** |

**SOURCE CODE:**

import numpy as np

# Create two sample matrices

matrix\_a = np.array([[1, 2, 3],

[4, 5, 6]])

matrix\_b = np.array([[7, 8],

[9, 10],

[11, 12]])

print("Matrix A:\n", matrix\_a)

print("\nMatrix B:\n", matrix\_b)

# Method 1: Using numpy.dot()

result\_dot = np.dot(matrix\_a, matrix\_b)

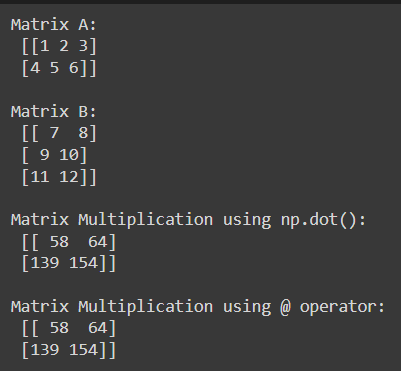
print("\nMatrix Multiplication using np.dot():\n", result\_dot)

# Method 2: Using the @ operator

result\_at = matrix\_a @ matrix\_b

print("\nMatrix Multiplication using @ operator:\n", result\_at)

**OUTPUT**



|  |
| --- |
| **Program #4.1.12 Date :** |
| **Multiply matrices of complex numbers using NumPy in Python.** |

**SOURCE CODE:**

import numpy as np

# Create two sample matrices with complex numbers

matrix\_a = np.array([[1 + 2j, 2 + 3j],

[3 + 4j, 4 + 5j]])

matrix\_b = np.array([[5 + 6j, 6 + 7j],

[7 + 8j, 8 + 9j]])

print("Matrix A:\n", matrix\_a)

print("\nMatrix B:\n", matrix\_b)

# Perform matrix multiplication

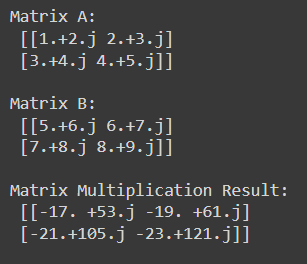
result = np.dot(matrix\_a, matrix\_b)

# Alternatively, you can use the @ operator

# result = matrix\_a @ matrix\_b

print("\nMatrix Multiplication Result:\n", result)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.13 Date :** |
| **Calculate the inner, outer, and cross products of matrices and vectors using NumPy.** |

**SOURCE CODE:**

import numpy as np

# Define two vectors

vector\_a = np.array([1, 2, 3])

vector\_b = np.array([4, 5, 6])

# Calculate the inner product

inner\_product = np.inner(vector\_a, vector\_b)

print("Inner Product of vector\_a and vector\_b:", inner\_product)

# Calculate the outer product

outer\_product = np.outer(vector\_a, vector\_b)

print("\nOuter Product of vector\_a and vector\_b:\n", outer\_product)

# Calculate the cross product

cross\_product = np.cross(vector\_a, vector\_b)

print("\nCross Product of vector\_a and vector\_b:", cross\_product)

# Define two matrices

matrix\_a = np.array([[1, 2],

[3, 4]])

matrix\_b = np.array([[5, 6],

[7, 8]])

# Calculate the inner product of matrices (dot product)

matrix\_inner\_product = np.dot(matrix\_a, matrix\_b)

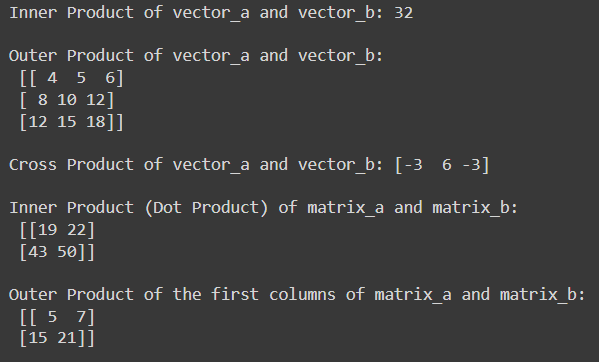
print("\nInner Product (Dot Product) of matrix\_a and matrix\_b:\n", matrix\_inner\_product)

# Calculate the outer product of the first column of matrix\_a and the first column of matrix\_b

outer\_product\_matrix = np.outer(matrix\_a[:, 0], matrix\_b[:, 0])

print("\nOuter Product of the first columns of matrix\_a and matrix\_b:\n", outer\_product\_matrix)

**OUTPUT:**

****

|  |
| --- |
| **Program #4.1.14 Date :** |
| **Compute the covariance matrix of two given NumPy arrays.** |

**SOURCE CODE:**

import numpy as np

# Define two sample NumPy arrays (samples of two variables)

data1 = np.array([1, 2, 3, 4, 5])

data2 = np.array([5, 4, 3, 2, 1])

# Stack the arrays vertically to create a 2D array

data = np.vstack((data1, data2))

# Compute the covariance matrix

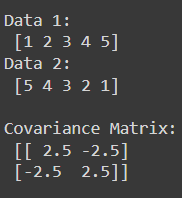
covariance\_matrix = np.cov(data)

print("Data 1:\n", data1)

print("Data 2:\n", data2)

print("\nCovariance Matrix:\n", covariance\_matrix)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.15 Date :** |
| **Convert covariance matrix to correlation matrix using Python.** |

**SOURCE CODE:**

import numpy as np

# Define a sample covariance matrix

covariance\_matrix = np.array([[4, 2],

[2, 3]])

print("Covariance Matrix:\n", covariance\_matrix)

# Calculate the standard deviations for each variable

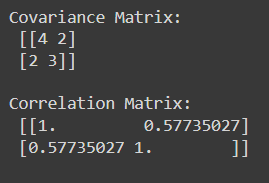
std\_devs = np.sqrt(np.diagonal(covariance\_matrix))

# Create a correlation matrix

correlation\_matrix = covariance\_matrix / std\_devs[:, None] / std\_devs[None, :]

print("\nCorrelation Matrix:\n", correlation\_matrix)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.16 Date :** |
| **Write a NumPy program to compute the histogram of nums against the bins.** |

**SOURCE CODE**

import numpy as np

import matplotlib.pyplot as plt

# Define a sample array of numbers (nums)

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

# Define the bins

bins = np.array([0, 1, 2, 3, 4, 5, 6, 7])

# Compute the histogram

histogram, bin\_edges = np.histogram(nums, bins)

print("Histogram:", histogram)

print("Bin edges:", bin\_edges)

# Plotting the histogram

plt.hist(nums, bins=bins, edgecolor='black', alpha=0.7)

plt.title("Histogram of nums")

plt.xlabel("Value")

plt.ylabel("Frequency")

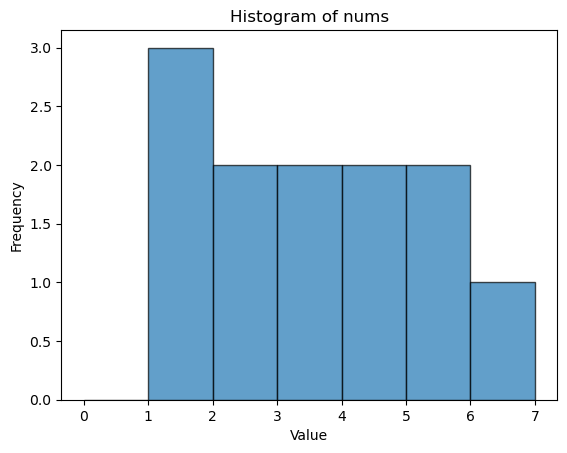
plt.xticks(bin\_edges)

plt.show()

**OUTPUT:**

Histogram: [0 3 2 2 2 2 1]

Bin edges: [0 1 2 3 4 5 6 7]



|  |
| --- |
| **Program #4.1.17 Date :** |
| **Write a NumPy program to compute the cross-correlation of two given arrays.** |

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

# Define two sample arrays

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

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

# Compute the cross-correlation

cross\_correlation = np.correlate(array\_x, array\_y, mode='full')

print("Array X:", array\_x)

print("Array Y:", array\_y)

print("\nCross-Correlation:", cross\_correlation)

# Plotting the cross-correlation

plt.stem(cross\_correlation)

plt.title("Cross-Correlation")

plt.xlabel("Lag")

plt.ylabel("Cross-Correlation Value")

plt.grid()

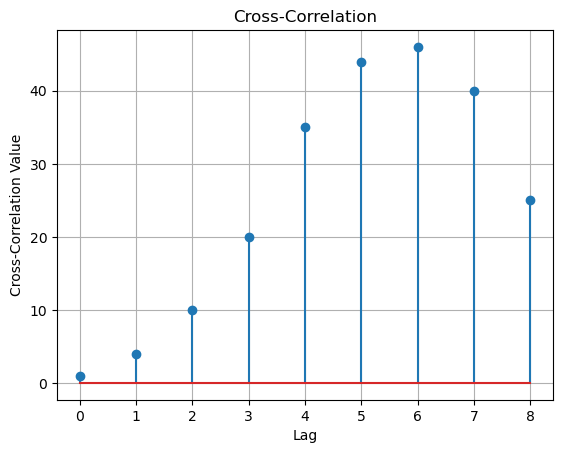
plt.show()

**OUTPUT:**

Array X: [1 2 3 4 5]

Array Y: [5 4 3 2 1]

Cross-Correlation: [ 1 4 10 20 35 44 46 40 25]



|  |
| --- |
| **Program #4.1.18 Date :** |
| **Write a NumPy program to compute the mean, standard deviation, and variance of a given array along the second axis.** |

**SOURCE CODE:**

import numpy as np

# Define a sample 2D array

array = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

print("Original Array:\n", array)

# Compute the mean along the second axis (axis 1)

mean = np.mean(array, axis=1)

print("\nMean along the second axis:", mean)

# Compute the standard deviation along the second axis (axis 1)

std\_deviation = np.std(array, axis=1)

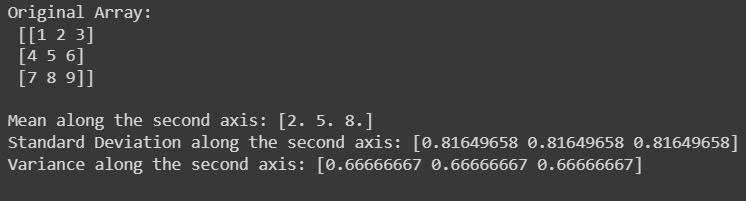
print("Standard Deviation along the second axis:", std\_deviation)

# Compute the variance along the second axis (axis 1)

variance = np.var(array, axis=1)

print("Variance along the second axis:", variance)

**OUTPUT:**



|  |
| --- |
| **Program #4.1.19 Date :** |
| **Write a NumPy program to compute the 80th percentile for all elements in a given array along the second axis.** |

**SOURCE CODE:**

import numpy as np

# Define a sample 2D array

array = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

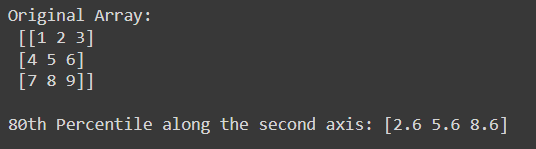
print("Original Array:\n", array)

# Compute the 80th percentile along the second axis (axis 1)

percentile\_80 = np.percentile(array, 80, axis=1)

print("\n80th Percentile along the second axis:", percentile\_80)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.1 Date :** |
| **Write a Pandas program to add, subtract, multiply, and divide two Pandas Series.** |

**SOURCE CODE:**

import pandas as pd

series1 = pd.Series([10, 20, 30, 40, 50])

series2 = pd.Series([5, 10, 15, 20, 25])

addition = series1 + series2

print("Addition of two Series:")

print(addition)

subtraction = series1 - series2

print("\nSubtraction of two Series:")

print(subtraction)

multiplication = series1 \* series2

print("\nMultiplication of two Series:")

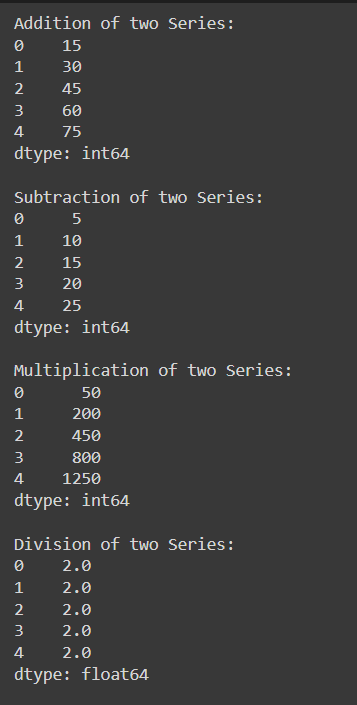
print(multiplication)

division = series1 / series2

print("\nDivision of two Series:")

print(division)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.2 Date :** |
| **Write a Pandas program to convert a dictionary to a Pandas series.** |

**SOURCE CODE:**

import pandas as pd

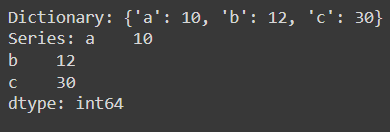
my\_dict={'a':10,'b':12,'c':30}

print("Dictionary:",my\_dict)

my\_series=pd.Series(my\_dict)

print("Series:",my\_series)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.3 Date :** |
| **Write a Pandas program to convert the first column of a data frame into a Series.** |

**SOURCE CODE:**

import pandas as pd

# Sample DataFrame

my\_data = {'Column1': [10, 20, 30, 40], 'Column2': [100, 200, 300, 400]}

df = pd.DataFrame(my\_data)

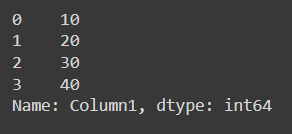
# Convert the first column into a Series

series = df.iloc[:, 0]

# Display the Series

print(series)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.4 Date :** |
| **Write a Pandas program to convert a Series of lists into one Series.** |

**SOURCE CODE:**

import pandas as pd

# Sample Series of lists

series\_of\_lists = pd.Series([[1, 2, 3], [4, 5], [6, 7, 8]])

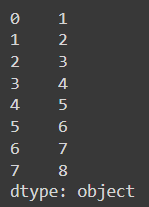
# Convert Series of lists into one Series

flattened\_series = series\_of\_lists.explode().reset\_index(drop=True)

# Display the result

print(flattened\_series)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.5 Date :** |
| **Write a Pandas program to create a subset of a given series based on value and condition.** |

**SOURCE CODE:**

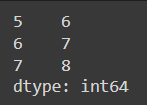
import pandas as pd

data=pd.Series([1,2,3,4,5,6,7,8])

subset=data[data>5]

print(subset)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.6 Date :** |
| **Write a Pandas program to get the items that are not common in two given series.** |

**SOURCE CODE:**

import pandas as pd

# Sample Series

series1 = pd.Series([1, 2, 3, 4, 5])

series2 = pd.Series([4, 5, 6, 7, 8])

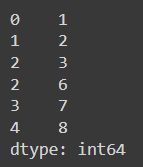
# Get items that are not common in both series

uncommon\_items = pd.concat([series1[~series1.isin(series2)], series2[~series2.isin(series1)]])

# Display the result

print(uncommon\_items)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.7 Date :** |
| **Write a Pandas program to calculate the frequency counts of each unique value of a given series.** |

**SOURCE CODE:**

import pandas as pd

# Sample Series

data = pd.Series([1, 2, 2, 3, 4, 4, 4, 5, 5])

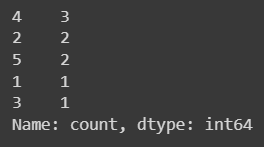
# Calculate frequency counts of each unique value

frequency\_counts = data.value\_counts()

# Display the result

print(frequency\_counts)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.8 Date :** |
| **Write a Pandas program to filter words from a given series that contain at least two vowels.** |

**SOURCE CODE:**

import pandas as pd

# Sample Series

data = pd.Series(['apple', 'banana', 'grape', 'kiwi', 'pear', 'mango'])

# Function to count vowels in a word

def count\_vowels(word):

vowels = set('aeiouAEIOU')

return sum(1 for char in word if char in vowels)

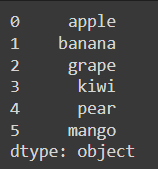
# Filter words with at least two vowels

filtered\_words = data[data.apply(lambda word: count\_vowels(word) >= 2)]

# Display the result

print(filtered\_words)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.9 Date :** |
| **Write a Pandas program to find the index of the first occurrence of the smallest and largest values of a given series.** |

**SOURCE CODE:**

import pandas as pd

data=pd.Series([2,67,100,0,3,7,8])

min\_index=data.idxmin()

max\_index=data.idxmax()

print("index of minimum value",min\_index)

print("index of max value",max\_index)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.10 Date :** |
| **Write a Pandas program to get the first 3 rows of a given data frame.** |

**SOURCE CODE:**

import pandas as pd

# Sample data for the DataFrame

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],

'Age': [25, 30, 35, 40, 28],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix']

}

# Creating a DataFrame

df = pd.DataFrame(data)

# Displaying the first 3 rows

first\_three\_rows = df.head(3)

print(first\_three\_rows)

**OUTPUT:**

Name Age City

0 Alice 25 New York

1 Bob 30 Los Angeles

2 Charlie 35 Chicago

|  |
| --- |
| **Program #4.2.11 Date :** |
| **Write a Pandas program to select the 'name' and 'score' columns from a data frame.** |

**SOURCE CODE:**

import pandas as pd

data={'name':['varghese','kuku','aami'],

'rollno':[12,45,19],

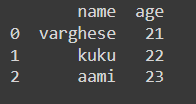
'age':[21,22,23]}

df=pd.DataFrame(data)

selected\_columns=df[['name','age']]

print(selected\_columns)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.12 Date :** |
| **Write a Pandas program to count the number of rows and columns in a data frame.** |

**SOURCE CODE:**

import pandas as pd

data={'name':['varghese','kuku','aami'],

'rollno':[34,78,12],

'age':[14,34,23]}

df=pd.DataFrame(data)

num\_rows=df.shape[0]

num\_cols=df.shape[1]

print("no of rows:",num\_rows)

print("no of columns",num\_cols)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.13 Date :** |
| **Write a Pandas program to add one row to an existing data frame.** |

**SOURCE CODE:**

import pandas as pd

data={'name':['varghese','kuku','aami'],

'rollno':[23,12,45],

'age':[22,21,20]}

df=pd.DataFrame(data)

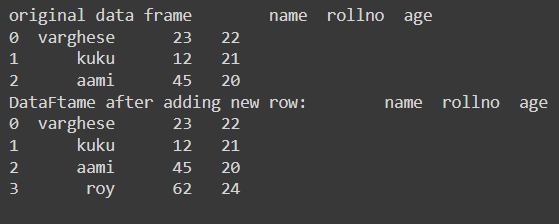
print("original data frame",df)

new\_row={'name':'roy','rollno':62,'age':24}

df = pd.concat([df, pd.DataFrame([new\_row])], ignore\_index=True)

print("DataFtame after adding new row:",df)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.14 Date :** |
| **Write a Pandas program to write a data frame to a CSV file using a tab separator.** |

**SOURCE CODE:**

import pandas as pd

# Creating a sample DataFrame

data = {'name': ['varghese', 'kuku', 'aami'],

'rollno': [23, 12, 45],

'age': [22, 21, 20]}

df = pd.DataFrame(data)

# Writing the DataFrame to a CSV file using a tab separator

df.to\_csv('output\_file.csv', sep='\t', index=False)

print("DataFrame has been written to 'output\_file.csv' using a tab separator.")

**OUTPUT:**



|  |
| --- |
| **Program #4.2.15 Date :** |
| **Write a Pandas program to replace all the NaN values with Zeros in a column of a data frame. Write a Pandas program to drop a list of rows from a specified data frame.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

# Creating a sample DataFrame with NaN values

data = {'name': ['John', 'Emily', 'Michael', 'Sara', 'David'],

'score': [85, np.nan, 78, np.nan, 95],

'age': [20, 21, 19, 22, 20]}

df = pd.DataFrame(data)

# Display original DataFrame

print("Original DataFrame with NaN values:")

print(df)

# Replacing NaN values with zeros in the 'score' column without using inplace

df['score'] = df['score'].fillna(0)

# Display DataFrame after replacing NaN values

print("\nDataFrame after replacing NaN values with zeros in 'score' column:")

print(df)

# Dropping rows with index 1 (Emily) and 3 (Sara)

rows\_to\_drop = [1, 3]

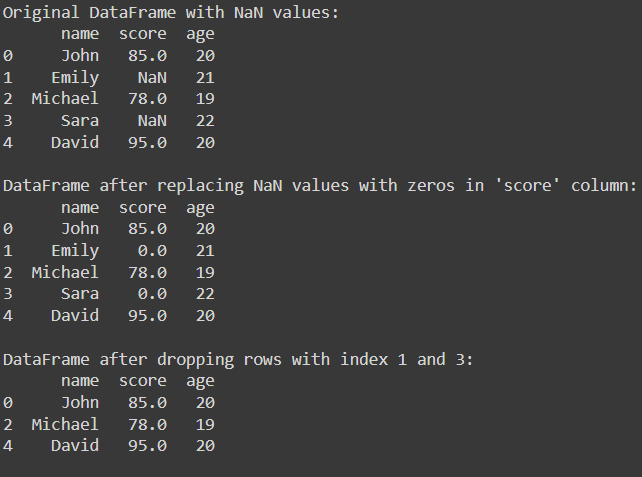
df = df.drop(rows\_to\_drop)

# Display DataFrame after dropping the specified rows

print("\nDataFrame after dropping rows with index 1 and 3:")

print(df)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.16 Date :** |
| **Write a Pandas program to shuffle a given data frame row.** |

**SOURCE CODE:**

import pandas as pd

# Creating a sample DataFrame

data = {

'name': ['John', 'Emily', 'Michael', 'Sara', 'David'],

'score': [85, 92, 78, 88, 95],

'age': [20, 21, 19, 22, 20]

}

df = pd.DataFrame(data)

# Display original DataFrame

print("Original DataFrame:")

print(df)

# Shuffling the DataFrame rows

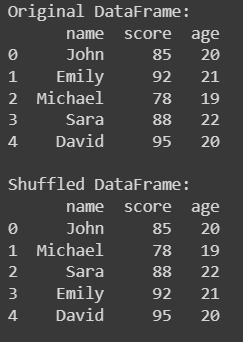
shuffled\_df = df.sample(frac=1).reset\_index(drop=True)

# Display shuffled DataFrame

print("\nShuffled DataFrame:")

print(shuffled\_df)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.17 Date :** |
| **Write a Pandas program to find the row where the value of a given column is maximum.** |

**SOURCE CODE:**

import pandas as pd

# Creating a sample DataFrame

data = {

'name': ['John', 'Emily', 'Michael', 'Sara', 'David'],

'score': [85, 92, 78, 88, 95],

'age': [20, 21, 19, 22, 20]

}

df = pd.DataFrame(data)

# Display original DataFrame

print("Original DataFrame:")

print(df)

# Finding the row where the value of the 'score' column is maximum

max\_score\_index = df['score'].idxmax()

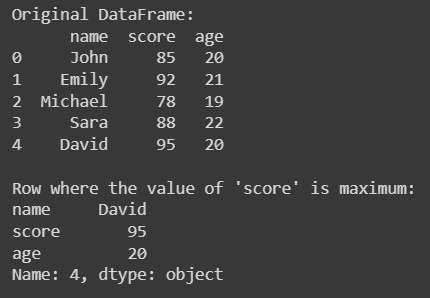
max\_score\_row = df.loc[max\_score\_index]

# Display the row with the maximum score

print("\nRow where the value of 'score' is maximum:")

print(max\_score\_row)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.18 Date :** |
| **Write a Pandas program to check whether a given column is present in a data frame or not.** |

**SOURCE CODE:**

import pandas as pd

# Creating a sample DataFrame

data = {

'name': ['John', 'Emily', 'Michael', 'Sara', 'David'],

'score': [85, 92, 78, 88, 95],

'age': [20, 21, 19, 22, 20]

}

df = pd.DataFrame(data)

# Display original DataFrame

print("Original DataFrame:")

print(df)

# Column to check

column\_to\_check = 'score'

# Checking if the column is present in the DataFrame

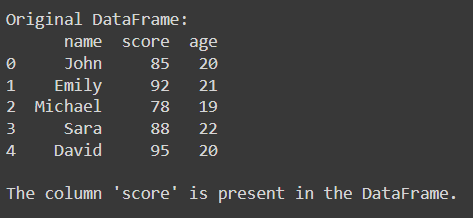
if column\_to\_check in df.columns:

print(f"\nThe column '{column\_to\_check}' is present in the DataFrame.")

else:

print(f"\nThe column '{column\_to\_check}' is not present in the DataFrame.")

**OUTPUT:**



|  |
| --- |
| **Program #4.2.19 Date :** |
| **Write a Pandas program to append data to an empty data frame.** |

**SOURCE CODE:**

import pandas as pd

# Creating an empty DataFrame

df = pd.DataFrame(columns=['name', 'score', 'age'])

# Display the empty DataFrame

print("Empty DataFrame:")

print(df)

# Data to append

data\_to\_append = [

{'name': 'John', 'score': 85, 'age': 20},

{'name': 'Emily', 'score': 92, 'age': 21},

{'name': 'Michael', 'score': 78, 'age': 19},

{'name': 'Sara', 'score': 88, 'age': 22},

{'name': 'David', 'score': 95, 'age': 20}

]

# Creating a DataFrame from the data to append

data\_df = pd.DataFrame(data\_to\_append)

# Appending data to the DataFrame using pd.concat

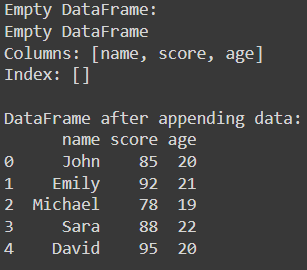
df = pd.concat([df, data\_df], ignore\_index=True)

# Display the DataFrame after appending data

print("\nDataFrame after appending data:")

print(df)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.20 Date :** |
| **Write a Pandas program to convert continuous values of a column in a given data frame to categorical. Input: { 'Name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Syed Wharton'], 'Age': [18, 22, 40, 50, 80, 5] }** |

**SOURCE CODE:**

import pandas as pd

# Creating a sample DataFrame

data = {

'Name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes', 'Eesha Hinton', 'Syed Wharton'],

'Age': [18, 22, 40, 50, 80]

}

df = pd.DataFrame(data)

# Display the original DataFrame

print("Original DataFrame:")

print(df)

# Define bins and labels for the age categories

bins = [0, 18, 35, 50, 80] # Define the edges of the bins

labels = ['Child', 'Young Adult', 'Adult', 'Senior'] # Define the labels for each bin

# Convert the 'Age' column to categorical using pd.cut

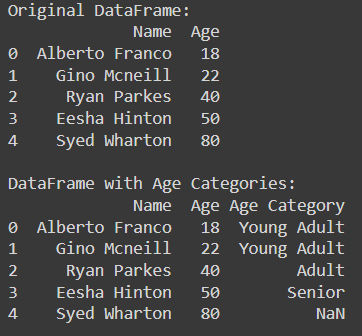
df['Age Category'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)

# Display the DataFrame with the new categorical column

print("\nDataFrame with Age Categories:")

print(df)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.21 Date :** |
| **Write a Pandas program to create data frames that contain random values, missing values, datetime values, and mixed values.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

import random

from datetime import datetime, timedelta

# Set a seed for reproducibility

np.random.seed(42)

# Create a DataFrame with random values

random\_values\_df = pd.DataFrame({

'Random Numbers': np.random.randint(1, 100, size=10),

'Floats': np.random.rand(10),

})

# Create a DataFrame with missing values

missing\_values\_df = pd.DataFrame({

'A': [1, 2, np.nan, 4, 5],

'B': [np.nan, 'two', 'three', np.nan, 'five'],

})

# Create a DataFrame with datetime values

dates = [datetime.now() + timedelta(days=i) for i in range(10)]

datetime\_values\_df = pd.DataFrame({

'Date': dates,

'Value': np.random.rand(10)

})

# Create a DataFrame with mixed values

mixed\_values\_df = pd.DataFrame({

'Integers': [1, 2, 3, 4, 5],

'Strings': ['one', 'two', 'three', 'four', 'five'],

'Booleans': [True, False, True, False, True],

})

# Display the DataFrames

print("DataFrame with Random Values:")

print(random\_values\_df)

print("\nDataFrame with Missing Values:")

print(missing\_values\_df)

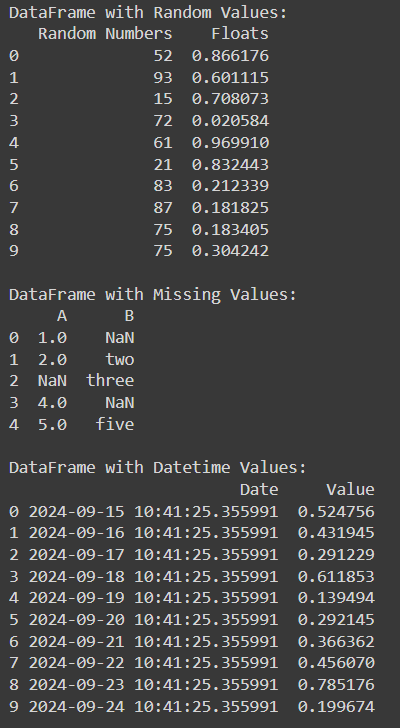
print("\nDataFrame with Datetime Values:")

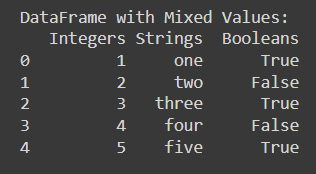
print(datetime\_values\_df)

print("\nDataFrame with Mixed Values:")

print(mixed\_values\_df)

**OUTPUT:**





|  |
| --- |
| **Program #4.2.22 Date :** |
| **Write a Pandas program to join the two given data frames along rows and assign all the data.**  ***student\_data1:***   |  |  |  |  | | --- | --- | --- | --- | |  | **student\_id** | **name** | **marks** | | 0 | s1 | Danniella Fenton | 200 | | 1 | s2 | Ryder Storey | 210 | | 2 | s3 | Bryce Jensen | 190 | | 3 | s4 | Ed Bernal | 222 | | 4 | s5 | Kwame Morin | 199 |   ***student\_data2:***   |  |  |  |  | | --- | --- | --- | --- | |  | **student\_id** | **name** | **marks** | | 0 | s4 | Scarlette Fisher | 201 | | 1 | s5 | Carla Williamson | 200 | | 2 | s6 | Dante Morse | 198 | | 3 | s7 | Kaiser William | 219 | | 4 | s8 | Madeeha Preston | 201 | |

**SOURCE CODE:**

import pandas as pd

# Creating the first DataFrame

student\_data1 = pd.DataFrame({

'student\_id': ['s1', 's2', 's3', 's4', 's5'],

'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],

'marks': [200, 210, 190, 222, 199]

})

# Creating the second DataFrame

student\_data2 = pd.DataFrame({

'student\_id': ['s4', 's5', 's6', 's7', 's8'],

'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston'],

'marks': [201, 200, 198, 219, 201]

})

# Displaying the original DataFrames

print("Student Data 1:")

print(student\_data1)

print("\nStudent Data 2:")

print(student\_data2)

# Joining the two DataFrames along rows

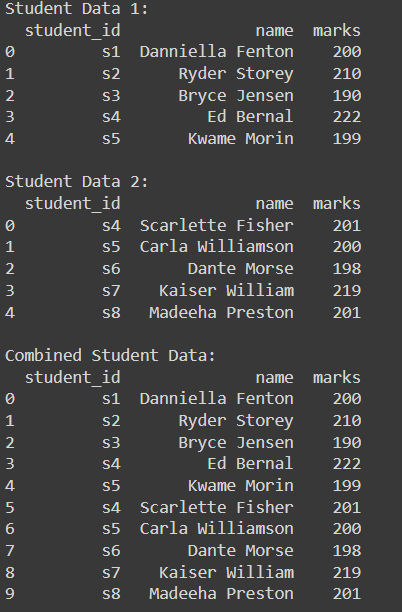
combined\_data = pd.concat([student\_data1, student\_data2], ignore\_index=True)

# Displaying the combined DataFrame

print("\nCombined Student Data:")

print(combined\_data)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.23 Date :** |
| **Write a Pandas program to join the two given data frames along columns and assign all the data. (Use the same dataset as above.)** |

**SOURCE CODE:**

import pandas as pd

# Creating the first DataFrame

student\_data1 = pd.DataFrame({

'student\_id': ['s1', 's2', 's3', 's4', 's5'],

'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],

'marks': [200, 210, 190, 222, 199]

})

# Creating the second DataFrame

student\_data2 = pd.DataFrame({

'student\_id': ['s4', 's5', 's6', 's7', 's8'],

'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston'],

'marks': [201, 200, 198, 219, 201]

})

# Displaying the original DataFrames

print("Student Data 1:")

print(student\_data1)

print("\nStudent Data 2:")

print(student\_data2)

# Joining the two DataFrames along columns

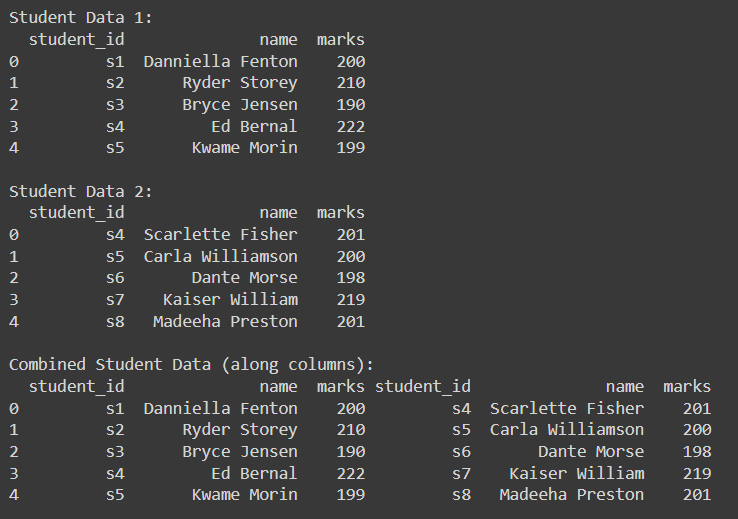
combined\_data\_columns = pd.concat([student\_data1, student\_data2], axis=1)

# Displaying the combined DataFrame along columns

print("\nCombined Student Data (along columns):")

print(combined\_data\_columns)

**OUTPUT:**



|  |
| --- |
| **Program #4.2.24 Date :** |
| **Write a Pandas program to join the two given data frames along rows and merge with another data frame along the common column id.**  ***exam\_data:***  **student\_id  exam\_id**  0          S1       23  1          S2       45  2          S3       12  3          S4       67  4          S5       21  5          S7      55  6          S8       33  7          S9       14  8         S10       56  9         S11       83  10       S12       88  11       S13       12  (Add this data to the above dataset.) |

**SOURCE CODE:**

import pandas as pd

# Creating the first DataFrame (student data)

student\_data = pd.DataFrame({

'student\_id': ['s1', 's2', 's3', 's4', 's5'],

'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],

'marks': [200, 210, 190, 222, 199]

})

# Creating the second DataFrame (more student data)

additional\_student\_data = pd.DataFrame({

'student\_id': ['s4', 's5', 's6', 's7', 's8'],

'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston'],

'marks': [201, 200, 198, 219, 201]

})

# Joining the two DataFrames along rows

combined\_student\_data = pd.concat([student\_data, additional\_student\_data], ignore\_index=True)

# Creating the exam data DataFrame

exam\_data = pd.DataFrame({

'student\_id': ['S1', 'S2', 'S3', 'S4', 'S5', 'S7', 'S8', 'S9', 'S10', 'S11', 'S12', 'S13'],

'exam\_id': [23, 45, 12, 67, 21, 55, 33, 14, 56, 83, 88, 12]

})

# Displaying the combined student data

print("Combined Student Data:")

print(combined\_student\_data)

# Merging the combined student data with exam data along the common column 'student\_id'

# First, we need to ensure student\_id in exam\_data matches the format in combined\_student\_data

exam\_data['student\_id'] = exam\_data['student\_id'].str.lower() # Convert to lowercase

# Merging the DataFrames

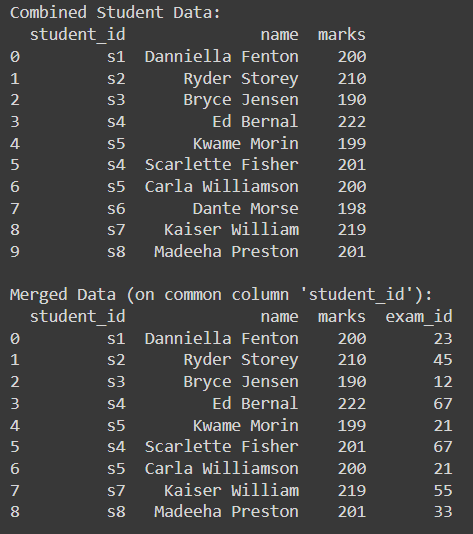
merged\_data = pd.merge(combined\_student\_data, exam\_data, on='student\_id', how='inner')

# Displaying the merged data

print("\nMerged Data (on common column 'student\_id'):")

print(merged\_data)

**OUTPUT**



|  |
| --- |
| **Program #4.2.25 Date :** |
| **Write a Pandas program to join the two data frames with matching records from both sides, where available. (Same dataset as above.)** |

**SOURCE CODE:**

import pandas as pd

# Creating the first DataFrame (student data)

student\_data = pd.DataFrame({

'student\_id': ['s1', 's2', 's3', 's4', 's5'],

'name': ['Danniella Fenton', 'Ryder Storey', 'Bryce Jensen', 'Ed Bernal', 'Kwame Morin'],

'marks': [200, 210, 190, 222, 199]

})

# Creating the second DataFrame (additional student data)

additional\_student\_data = pd.DataFrame({

'student\_id': ['s4', 's5', 's6', 's7', 's8'],

'name': ['Scarlette Fisher', 'Carla Williamson', 'Dante Morse', 'Kaiser William', 'Madeeha Preston'],

'marks': [201, 200, 198, 219, 201]

})

# Joining the two DataFrames along rows

combined\_student\_data = pd.concat([student\_data, additional\_student\_data], ignore\_index=True)

# Creating the exam data DataFrame

exam\_data = pd.DataFrame({

'student\_id': ['s1', 's2', 's3', 's4', 's5', 's7', 's8', 's9', 's10', 's11', 's12', 's13'],

'exam\_id': [23, 45, 12, 67, 21, 55, 33, 14, 56, 83, 88, 12]

})

# Displaying the combined student data

print("Combined Student Data:")

print(combined\_student\_data)

# Merging the combined student data with exam data on common column 'student\_id'

# Use an inner join to keep matching records from both sides

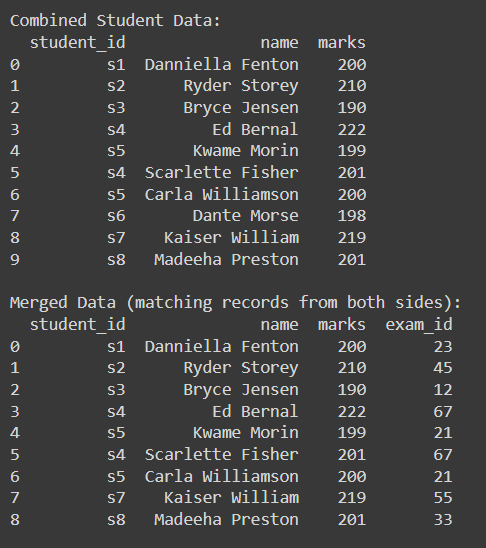
merged\_data = pd.merge(combined\_student\_data, exam\_data, on='student\_id', how='inner')

# Displaying the merged data with matching records

print("\nMerged Data (matching records from both sides):")

print(merged\_data)

**OUTPUT:**



|  |
| --- |
| **Program #4.3.1 Date :** |
| **Write a Pandas program to split the following data frame into groups based on school code. Also, check the type of GroupBy object.**   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **school** | **class** | **name** | **date\_Of\_Birth** | **age** | **height** | **weight** | **address** | | S1 | s001 | V | Alberto Franco | 15/05/2002 | 12 | 173 | 35 | street1 | | S2 | s002 | V | Gino Mcneill | 17/05/2002 | 12 | 192 | 32 | street2 | | S3 | s003 | VI | Ryan Parkes | 16/02/1999 | 13 | 186 | 33 | street3 | | S4 | s001 | VI | Eesha Hinton | 25/09/1998 | 13 | 167 | 30 | street1 | | S5 | s002 | V | Gino Mcneill | 11/05/2002 | 14 | 151 | 31 | street2 | | S6 | s004 | VI | David Parkes | 15/09/1997 | 12 | 159 | 32 | street4 | |

**SOURCE CODE:**

import pandas as pd

# Create the DataFrame

data = {

    'school': ['s001', 's002', 's003', 's001', 's002', 's004'],

    'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],

    'name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes',

             'Eesha Hinton', 'Gino Mcneill', 'David Parkes'],

    'date\_Of\_Birth': ['15/05/2002', '17/05/2002', '16/02/1999',

                      '25/09/1998', '11/05/2002', '15/09/1997'],

    'age': [12, 12, 13, 13, 14, 12],

    'height': [173, 192, 186, 167, 151, 159],

    'weight': [35, 32, 33, 30, 31, 32],

    'address': ['street1', 'street2', 'street3', 'street1',

                'street2', 'street4']

}

df = pd.DataFrame(data)

# Group by school code

grouped = df.groupby('school')

# Display the grouped data

for school\_code, group in grouped:

    print(f"Group for school code {school\_code}:")

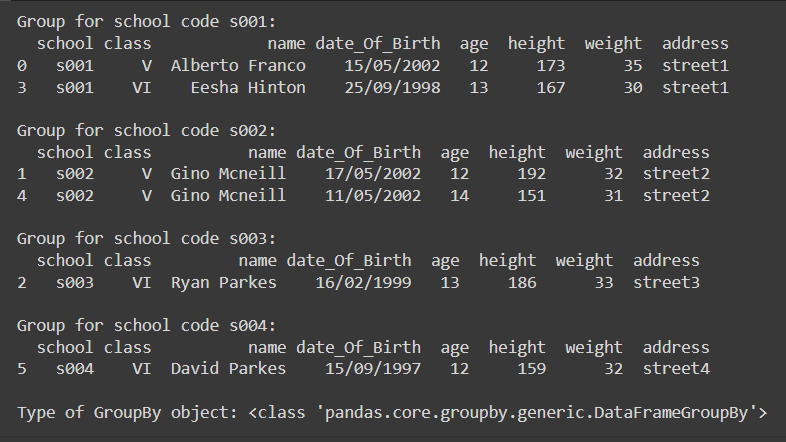
    print(group)

    print()

# Check the type of GroupBy object

print(f"Type of GroupBy object: {type(grouped)}")

**OUTPUT:**



|  |
| --- |
| **Program #4.3.2 Date :** |
| **Write a Pandas program to split the following data frame by school code and get the mean, min, and max values of age for each school. (Use the above dataset.)** |

**SOURCE CODE:**

import pandas as pd

# Create the DataFrame

data = {

    'school': ['s001', 's002', 's003', 's001', 's002', 's004'],

    'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],

    'name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes',

             'Eesha Hinton', 'Gino Mcneill', 'David Parkes'],

    'date\_Of\_Birth': ['15/05/2002', '17/05/2002', '16/02/1999',

                      '25/09/1998', '11/05/2002', '15/09/1997'],

    'age': [12, 12, 13, 13, 14, 12],

    'height': [173, 192, 186, 167, 151, 159],

    'weight': [35, 32, 33, 30, 31, 32],

    'address': ['street1', 'street2', 'street3', 'street1',

                'street2', 'street4']

}

df = pd.DataFrame(data)

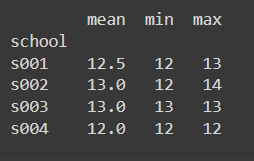
# Group by school code and calculate mean, min, and max for age

age\_stats = df.groupby('school')['age'].agg(['mean', 'min', 'max']

# Display the result

print(age\_stats)

**OUTPUT:**

****

|  |
| --- |
| **Program #4.3.3 Date :** |
| **Write a Pandas program to split the following data frame into groups based on all columns and calculate groupby value counts on the data frame.**  ***Test Data:***  ***Id type book***  *0 1    10     Math*  *1 2    15  English*  *2   1 11 Physics*  *3   1    20     Math*  *4   2    21  English*  *5   1    12  Physics*  *6   2    14  English* |

**SOURCE CODE:**

import pandas as pd

# Create the DataFrame

data = {

    'Id': [1, 2, 1, 1, 2, 1, 2],

    'type': [10, 15, 11, 20, 21, 12, 14],

    'book': ['Math', 'English', 'Physics', 'Math', 'English', 'Physics', 'English']

}

df = pd.DataFrame(data)

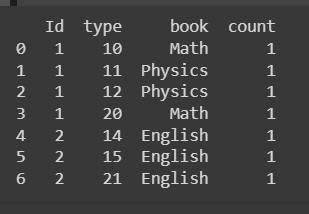
# Group by all columns and calculate value counts

grouped\_counts = df.groupby(['Id', 'type', 'book']).size().reset\_index(name='count')

# Display the result

print(grouped\_counts)

**OUTPUT:**

****

|  |
| --- |
| **Program #4.3.4 Date :** |
| **Write a Pandas program to split the following data frame into groups by school code and get the mean, min, and max values of age with customized column names for each school.** |

**SOURCE CODE:**

import pandas as pd

# Create the DataFrame

data = {

    'school': ['s001', 's002', 's003', 's001', 's002', 's004'],

    'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],

    'name': ['Alberto Franco', 'Gino Mcneill', 'Ryan Parkes',

             'Eesha Hinton', 'Gino Mcneill', 'David Parkes'],

    'date\_Of\_Birth': ['15/05/2002', '17/05/2002', '16/02/1999',

                      '25/09/1998', '11/05/2002', '15/09/1997'],

    'age': [12, 12, 13, 13, 14, 12],

    'height': [173, 192, 186, 167, 151, 159],

    'weight': [35, 32, 33, 30, 31, 32],

    'address': ['street1', 'street2', 'street3', 'street1',

                'street2', 'street4']

}

df = pd.DataFrame(data)

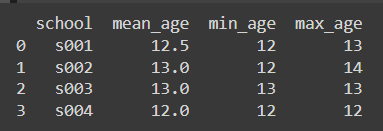
# Group by school code and calculate mean, min, and max for age

age\_stats = df.groupby('school')['age'].agg(mean\_age='mean', min\_age='min', max\_age='max').reset\_index()

# Display the result

print(age\_stats)

**OUTPUT:**

****

|  |
| --- |
| **Program #4.4.1 Date :** |
| **Visualize the following using the given dataset (alphabet\_stock\_data.csv),**  **4.4.1.1 Create a line plot of the historical stock prices of Alphabet Inc. between two specific dates.**  **4.4.1.2 Create a bar plot of the trading volume of Alphabet Inc. stock between two specific dates.**  **4.4.1.3 Create a stacked histogram plot with more bins of opening, closing, high, and low stock prices of Alphabet Inc. between two specific dates.**  **4.4.1.4 Create a scatter plot of the trading volume/stock prices of Alphabet Inc. stock between two specific dates.** |

**SOURCE CODE 1.1:**

# Step 1: Install libraries (uncomment if needed)

# !pip install pandas matplotlib

# Step 2: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

# Step 3: Load the dataset

df = pd.read\_csv('alphabet\_stock\_data.csv')

# Step 4: Convert the date column to datetime

df['Date'] = pd.to\_datetime(df['Date'])

# Step 5: Set the start and end dates for filtering

start\_date = '2023-01-01'  # Replace with your start date

end\_date = '2023-12-31'    # Replace with your end date

# Filter the DataFrame for the date range

filtered\_df = df[(df['Date'] >= start\_date) & (df['Date'] <= end\_date)]

# Step 6: Plot the historical stock prices

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

plt.plot(filtered\_df['Date'], filtered\_df['Close'], label='Closing Price', color='blue')

plt.title('Historical Stock Prices of Alphabet Inc.')

plt.xlabel('Date')

plt.ylabel('Stock Price (USD)')

plt.xticks(rotation=45)

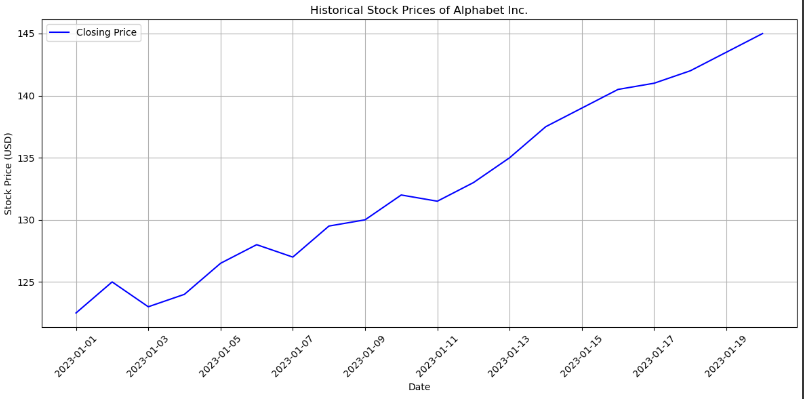
plt.legend()

plt.grid()

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**SOURCE CODE 1.2:**

# Step 1: Install libraries (uncomment if needed)

# !pip install pandas matplotlib

# Step 2: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

# Step 3: Load the dataset

df = pd.read\_csv('alphabet\_stock\_data.csv')

# Step 4: Convert the date column to datetime

df['Date'] = pd.to\_datetime(df['Date'])

# Step 5: Set the start and end dates for filtering

start\_date = '2023-01-01'  # Replace with your start date

end\_date = '2023-01-20'    # Replace with your end date

# Filter the DataFrame for the date range

filtered\_df = df[(df['Date'] >= start\_date) & (df['Date'] <= end\_date)]

# Step 6: Plot the trading volume

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

plt.bar(filtered\_df['Date'], filtered\_df['Volume'], color='orange')

plt.title('Trading Volume of Alphabet Inc. Stock')

plt.xlabel('Date')

plt.ylabel('Trading Volume')

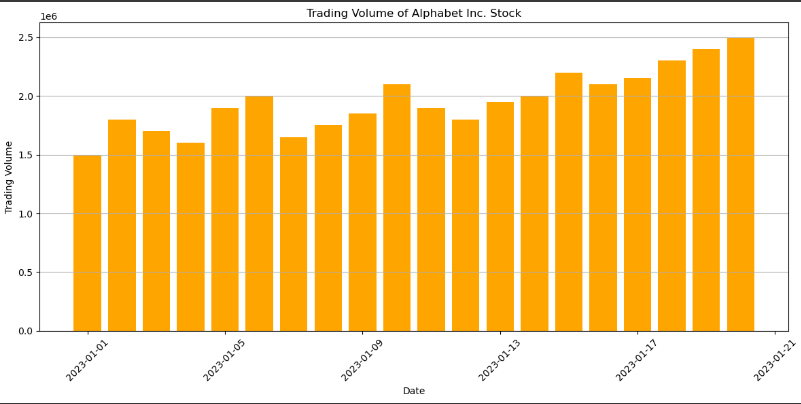
plt.xticks(rotation=45)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

**OUTPUT:**



**SOURCE CODE 1.3:**

# Step 1: Install libraries (uncomment if needed)

# !pip install pandas matplotlib

# Step 2: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

# Step 3: Load the dataset

df = pd.read\_csv('alphabet\_stock\_data.csv')

# Step 4: Convert the date column to datetime

df['Date'] = pd.to\_datetime(df['Date'])

# Step 5: Set the start and end dates for filtering

start\_date = '2023-01-01'  # Replace with your start date

end\_date = '2023-01-20'    # Replace with your end date

# Filter the DataFrame for the date range

filtered\_df = df[(df['Date'] >= start\_date) & (df['Date'] <= end\_date)]

# Step 6: Create a stacked histogram

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

plt.hist(filtered\_df['Open'], bins=30, alpha=0.5, label='Opening Price', color='blue')

plt.hist(filtered\_df['Close'], bins=30, alpha=0.5, label='Closing Price', color='green')

plt.hist(filtered\_df['High'], bins=30, alpha=0.5, label='High Price', color='orange')

plt.hist(filtered\_df['Low'], bins=30, alpha=0.5, label='Low Price', color='red')

plt.title('Stacked Histogram of Alphabet Inc. Stock Prices')

plt.xlabel('Stock Price (USD)')

plt.ylabel('Frequency')

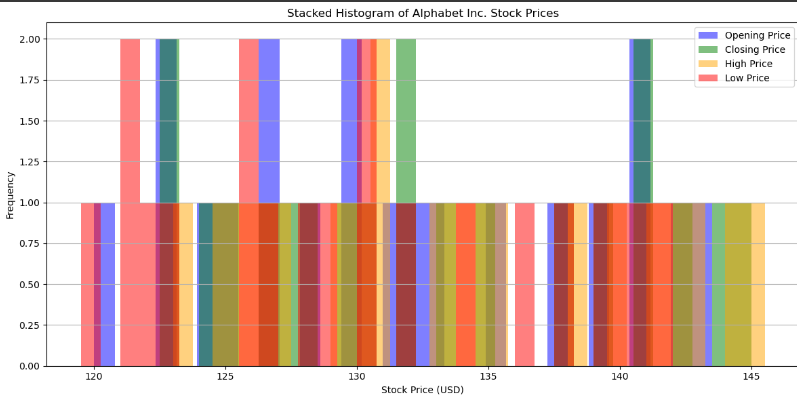
plt.legend(loc='upper right')

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

**SOURCE CODE 1.4:**

# Step 1: Install libraries (uncomment if needed)

# !pip install pandas matplotlib

# Step 2: Import libraries

import pandas as pd

import matplotlib.pyplot as plt

# Step 3: Load the dataset

df = pd.read\_csv('alphabet\_stock\_data.csv')

# Step 4: Convert the date column to datetime

df['Date'] = pd.to\_datetime(df['Date'])

# Step 5: Set the start and end dates for filtering

start\_date = '2023-01-01'  # Replace with your start date

end\_date = '2023-01-20'    # Replace with your end date

# Filter the DataFrame for the date range

filtered\_df = df[(df['Date'] >= start\_date) & (df['Date'] <= end\_date)]

# Step 6: Create a scatter plot

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

plt.scatter(filtered\_df['Close'], filtered\_df['Volume'], color='purple', alpha=0.5)

plt.title('Scatter Plot of Trading Volume vs. Closing Prices of Alphabet Inc.')

plt.xlabel('Closing Price (USD)')

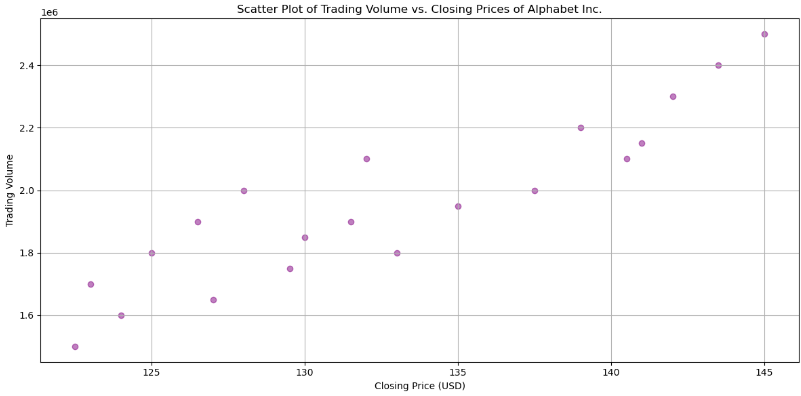
plt.ylabel('Trading Volume')

plt.grid()

plt.tight\_layout()

plt.show()

**OUTPUT**



|  |
| --- |
| **Program #4.4.2 Date :** |
| **Write a Python program to draw a line with a suitable label on the x-axis, y-axis, and title.** |

**SOURCE CODE:**

import matplotlib.pyplot as plt

# Sample data for the line plot

x = [1, 2, 3, 4, 5]  # X-axis data

y = [2, 3, 5, 7, 11]  # Y-axis data

# Create the line plot

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

plt.plot(x, y, marker='o', linestyle='-', color='b')  # You can customize marker and linestyle

# Adding labels and title

plt.title('Sample Line Plot')

plt.xlabel('X-axis Label')  # Replace with your x-axis label

plt.ylabel('Y-axis Label')  # Replace with your y-axis label

# Show grid

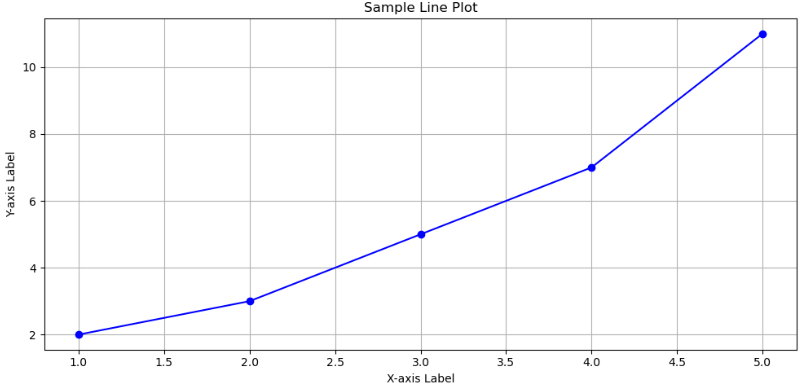
plt.grid()

# Show the plot

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

|  |
| --- |
| **Program #4.4.3 Date :** |
| **Write a Python program to draw line charts of the financial data of Alphabet Inc. between October 3, 2016, and October 7, 2016.**  **Date,Open,High,Low,Close**  **10-03-16,774.25,776.065002,769.5,772.559998**  **10-04-16,776.030029,778.710022,772.890015,776.429993**  **10-05-16,779.309998,782.070007,775.650024,776.469971**  **10-06-16,779,780.47998,775.539978,776.859985**  **10-07-16,779.659973,779.659973,770.75,775.080017** |

**SOURCE CODE:**

import pandas as pd

import matplotlib.pyplot as plt

# Sample financial data for Alphabet Inc.

data = {

    'Date': ['10-03-16', '10-04-16', '10-05-16', '10-06-16', '10-07-16'],

    'Open': [774.25, 776.03, 779.31, 779, 779.66],

    'High': [776.065002, 778.710022, 782.070007, 780.47998, 779.659973],

    'Low': [769.5, 772.890015, 775.650024, 775.539978, 770.75],

    'Close': [772.559998, 776.429993, 776.469971, 776.859985, 775.080017]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Convert the 'Date' column to datetime format

df['Date'] = pd.to\_datetime(df['Date'], format='%m-%d-%y')

# Set the 'Date' column as the index

df.set\_index('Date', inplace=True)

# Plotting the financial data

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

# Line charts for Open, High, Low, and Close prices

plt.plot(df.index, df['Open'], marker='o', label='Open', linestyle='-')

plt.plot(df.index, df['High'], marker='o', label='High', linestyle='-')

plt.plot(df.index, df['Low'], marker='o', label='Low', linestyle='-')

plt.plot(df.index, df['Close'], marker='o', label='Close', linestyle='-')

# Adding title and labels

plt.title('Financial Data of Alphabet Inc. (Oct 3, 2016 - Oct 7, 2016)')

plt.xlabel('Date')

plt.ylabel('Stock Price (USD)')

plt.xticks(rotation=45)

plt.grid()

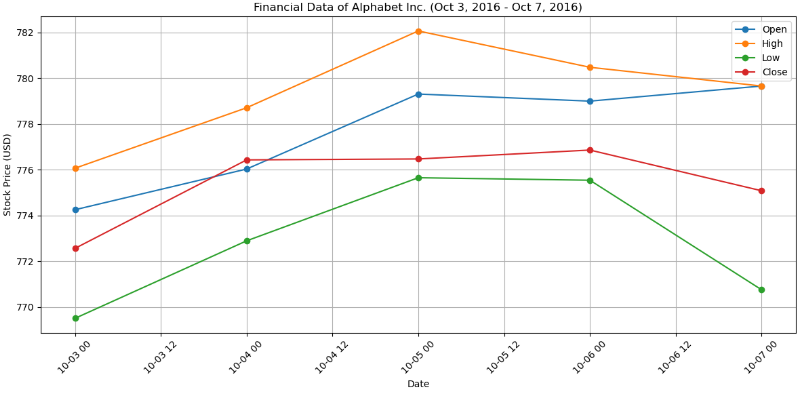
plt.legend()  # Show legend

plt.tight\_layout()

# Show the plot

plt.show()

**OUTPUT:**



|  |
| --- |
| **Program #4.4.4 Date :** |
| **Write a Python program to draw a line with a suitable label on the x-axis, and y-axis and a title. Create the code snippet that gives the output shown in the following screenshot:** |

**SOURCE CODE:**

import matplotlib.pyplot as plt

# Data for plotting

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

y1 = [10, 20, 30, 20, 10]  # Blue line data (touches lower boundary at 10)

y2 = [40, 30, 20, 30, 40]  # Red line data (touches upper boundary at 40)

# Create the plot with specific line styles

plt.plot(x, y1, 'b:', label='line1-dotted', linewidth=1)  # Blue dotted line, thinner

plt.plot(x, y2, 'r--', label='line2-dashed', linewidth=5)  # Red dashed line, thicker

# Add labels and title

plt.xlabel('x - axis')

plt.ylabel('y - axis')

plt.title('Plot with two or more lines with different styles')

# Set x and y-axis limits to ensure the lines touch the boundaries

plt.xlim(10, 30)

plt.ylim(10, 40)

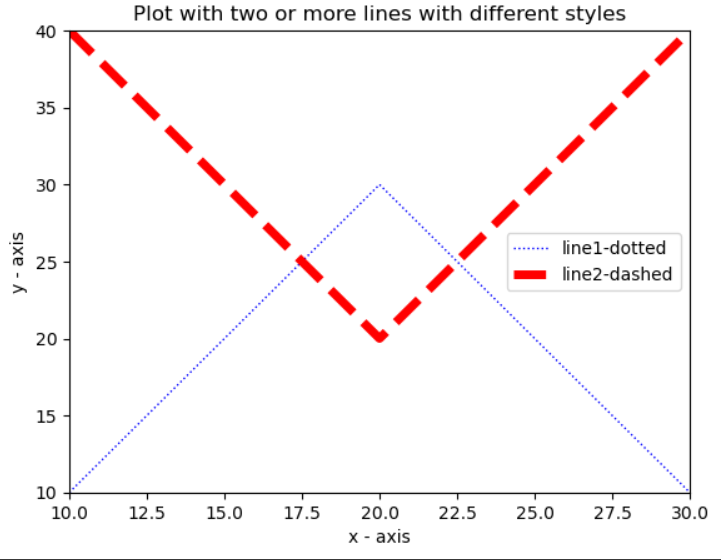
# Add a legend

plt.legend()

# Show the plot

plt.show()

**OUTPUT**



|  |
| --- |
| **Program #4.4.5 Date :** |
| **Write a Python program to display the grid and draw line charts of the closing value of Alphabet Inc. between October 3, 2016, and October 7, 2016. Customized the grid lines with linestyle -, width 0.5, and color blue.**  **Date,Close**  **03-10-16,772.559998**  **04-10-16,776.429993**  **05-10-16,776.469971**  **06-10-16,776.859985**  **07-10-16,775.080017** |

**SOURCE CODE:**

import matplotlib.pyplot as plt

import pandas as pd

# Data: Date and Closing value of Alphabet Inc.

dates = ['03-10-16', '04-10-16', '05-10-16', '06-10-16', '07-10-16']

close\_values = [772.559998, 776.429993, 776.469971, 776.859985, 775.080017]

# Convert date strings into a pandas datetime format

dates = pd.to\_datetime(dates, format='%d-%m-%y')

# Plotting the data

plt.plot(dates, close\_values, marker='o', linestyle='-', color='green')

# Adding title and labels

plt.title('Alphabet Inc. Closing Value (Oct 3, 2016 - Oct 7, 2016)')

plt.xlabel('Date')

plt.ylabel('Closing Value')

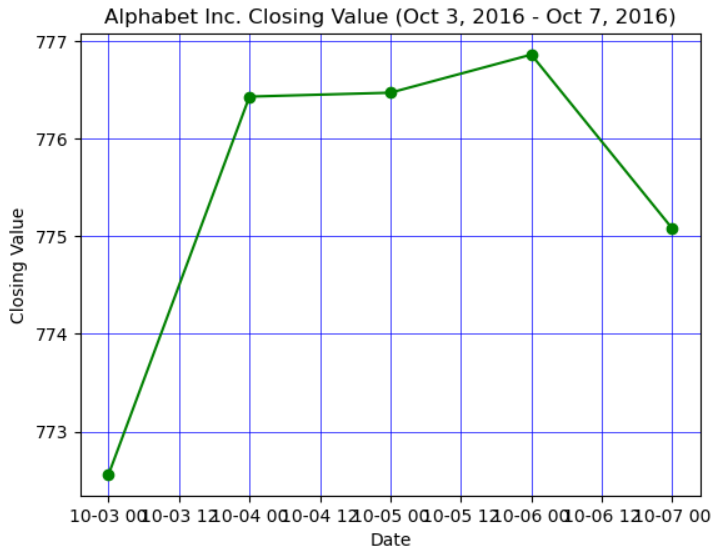
# Customizing the grid with linestyle, width, and color

plt.grid(True, linestyle='-', linewidth=0.5, color='blue')

# Display the plot

plt.show()

**OUTPUT**

****

|  |
| --- |
| **Program #4.4.6 Date :** |
| **Write a Python program to create multiple plots as in the screenshot (use any method).** |

**SOURCE CODE:**

import matplotlib.pyplot as plt

# Create figure

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

# Add the large plot (1 row, 1 column)

ax1 = plt.subplot2grid((2, 3), (0, 0), colspan=3)

# Add the three smaller plots below (3 columns)

ax2 = plt.subplot2grid((2, 3), (1, 0))

ax3 = plt.subplot2grid((2, 3), (1, 1))

ax4 = plt.subplot2grid((2, 3), (1, 2))

# Plot dummy data (optional)

ax1.plot([0, 1], [0, 1])

ax1.set\_title('Large Plot')

ax2.plot([0, 1], [0, 1])

ax2.set\_title('Small Plot 1')

ax3.plot([0, 1], [0, 1])

ax3.set\_title('Small Plot 2')

ax4.plot([0, 1], [0, 1])

ax4.set\_title('Small Plot 3')

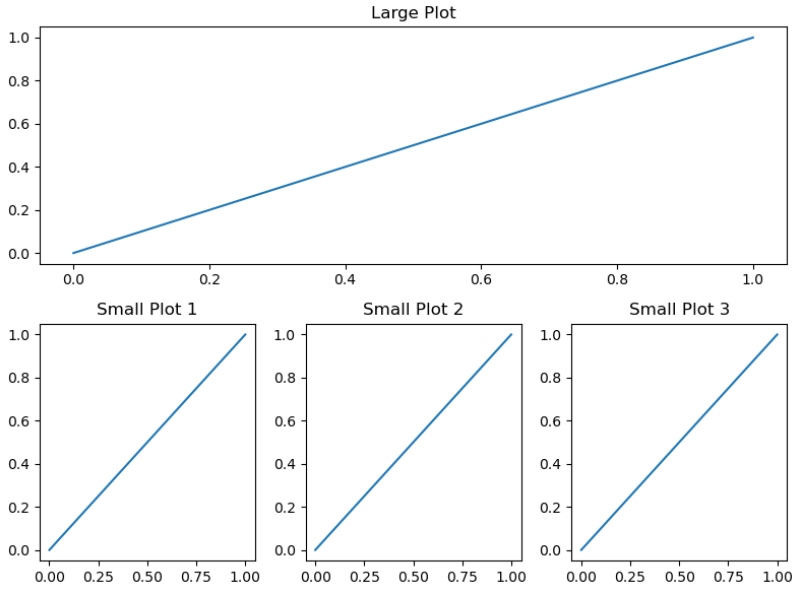
# Adjust layout

plt.tight\_layout()

# Show plot

plt.show()

**OUTPUT:**

****

|  |
| --- |
| **Program #4.4.7 Date :** |
| **Write a Python program to create a bar plot from a data frame.**  **Sample Data Frame: s**  **a b c d e f**  **2 4,8,5,7,6**  **4 2,3,4,2,6**  **6 4,7,4,7,8**  **8 2,6,4,8,6**  **10 2,4,3,3,2**  **Create the code snippet which gives the output shown in the following screenshot:** |

**SOURCE CODE:**

import pandas as pd

import matplotlib.pyplot as plt

# Sample data

data = {

    'a': [4, 2, 4, 2, 2],

    'b': [8, 3, 7, 6, 4],

    'c': [5, 4, 4, 4, 3],

    'd': [7, 2, 7, 8, 3],

    'e': [6, 6, 8, 6, 2]

}

# Create DataFrame

index = [2, 4, 6, 8, 10]

df = pd.DataFrame(data, index=index)

# Plotting the bar plot

df.plot(kind='bar', width=0.8)

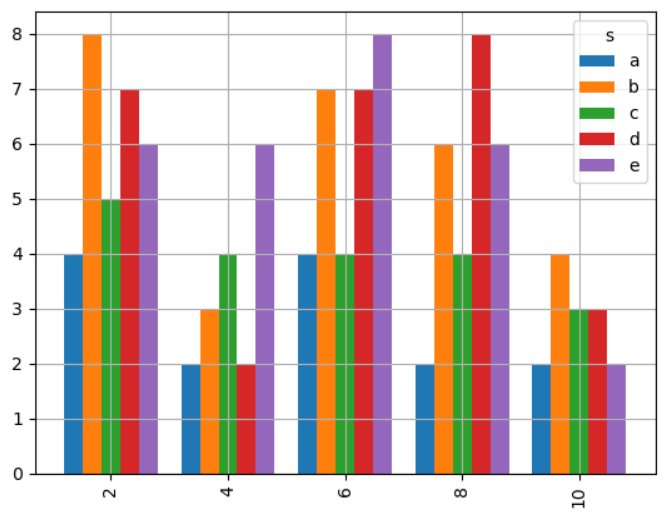
# Adding legend, labels, and grid

plt.legend(title="s", loc='best')

plt.grid(True)

plt.show()

**OUTPUT**



|  |
| --- |
| **Program #4.4.8 Date :** |
| **Write a Python program to create a stacked bar plot with error bars. Note: Use the bottom to stack the women's bars on top of the men's bars.**  **Sample Data:**  **Means (men) = (22, 30, 35, 35, 26)**  **Means (women) = (25, 32, 30, 35, 29)**  **Men's Standard deviation = (4, 3, 4, 1, 5)**  **Women's Standard deviation = (3, 5, 2, 3, 3)**  **Create the code snippet that gives the output shown in the following screenshot:** |

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

# Data

means\_men = (22, 30, 35, 35, 26)

means\_women = (25, 32, 30, 35, 29)

std\_men = (4, 3, 4, 1, 5)

std\_women = (3, 5, 2, 3, 3)

# Group labels

groups = ('Group1', 'Group2', 'Group3', 'Group4', 'Group5')

# X locations for the groups

x = np.arange(len(groups))

# Plotting the men's bars

plt.bar(x, means\_men, yerr=std\_men, color='r', width=0.6, label='Men')

# Plotting the women's bars on top of the men's bars (stacked)

plt.bar(x, means\_women, yerr=std\_women, color='g', width=0.6, bottom=means\_men, label='Women')

# Adding labels and title

plt.ylabel('Scores')

plt.title('Scores by group\nand gender')

# Adding group labels to the x-axis

plt.xticks(x, groups)

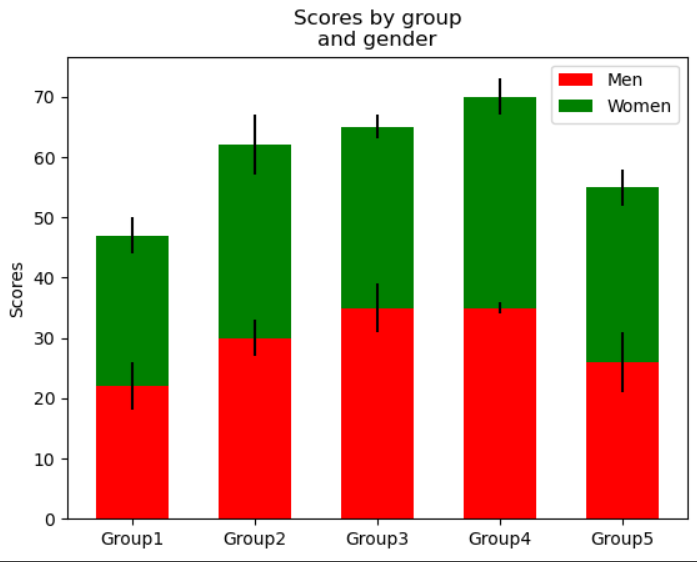
# Adding a legend

plt.legend()

# Display the plot

plt.show()

**OUTPUT:**



|  |
| --- |
| **Program #4.4.9 Date :** |
| **Write a Python program to create stack bar plot and add labels to each section.**  **Sample data:**  **people = ('G1','G2','G3','G4','G5','G6','G7','G8')**  **segments = 4**  **# multi-dimensional data**  **data = [[ 3.40022085, 7.70632498, 6.4097905, 10.51648577, 7.5330039, 7.1123587, 12.77792868, 3.44773477], [ 11.24811149, 5.03778215, 6.65808464, 12.32220677, 7.45964195, 6.79685302, 7.24578743, 3.69371847], [ 3.94253354, 4.74763549, 11.73529246, 4.6465543, 12.9952182, 4.63832778, 11.16849999, 8.56883433], [ 4.24409799, 12.71746612, 11.3772169, 9.00514257, 10.47084185, 10.97567589, 3.98287652, 8.80552122]]**  **Create the code snippet that gives the output shown in the following screenshot:** |

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

# Sample data

people = ('G1','G2','G3','G4','G5','G6','G7','G8')

segments = 4

data = np.array([

    [ 3.40022085, 7.70632498, 6.4097905, 10.51648577, 7.5330039, 7.1123587, 12.77792868, 3.44773477],

    [ 11.24811149, 5.03778215, 6.65808464, 12.32220677, 7.45964195, 6.79685302, 7.24578743, 3.69371847],

    [ 3.94253354, 4.74763549, 11.73529246, 4.6465543, 12.9952182, 4.63832778, 11.16849999, 8.56883433],

    [ 4.24409799, 12.71746612, 11.3772169, 9.00514257, 10.47084185, 10.97567589, 3.98287652, 8.80552122]

])

# Assign colors for each segment

colors = ['red', 'green', 'white', 'purple']

# Set up the bar positions

index = np.arange(len(people))

# Initialize the bottom positions for stacking

bottom = np.zeros(len(people))

# Create the plot

fig, ax = plt.subplots()

# Plot each segment

for i in range(segments):

    ax.barh(index, data[i], left=bottom, color=colors[i % len(colors)], label=f'Segment {i+1}')

    # Add text labels on each segment

    for j in range(len(people)):

        ax.text(bottom[j] + data[i][j]/2, j, f'{int(data[i][j])}%', ha='center', va='center', color='black')

    bottom += data[i]

# Add labels and title

ax.set(yticks=index, yticklabels=people, xlabel='Scores')

ax.set\_title('Stacked Bar Plot with Labels')

# Display the legend

ax.legend()

# Show the plot

plt.show()

**OUTPUT:**

****

|  |
| --- |
| **Program #4.4.10 Date :** |
| **Write a Python program to add textures (black and white) to bars and wedges. Note: Use the bottom to stack the women's bars on top of the men's bars. Create the code snippet that gives the output shown in the following screenshot:** |

**SOURCE CODE:**

import matplotlib.pyplot as plt

import numpy as np

# Sample data

men\_means = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

women\_means = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Define x-axis labels

x = np.arange(len(men\_means))

# Define hatch patterns (textures)

hatch\_patterns = ['/', '\\', '|', '-', '+', 'x', 'o', 'O', '.', '\*']

fig, ax = plt.subplots()

# Plot men's bars with black edge and white face

bars\_men = ax.bar(x, men\_means, color='white', edgecolor='black', hatch=hatch\_patterns)

# Plot women's bars on top of men's bars using the bottom parameter, also black and white

bars\_women = ax.bar(x, women\_means, bottom=men\_means, color='white', edgecolor='black', hatch=hatch\_patterns)

# Set labels and title

ax.set\_xlabel('X')

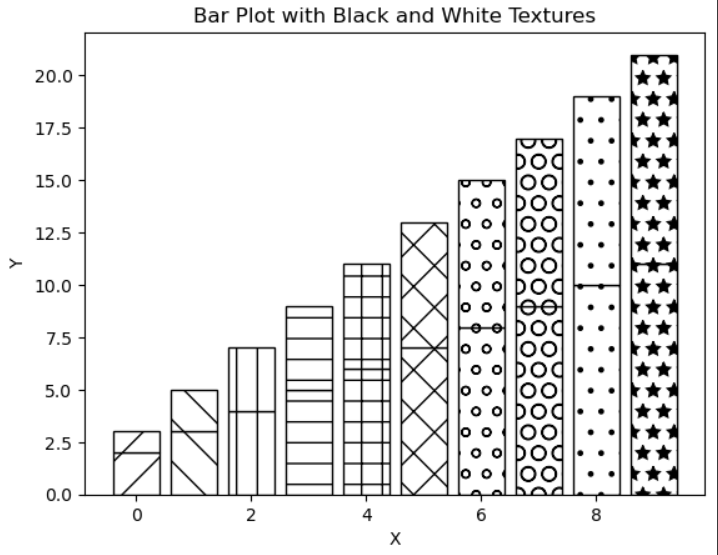
ax.set\_ylabel('Y')

ax.set\_title('Bar Plot with Black and White Textures')

# Display the plot

plt.show()

**OUTPUT**



|  |
| --- |
| **Program #5 Date :** |
| * 1. **Handle the given dataset (Data.csv) with adequate preprocessing steps mentioned and visualize the dataset with appropriate graphs.**      1. **Handle Missing Data Values**      2. **Encode the categorical data**   **5.1.3 Scale your features** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder, StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Load the dataset from the local CSV file

data = pd.read\_csv('Data.csv')  # Ensure the Data.csv file is in the same directory as the script

print("Original Dataset:\n", data)

# Step 5.1.1: Handle Missing Data Values

# Filling missing Age and Salary with the mean of the respective columns

data['Age'] = data['Age'].fillna(data['Age'].mean())  # Assign the result back to the 'Age' column

data['Salary'] = data['Salary'].fillna(data['Salary'].mean())  # Assign the result back to the 'Salary' column

print("\nDataset after handling missing values:\n", data)

# Step 5.1.2: Encode the categorical data

# Encoding 'Country' and 'Purchased' columns

label\_encoder = LabelEncoder()

# Encode Purchased (Yes/No) into 1/0

data['Purchased'] = label\_encoder.fit\_transform(data['Purchased'])

# Encode 'Country' using OneHotEncoding (creates separate columns for each category)

data = pd.get\_dummies(data, columns=['Country'], drop\_first=True)  # drop\_first to avoid dummy variable trap

print("\nDataset after encoding categorical variables:\n", data)

# Step 5.1.3: Scale your features

scaler = StandardScaler()

data[['Age', 'Salary']] = scaler.fit\_transform(data[['Age', 'Salary']])

print("\nDataset after scaling features:\n", data)

# Step 5.1.4: Visualization

# Visualization of the dataset using various plots

# Plot 1: Age distribution

plt.figure(figsize=(8, 4))

sns.histplot(data['Age'], kde=True)

plt.title('Age Distribution')

plt.show()

# Plot 2: Salary distribution

plt.figure(figsize=(8, 4))

sns.histplot(data['Salary'], kde=True, color='red')

plt.title('Salary Distribution')

plt.show()

# Plot 3: Count of Purchases (Purchased or Not)

plt.figure(figsize=(6, 4))

sns.countplot(x='Purchased', data=data)

plt.title('Count of Purchases (Yes/No)')

plt.show()

# Plot 4: Country-wise Purchase

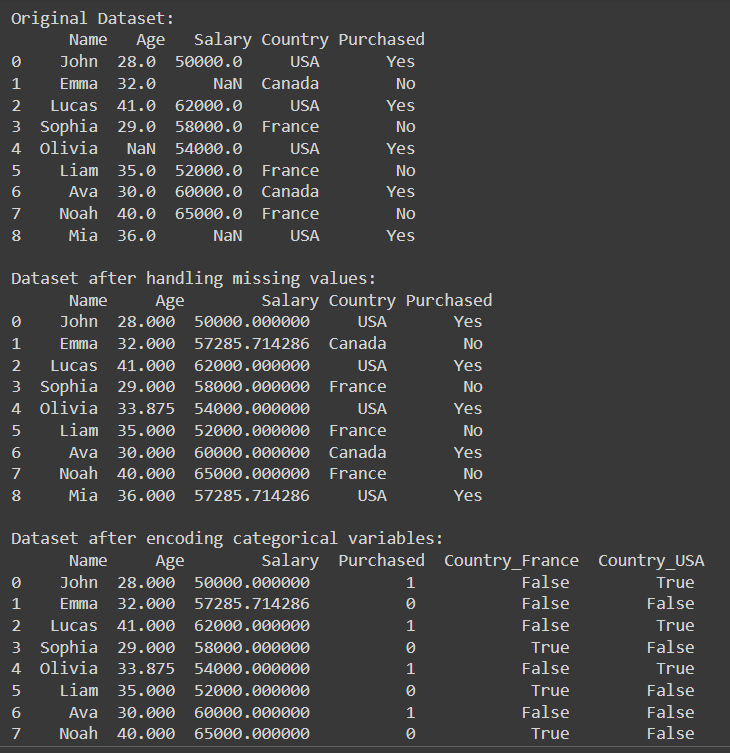
plt.figure(figsize=(8, 4))

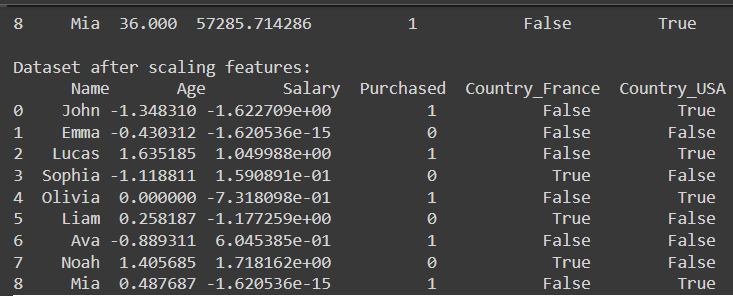
sns.barplot(x='Purchased', y='Salary', hue='Country\_France', data=data)

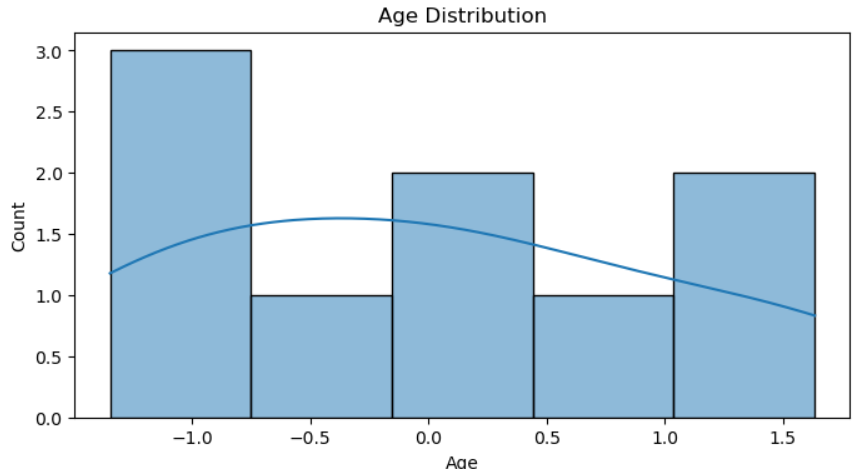
plt.title('Country-wise Salary of Purchasers')

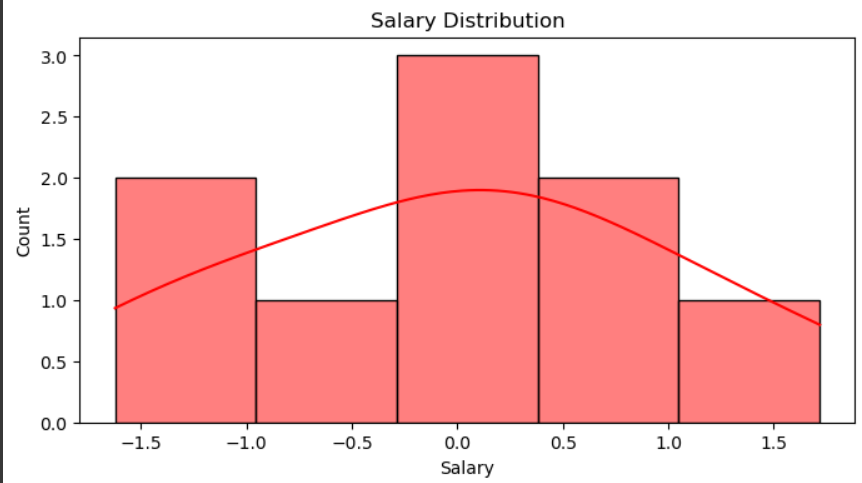
plt.show()

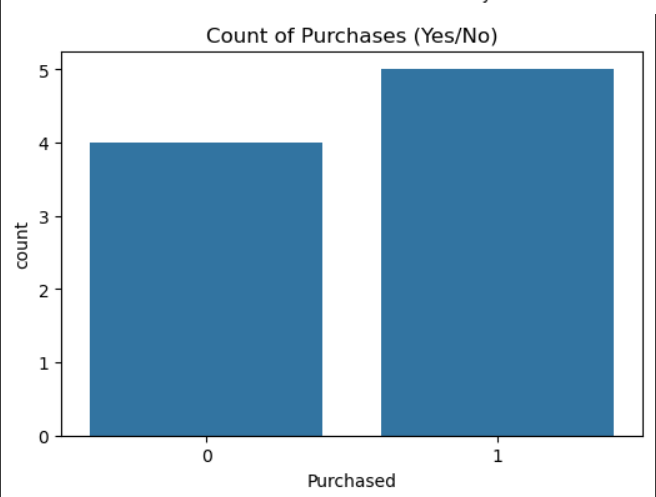
**OUTPUT:**

****

****

****

****

****

|  |
| --- |
| **Program #5.2 Date :** |
| **5.2. Using the given dataset (dirtydata.csv),**   * + 1. **Handle the data with empty cells (Use dropna() and fillna())**     2. **Replace the empty cells using mean, median, and mode.**     3. **Handle the data in the wrong format.**     4. **Handle the wrong data from the dataset.**     5. **Discover and remove duplicates.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

# Step 1: Load the dataset

data = pd.read\_csv('dirtydata.csv')

print("Original Dataset:\n", data)

# Step 5.2.1: Handle the data with empty cells

# Using dropna() - Drop rows with any missing values

data\_dropped = data.dropna()

print("\nDataset after dropping rows with missing values:\n", data\_dropped)

# Using fillna() - Fill missing values with a placeholder (e.g., 'Unknown')

data\_filled = data.fillna("Unknown")

print("\nDataset after filling missing values with 'Unknown':\n", data\_filled)

# Step 5.2.2: Replace empty cells using mean, median, and mode

# Filling missing 'Age' with mean, 'Salary' with median

data['Age'] = data['Age'].fillna(data['Age'].mean())

data['Salary'] = pd.to\_numeric(data['Salary'], errors='coerce')  # Convert invalid salary to NaN

data['Salary'] = data['Salary'].fillna(data['Salary'].median())

print("\nDataset after filling missing values using mean for Age and median for Salary:\n", data)

# Step 5.2.3: Handle the data in the wrong format

# Correcting the date format by replacing 'wrong\_date' with a valid date or marking it as NaT (Not a Time)

data['Date\_of\_Joining'] = pd.to\_datetime(data['Date\_of\_Joining'], errors='coerce')

print("\nDataset after handling wrong date format:\n", data)

# Step 5.2.4: Handle wrong data in the dataset

# For example, invalid 'Salary' values have been set to NaN in step 5.2.2 using pd.to\_numeric()

# You can also apply conditions to identify and correct wrong data

# Step 5.2.5: Discover and remove duplicates

# Checking for duplicates based on all columns

duplicates = data.duplicated()

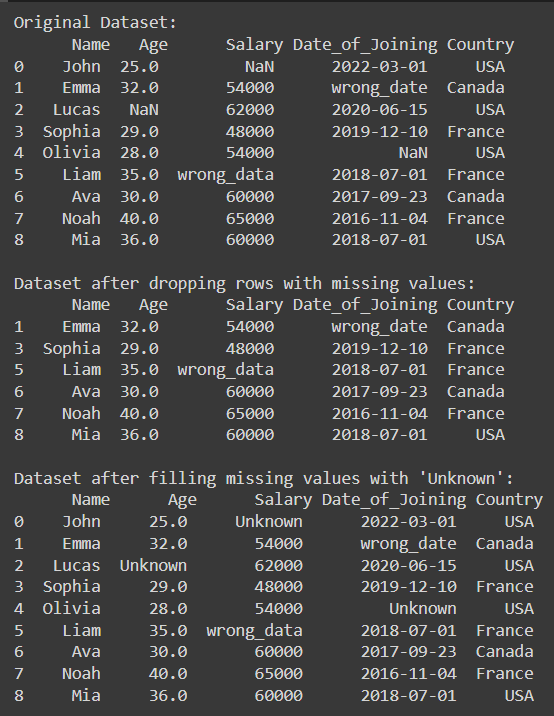
print("\nDuplicate entries:\n", data[duplicates])

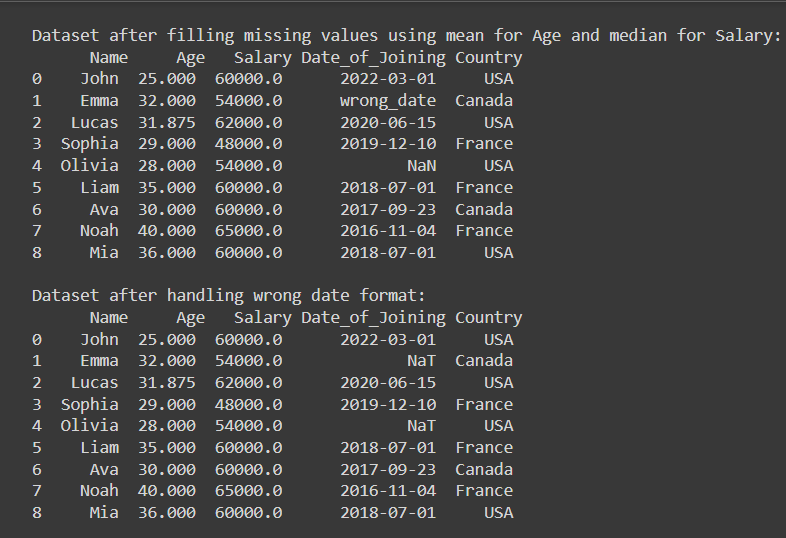
# Removing duplicates

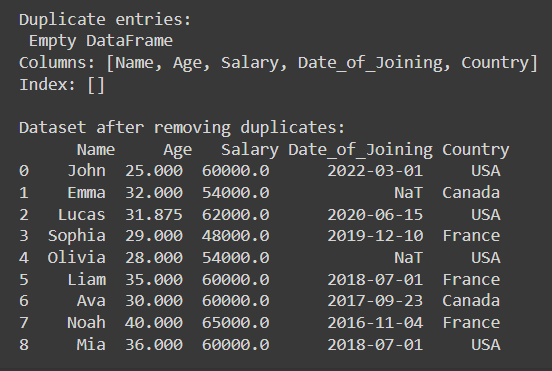
data\_cleaned = data.drop\_duplicates()

print("\nDataset after removing duplicates:\n", data\_cleaned)

**OUTPUT:**







|  |
| --- |
| **Program #5.3 Date :** |
| **Create a cricketer dataset using a dictionary of lists, and create a new attribute ‘Experience Category’ using ‘Age’ as the binning factor.** |

**SOURCE CODE:**

import pandas as pd

# Step 1: Create the cricketer dataset using a dictionary of lists

data = {

    'Name': ['Virat Kohli', 'Rohit Sharma', 'MS Dhoni', 'Sachin Tendulkar', 'Hardik Pandya', 'Ravindra Jadeja', 'Shubman Gill', 'Rishabh Pant', 'Yuvraj Singh'],

    'Age': [34, 36, 39, 47, 28, 32, 24, 26, 41],

    'Runs': [12000, 10000, 10500, 18426, 3500, 2400, 1800, 2500, 8700],

    'Wickets': [4, 8, 1, 154, 55, 185, 0, 0, 111],

    'Matches': [250, 200, 350, 463, 80, 100, 30, 50, 300]

}

# Convert the dictionary to a pandas DataFrame

cricketer\_df = pd.DataFrame(data)

print("Original Cricketer Dataset:\n", cricketer\_df)

# Step 2: Create 'Experience Category' using 'Age' as the binning factor

# Define the bins and labels for the experience category

bins = [0, 25, 35, 50]

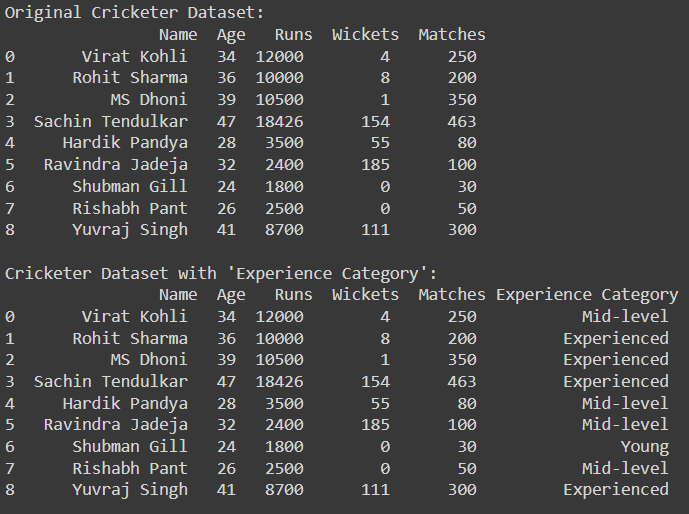
labels = ['Young', 'Mid-level', 'Experienced']

# Create the new column 'Experience Category' by binning the 'Age'

cricketer\_df['Experience Category'] = pd.cut(cricketer\_df['Age'], bins=bins, labels=labels, right=False)

print("\nCricketer Dataset with 'Experience Category':\n", cricketer\_df)

**OUTPUT:**

****

|  |
| --- |
| **Program #5.4 Date :** |
| **car\_age = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]**  **car\_speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]**  **Using the given dataset,**   * + 1. **Draw the line of linear regression**     2. **Evaluate how well the data fit in linear regression.**   **5.4.3 Predict the speed of a 10-year-old car.** |

**SOURCE CODE:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

# Data

car\_age = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])

car\_speed = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])

# Reshape data for sklearn

X = car\_age.reshape(-1, 1)

y = car\_speed

# Fit the model

model = LinearRegression().fit(X, y)

y\_pred = model.predict(X)

# Linear regression parameters

m = model.coef\_[0]

c = model.intercept\_

print(f'Regression Line: y = {m:.2f}x + {c:.2f}')

# Plot data points

plt.scatter(car\_age, car\_speed, color='blue', label='Data points')

# Plot regression line

x\_values = np.linspace(min(car\_age), max(car\_age), 100)

y\_values = m \* x\_values + c

plt.plot(x\_values, y\_values, color='red', label='Regression line')

plt.xlabel('Car Age')

plt.ylabel('Car Speed')

plt.title('Car Age vs Speed with Linear Regression Line')

plt.legend()

plt.show()

# Calculate R^2 score

r2 = r2\_score(y, y\_pred)

print(f'R^2 score: {r2:.2f}')

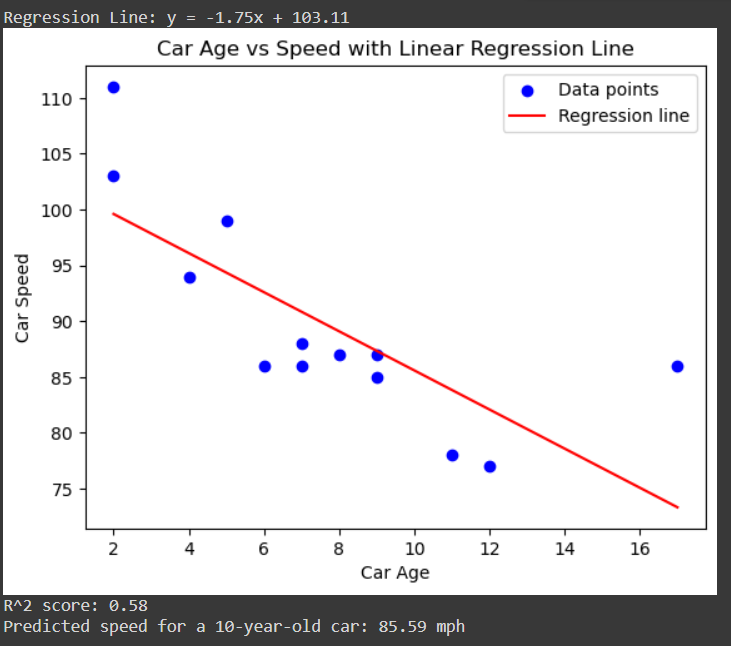
# Predict speed for a 10-year-old car

age\_to\_predict = 10

predicted\_speed = model.predict([[age\_to\_predict]])[0]

print(f'Predicted speed for a {age\_to\_predict}-year-old car: {predicted\_speed:.2f} mph')

**OUTPUT:**

****

|  |
| --- |
| **Program #5.5 Date :** |
| **Using the dataset (cars.csv),**   * + 1. **Predict the CO2 emissions of a car with a weight of 2300 kg and volume of 1300 cm3.**     2. **Print the coefficient values of the regression object.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

# Load dataset

df = pd.read\_csv('cars.csv')

# Display the first few rows of the dataframe to understand its structure

print(df.head())

# Extract features and target variable

X = df[['Weight', 'Volume']]

y = df['CO2\_Emissions']

# Create and fit the model

model = LinearRegression()

model.fit(X, y)

# Print the coefficients of the regression model

print(f'Coefficients: {model.coef\_}')

print(f'Intercept: {model.intercept\_}')

# Create a DataFrame for the prediction

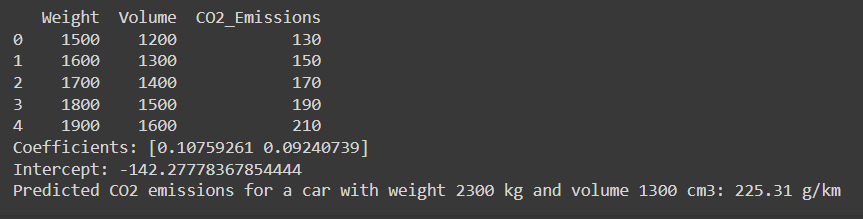
prediction\_data = pd.DataFrame({'Weight': [2300], 'Volume': [1300]})

# Predict the CO2 emissions for the car with the given weight and volume

predicted\_emissions = model.predict(prediction\_data)

print(f'Predicted CO2 emissions for a car with weight 2300 kg and volume 1300 cm3: {predicted\_emissions[0]:.2f} g/km')

**OUTPUT:**

****

|  |
| --- |
| **Program #5.6 Date :** |
| **Using the insurance dataset (insurance.csv) with adequate preprocessing steps,**   * + 1. **Visualize the correlation among variables using a heatmap.**     2. **Create a linear regression model.**     3. **Evaluate the model.  (Find MSE and R\_square.)**     4. **Predict the charges for a person with an age of 30, a BMI of 32.00, and who is a smoker.** |

**SOURCE CODE:**

# Step 5.6.1: Visualize the correlation among variables using a heatmap.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('insurance.csv')

# Show basic information about the dataset

print(df.info())

# Display the first few rows of the dataset

print(df.head())

# Preprocess the data

# Convert categorical columns to numerical if needed (e.g., using one-hot encoding)

df\_encoded = pd.get\_dummies(df)

# Calculate the correlation matrix

corr\_matrix = df\_encoded.corr()

# Plot the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

# Step 5.6.2: Create a linear regression model.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

df = pd.read\_csv('insurance.csv')

# Show basic information about the dataset

print(df.info())

print(df.head())

# Preprocess the data

# Convert categorical columns to numerical using one-hot encoding

df\_encoded = pd.get\_dummies(df)

# Define features and target

# Assume 'charges' is the target variable and the rest are features

X = df\_encoded.drop('charges', axis=1)  # Features

y = df\_encoded['charges']  # Target variable

# Split the data into training and testing sets

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

# Initialize and train the Linear Regression model

model = LinearRegression()

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

# Make predictions

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error (MSE): {mse}')

print(f'Mean Absolute Error (MAE): {mae}')

print(f'R-squared (R2): {r2}')

# Optional: Plotting actual vs. predicted values

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.xlabel('Actual Charges')

plt.ylabel('Predicted Charges')

plt.title('Actual vs. Predicted Charges')

plt.show()

# Step 5.6.3: Evaluate the model.  (Find MSE and R\_square.)

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

df = pd.read\_csv('insurance.csv')

# Preprocess the data

# Convert categorical columns to numerical using one-hot encoding

df\_encoded = pd.get\_dummies(df)

# Define features and target

X = df\_encoded.drop('charges', axis=1)  # Features

y = df\_encoded['charges']  # Target variable

# Split the data into training and testing sets

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

# Initialize and train the Linear Regression model

model = LinearRegression()

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

# Make predictions

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error (MSE): {mse}')

print(f'R-squared (R2): {r2}')

# Step 5.6.4 :Predict the charges for a person with an age of 30, a BMI of 32.00, and who is a smoker.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

df = pd.read\_csv('insurance.csv')

# Preprocess the data

# Convert categorical columns to numerical using one-hot encoding

df\_encoded = pd.get\_dummies(df)

# Define features and target

X = df\_encoded.drop('charges', axis=1)  # Features

y = df\_encoded['charges']  # Target variable

# Split the data into training and testing sets

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

# Initialize and train the Linear Regression model

model = LinearRegression()

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

# Retrieve feature names

feature\_names = X\_train.columns

# Define the input for the new person

# Create a DataFrame with all features, even if they are zero

new\_person = pd.DataFrame([{

    'age': 30,

    'bmi': 32.00,

    'sex\_female': 0,  # Example, adjust based on encoding

    'sex\_male': 1,    # Example, adjust based on encoding

    'smoker\_no': 0,   # Example, adjust based on encoding

    'smoker\_yes': 1,  # Example, adjust based on encoding

    'region\_northeast': 0,  # Example, adjust based on encoding

    'region\_northwest': 0,  # Example, adjust based on encoding

    'region\_southeast': 0,  # Example, adjust based on encoding

    'region\_southwest': 0,  # Example, adjust based on encoding

    'children': 0         # Add missing feature if necessary

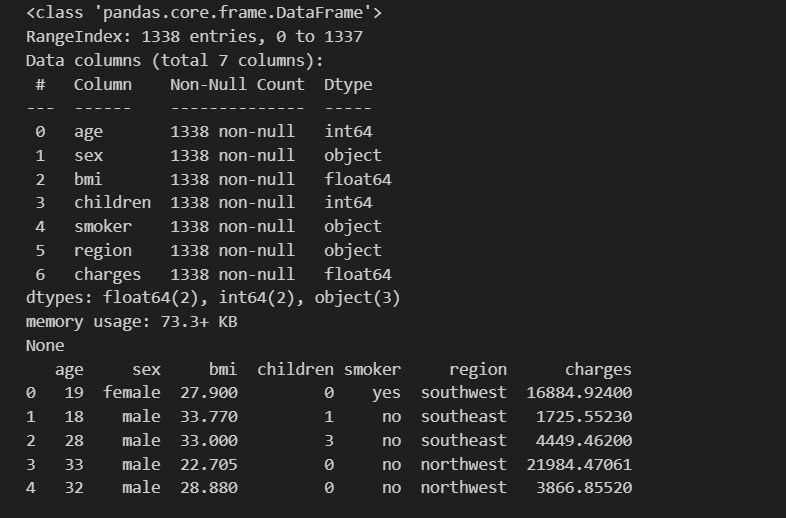
}], columns=feature\_names)

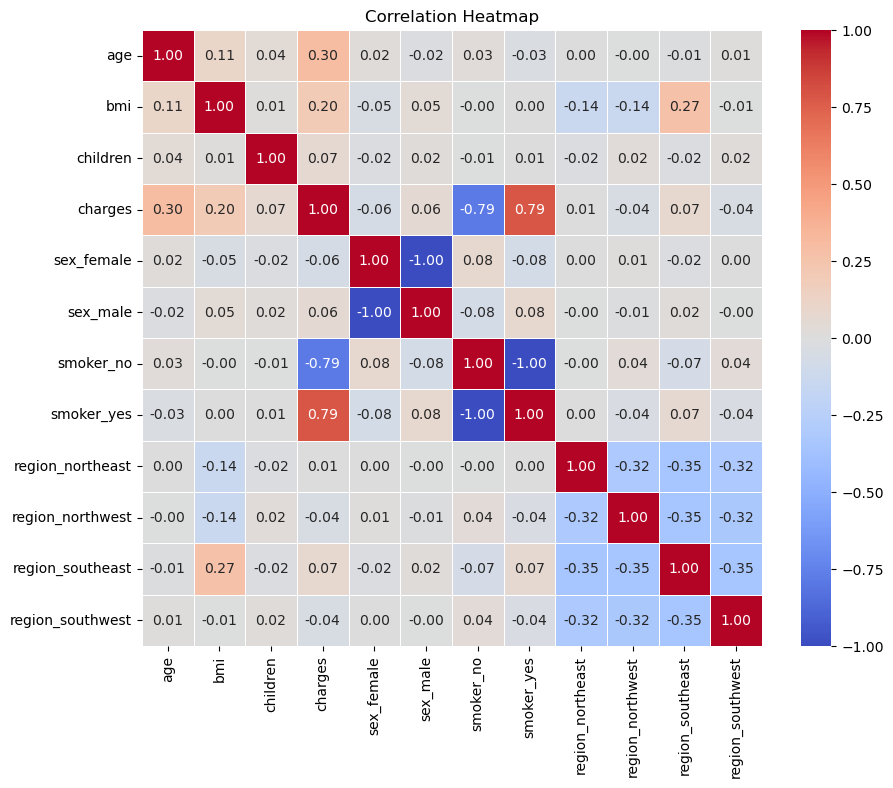
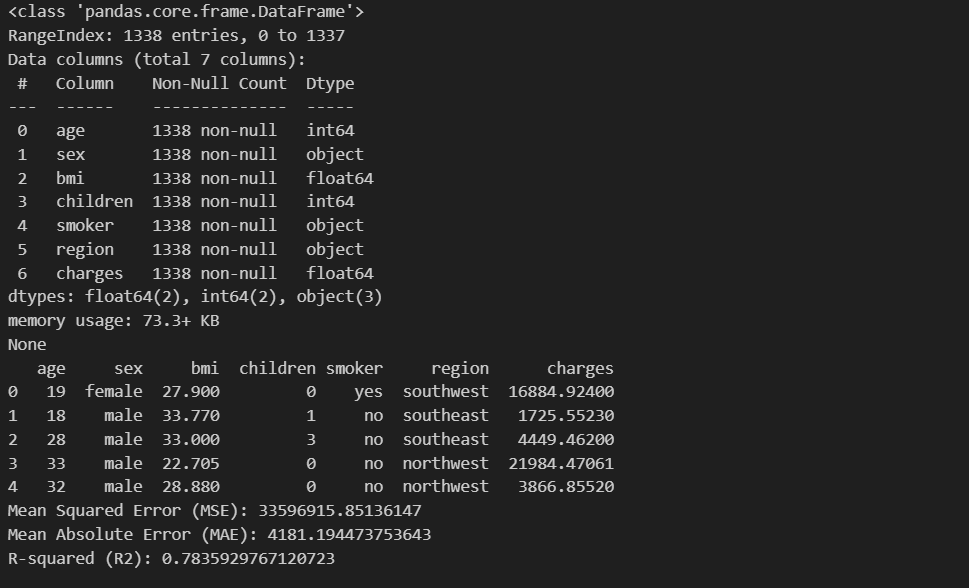
# Predict the charges for the new person

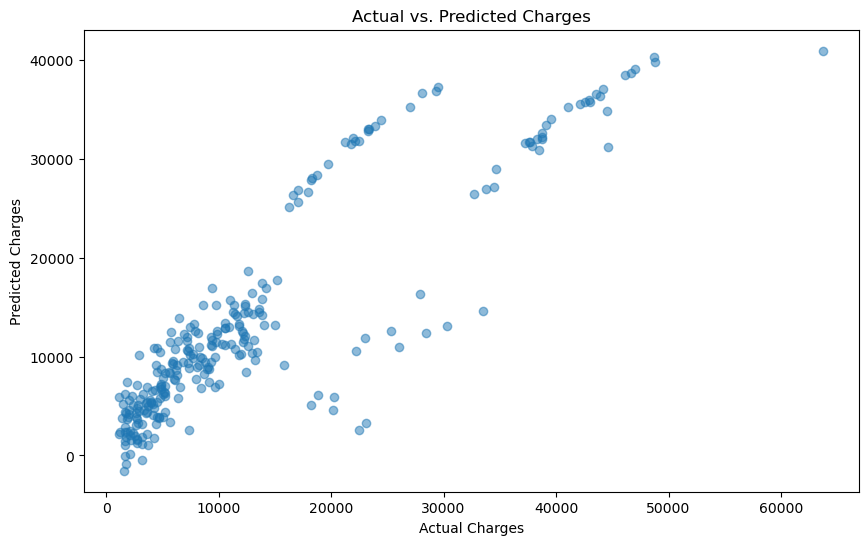
predicted\_charges = model.predict(new\_person)

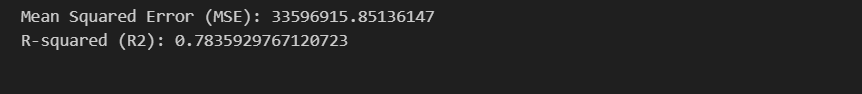
print(f'Predicted charges: ${predicted\_charges[0]:,.2f}')

**OUTPUT:**











|  |
| --- |
| **Program #5.7 Date :** |
| **Evaluate the dataset (User\_Data.csv) and predict whether a user will purchase the company's product or not. (Use logistic regression.)** |

**SOURCE CODE:**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Load the dataset

df = pd.read\_csv('User\_Data.csv')

# Inspect the dataset

print(df.info())

print(df.head())

# Preprocess the data

# Handle missing values (if any)

df = df.dropna()  # Dropping missing values; you may choose to impute instead

# Convert categorical columns to numerical using Label Encoding or One-Hot Encoding

label\_encoders = {}

for column in df.select\_dtypes(include=['object']).columns:

    le = LabelEncoder()

    df[column] = le.fit\_transform(df[column])

    label\_encoders[column] = le

# Define features and target

X = df.drop('Purchased', axis=1)  # Replace 'Purchased' with the actual name of the target column

y = df['Purchased']  # Replace 'Purchased' with the actual name of the target column

# Split the data into training and testing sets

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

# Standardize the features (optional but recommended for better performance)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Initialize and train the Logistic Regression model

model = LogisticRegression()

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

# Make predictions

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

# Evaluate the model

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

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

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

f1 = f1\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print(f'Precision: {precision:.2f}')

print(f'Recall: {recall:.2f}')

print(f'F1 Score: {f1:.2f}')

# Plot confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

            xticklabels=['Not Purchased', 'Purchased'],

            yticklabels=['Not Purchased', 'Purchased'])

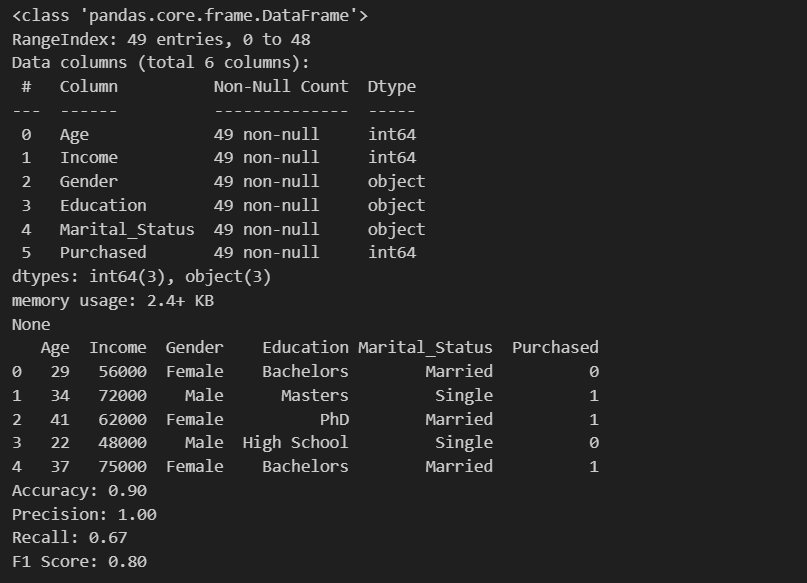
plt.xlabel('Predicted')

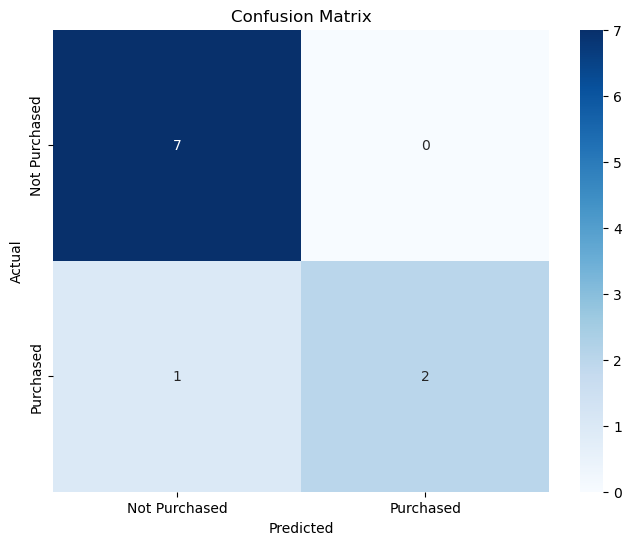
plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

**OUTPUT:**





|  |
| --- |
| **Program #5.8 Date :** |
| **Use the Iris dataset to visualize a decision tree with a depth=4 and save the plot as a PNG file. Also, print the confusion matrix and generate the classification report.** |

**SOURCE CODE:**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize and train the Decision Tree Classifier

clf = DecisionTreeClassifier(max\_depth=4, random\_state=42)

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

# Plot the decision tree

plt.figure(figsize=(20,10))

plot\_tree(clf, feature\_names=feature\_names, class\_names=target\_names, filled=True)

plt.title('Decision Tree with Depth=4')

plt.savefig('decision\_tree.png')  # Save the plot as a PNG file

plt.show()

# Make predictions

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

# Print the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# Print the classification report

class\_report = classification\_report(y\_test, y\_pred, target\_names=target\_names)

print("Classification Report:")

print(class\_report)

# Optionally, plot the confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=target\_names, yticklabels=target\_names)

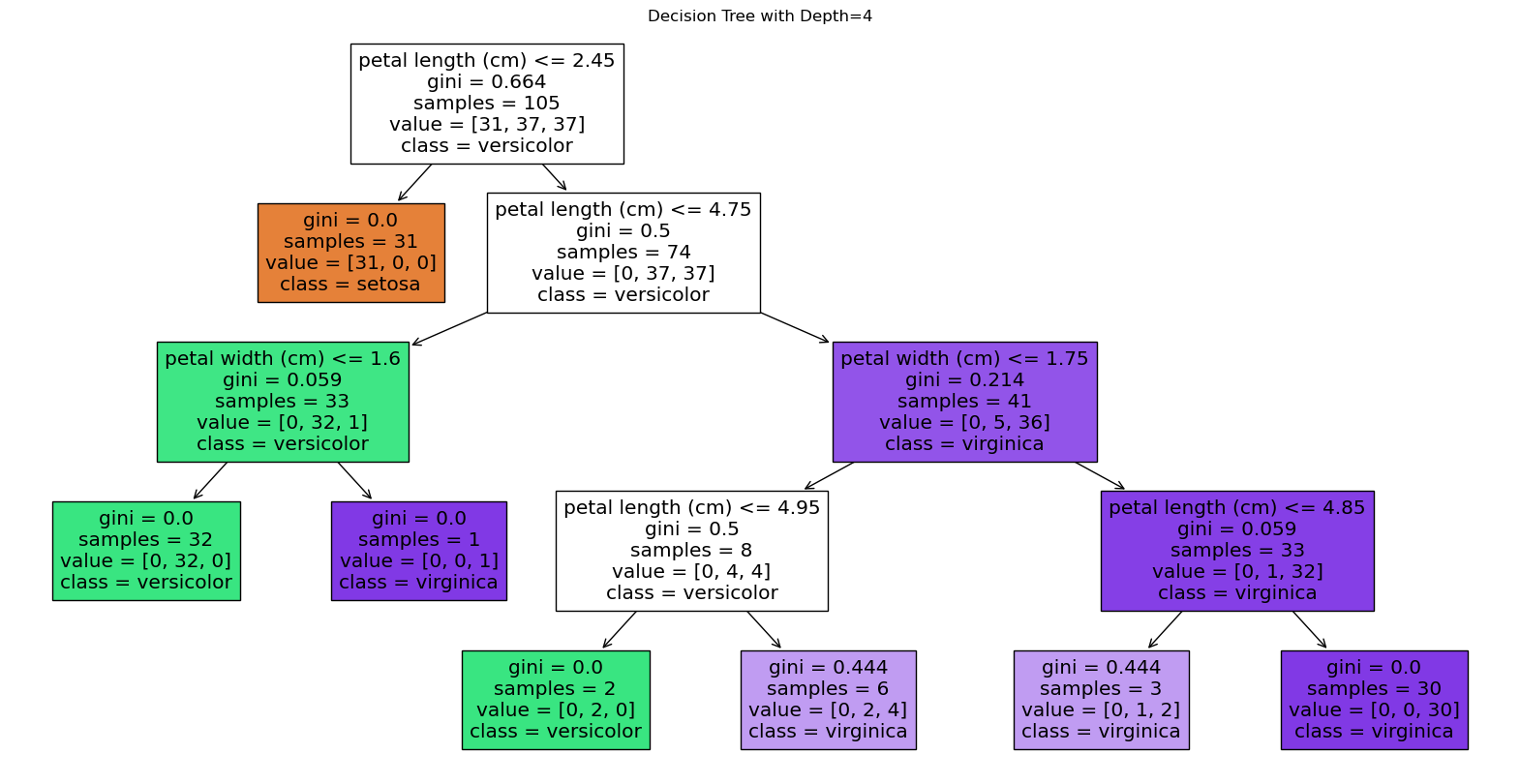
plt.xlabel('Predicted')

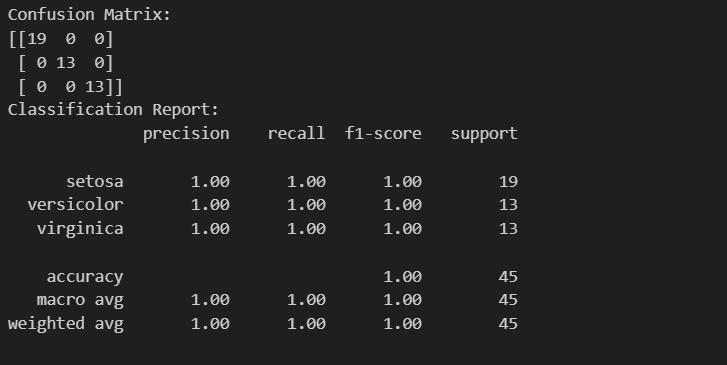
plt.ylabel('Actual')

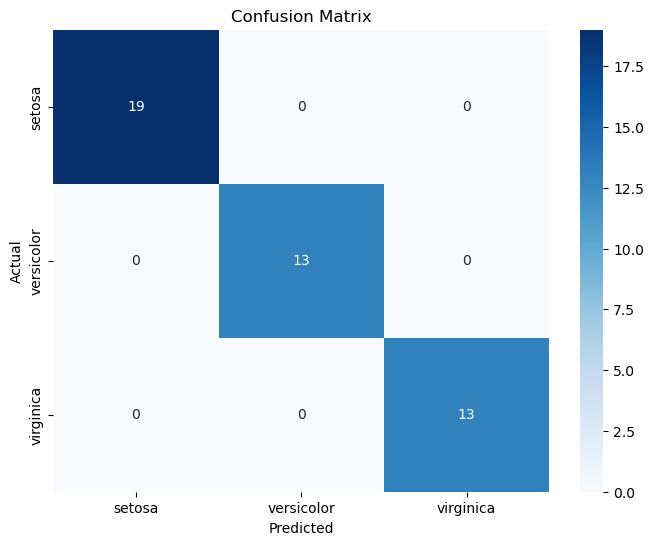
plt.title('Confusion Matrix')

plt.show()

**OUTPUT:**







|  |
| --- |
| **Program #5.9 Date :** |
| **Use the KNN algorithm to train the model and predict the future using the Iris dataset. Also, measure the accuracy of the model.** |

**SOURCE CODE:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize the KNN Classifier

# You can experiment with different values of k

k = 5  # Choose the number of neighbors

knn = KNeighborsClassifier(n\_neighbors=k)

# Train the model

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

# Make predictions

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

# Measure the accuracy

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

print(f'Accuracy of KNN model with k={k}: {accuracy:.2f}')

# Print the classification report

print("Classification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=target\_names))

# Print the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# Plot the confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=target\_names, yticklabels=target\_names)

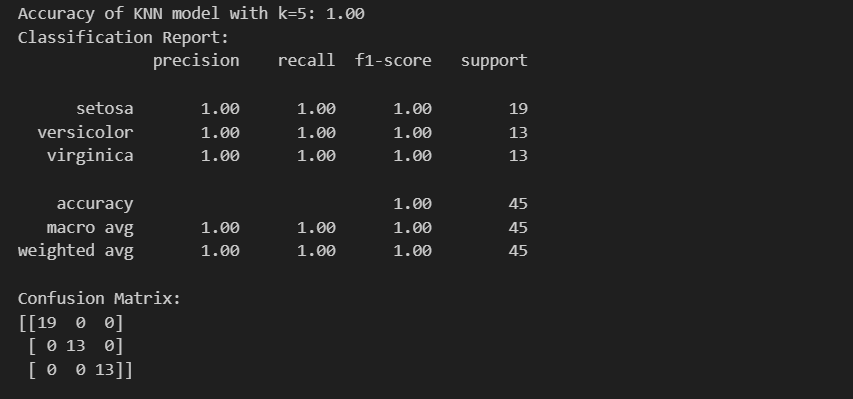
plt.xlabel('Predicted')

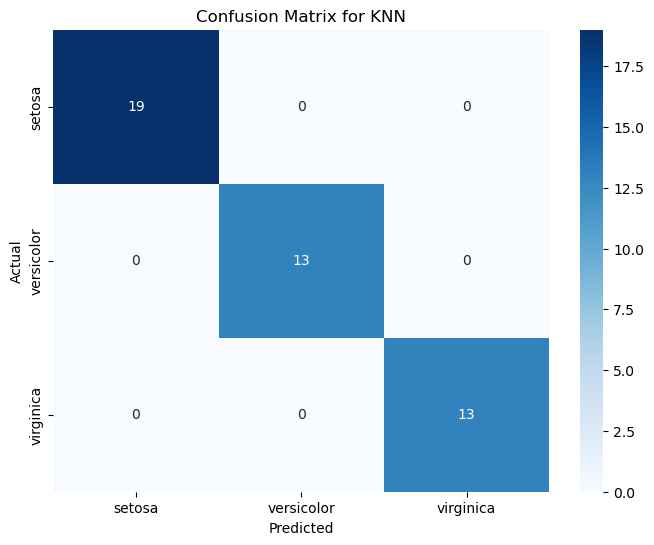
plt.ylabel('Actual')

plt.title('Confusion Matrix for KNN')

plt.show()

**OUTPUT:**





|  |
| --- |
| **Program #5.10 Date :** |
| **Analyze the given dataset (gym\_data.csv) using RandomForestRegressor and visualize the ‘Effect of n\_estimators.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

# Load the dataset

df = pd.read\_csv('gym\_data.csv')

# Inspect the dataset

print(df.head())

print(df.info())

# Assume that the dataset has columns 'feature1', 'feature2', ..., 'featureN', and 'target'

# You need to replace 'feature1', ..., 'featureN', and 'target' with the actual column names

X = df.drop('target', axis=1)  # Replace 'target' with the actual target column name

y = df['target']  # Replace 'target' with the actual target column name

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define different values for n\_estimators

n\_estimators\_list = [10, 50, 100, 200, 300]

mse\_list = []

# Train and evaluate RandomForestRegressor with different n\_estimators

for n in n\_estimators\_list:

    rf = RandomForestRegressor(n\_estimators=n, random\_state=42)

    rf.fit(X\_train, y\_train)

    y\_pred = rf.predict(X\_test)

    mse = mean\_squared\_error(y\_test, y\_pred)

    mse\_list.append(mse)

    print(f'n\_estimators: {n}, MSE: {mse:.2f}')

# Plot the effect of n\_estimators on MSE

plt.figure(figsize=(10, 6))

plt.plot(n\_estimators\_list, mse\_list, marker='o')

plt.xlabel('Number of Estimators')

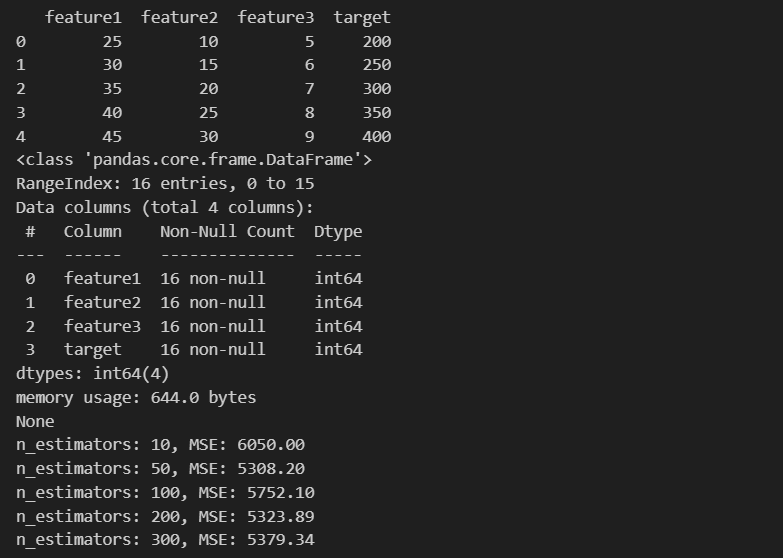
plt.ylabel('Mean Squared Error')

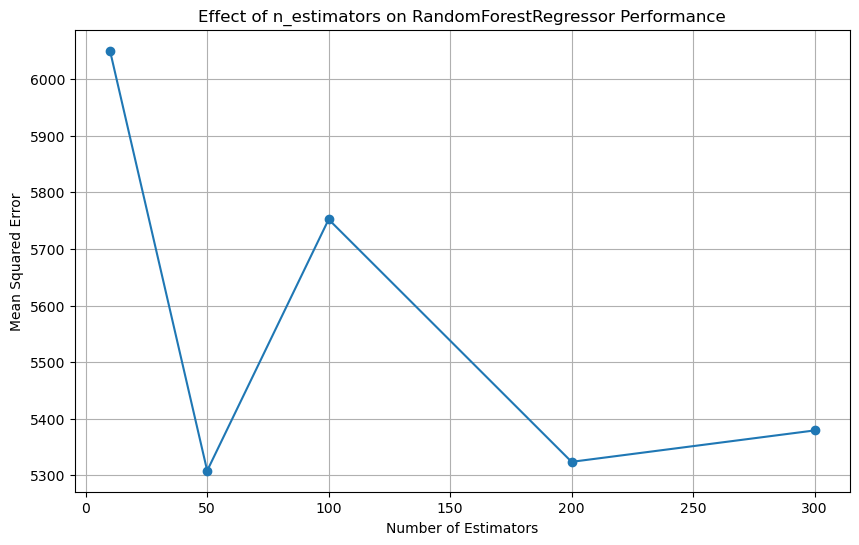
plt.title('Effect of n\_estimators on RandomForestRegressor Performance')

plt.grid(True)

plt.show()

**OUTPUT:**





|  |
| --- |
| **Program #5.11 Date :** |
| **Visualize a 3-dimensional cluster using the given dataset ‘Mall\_Customers.csv’, where no\_of\_clusters = 5.** |

**SOURCE CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

# Load the dataset

df = pd.read\_csv('Mall\_Customers.csv')

# Display the first few rows of the dataset

print(df.head())

# Select relevant features for clustering

# Ensure these columns exist in your dataset; adjust names if needed

X = df[['Annual Income (k$)', 'Spending Score (1-100)', 'Age']]

# Standardize the features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply K-Means clustering

n\_clusters = 5

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

# Reduce dimensions to 3D using PCA for visualization

pca = PCA(n\_components=3)

X\_pca = pca.fit\_transform(X\_scaled)

# Create a 3D scatter plot

fig = plt.figure(figsize=(12, 8))

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

# Plot the data points

scatter = ax.scatter(X\_pca[:, 0], X\_pca[:, 1], X\_pca[:, 2], c=df['Cluster'], cmap='viridis', marker='o')

# Add color bar

legend1 = ax.legend(\*scatter.legend\_elements(), title="Clusters")

ax.add\_artist(legend1)

# Label the axes and add a title

ax.set\_xlabel('Principal Component 1')

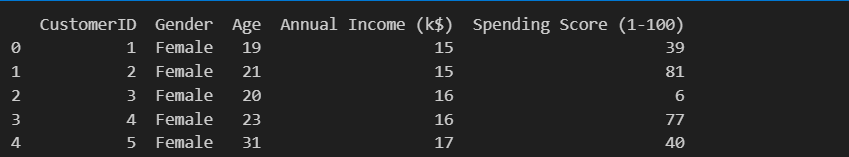
ax.set\_ylabel('Principal Component 2')

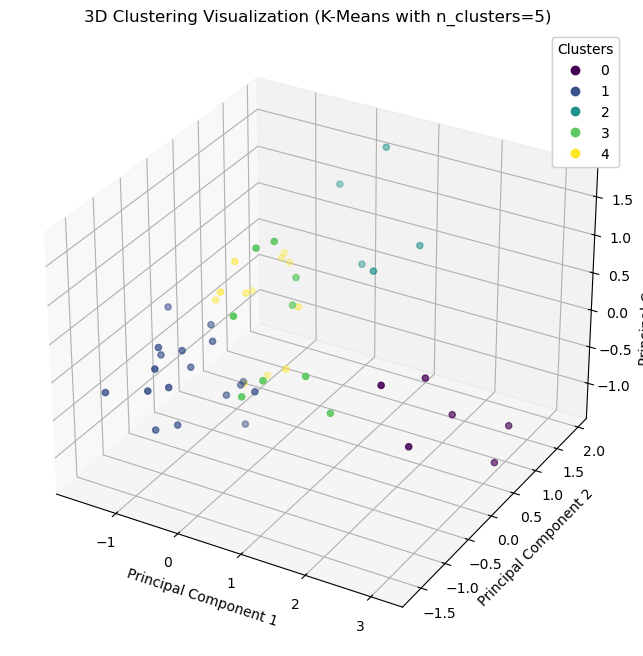
ax.set\_zlabel('Principal Component 3')

ax.set\_title('3D Clustering Visualization (K-Means with n\_clusters=5)')

plt.show()

**OUTPUT:**





|  |
| --- |
| **Program #5.12 Date :** |
| **Using the dataset provided (Online Retail.xlsx),**   * + 1. **Split the data according to the region of the transaction.**     2. **Build the models using the apriori algorithm.**     3. **Develop the association rules.**     4. **Find the most frequent items in any one of the regions.** |

**SOURCE CODE:**

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

# Load the CSV dataset

file\_path = 'Online Retail.csv'

df = pd.read\_csv(file\_path)

# Display the first few rows of the dataset

print(df.head())

# Split the data by region (assuming 'Country' is the region column)

regions = df['Country'].unique()

region\_dfs = {region: df[df['Country'] == region] for region in regions}

# Example: data for one region, e.g., 'United Kingdom'

uk\_data = region\_dfs['United Kingdom']

# Prepare the data for Apriori

def prepare\_data(df):

    df = df.dropna(subset=['InvoiceNo', 'Description'])

    basket = df.groupby(['InvoiceNo', 'Description'])['Quantity'].sum().unstack().reset\_index().fillna(0).set\_index('InvoiceNo')

    basket = basket.apply(lambda x: x > 0)  # Convert to boolean

    return basket

basket\_uk = prepare\_data(uk\_data)

# Apply the Apriori algorithm

frequent\_itemsets = apriori(basket\_uk, min\_support=0.01, use\_colnames=True)

print('Frequent Itemsets:')

print(frequent\_itemsets.head())

# Generate association rules

rules = association\_rules(frequent\_itemsets, metric='confidence', min\_threshold=0.5)

print('Association Rules:')

print(rules.head())

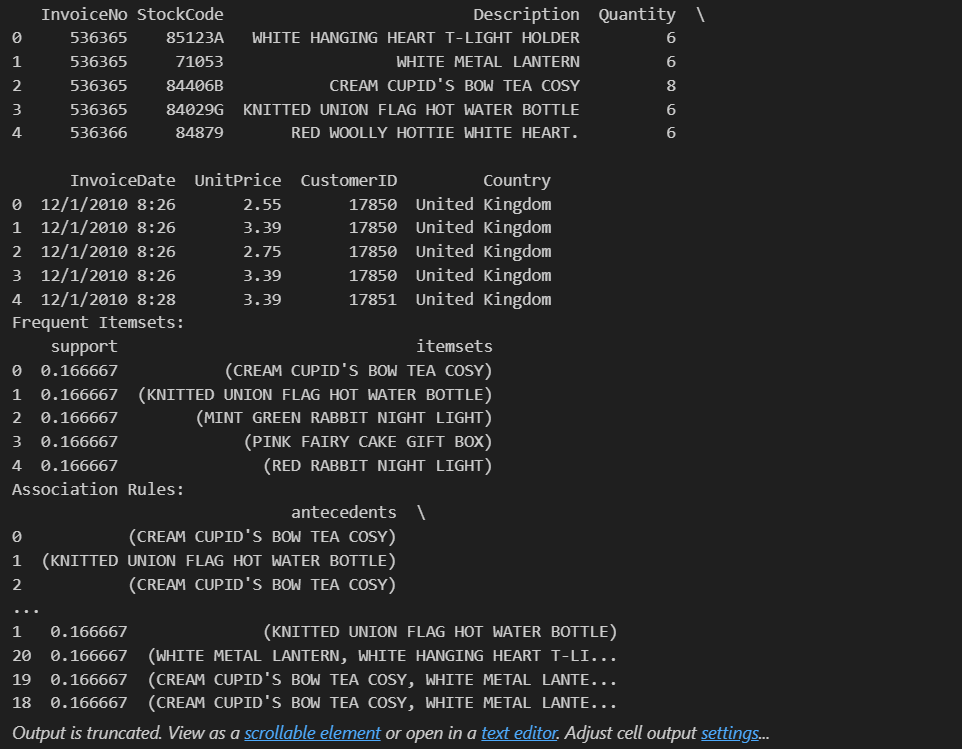
# Find the most frequent items

most\_frequent\_items = frequent\_itemsets.sort\_values(by='support', ascending=False)

print('Most Frequent Items:')

print(most\_frequent\_items.head())

**OUTPUT:**



|  |
| --- |
| **Project Date : 15/09/2024** |
| **Topic : Alcohol consumption and its impact on students** |

import numpy as np#for numerical computation

import pandas as pd#for handling data set

import matplotlib.pyplot as plt #plotting graph

from sklearn.model\_selection import train\_test\_split #for splitting data into

from sklearn.linear\_model import LinearRegression #for creatinng lenair

from sklearn import datasets

from sklearn.metrics import mean\_squared\_error,r2\_score

import seaborn as sns  #evaluating the model

from google.colab import drive

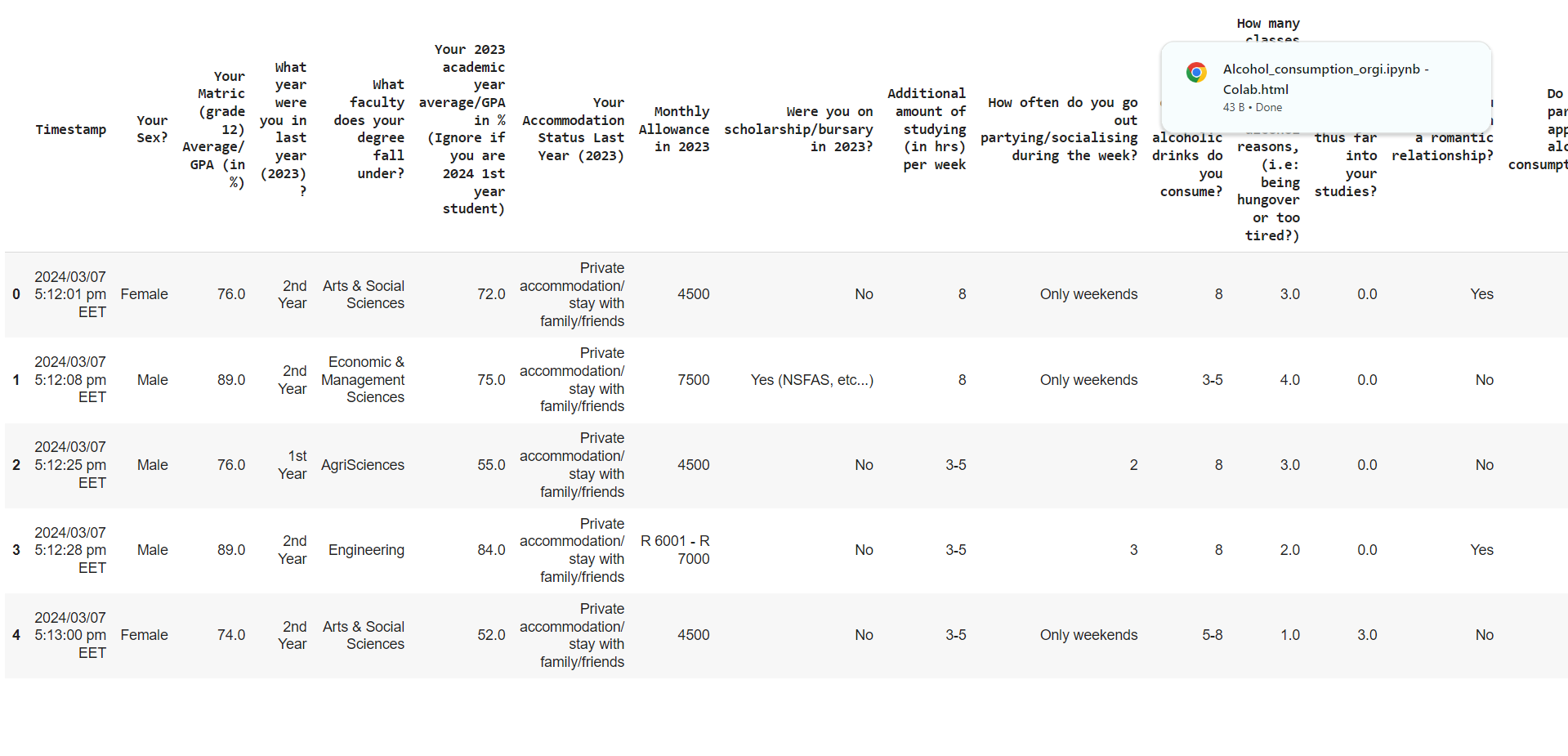
drive.mount('/content/drive')

#Load the dataset

file\_path='/content/drive/MyDrive/stats\_survey\_org.csv'

data=pd.read\_csv(file\_path)

data.head()



print(data.isnull().sum())

data.info()



# Impute missing values in categorical columns using mode

data['Your Sex?'] = data['Your Sex?'].fillna(data['Your Sex?'].mode()[0])

data['What faculty does your degree fall under?'] = data['What faculty does your degree fall under?'].fillna(data['What faculty does your degree fall under?'].mode()[0])

data['Were you on scholarship/bursary in 2023?'] = data['Were you on scholarship/bursary in 2023?'].fillna(data['Were you on scholarship/bursary in 2023?'].mode()[0])

data['Your Accommodation Status Last Year (2023)'] = data['Your Accommodation Status Last Year (2023)'].fillna(data['Your Accommodation Status Last Year (2023)'].mode()[0])

data['Do your parents approve alcohol consumption?'] = data['Do your parents approve alcohol consumption?'].fillna(data['Do your parents approve alcohol consumption?'].mode()[0])

data['Are you currently in a romantic relationship?'] = data['Are you currently in a romantic relationship?'].fillna(data['Are you currently in a romantic relationship?'].mode()[0])

data['How strong is your relationship with your parent/s?'] = data['How strong is your relationship with your parent/s?'].fillna(data['How strong is your relationship with your parent/s?'].mode()[0])

data['What year were you in last year (2023) ?']=data['What year were you in last year (2023) ?'].fillna(data['What year were you in last year (2023) ?'].mode()[0])

data['Monthly Allowance in 2023']=data['Monthly Allowance in 2023'].fillna(data['Monthly Allowance in 2023'].mode()[0])

# Impute missing values in categorical columns using median

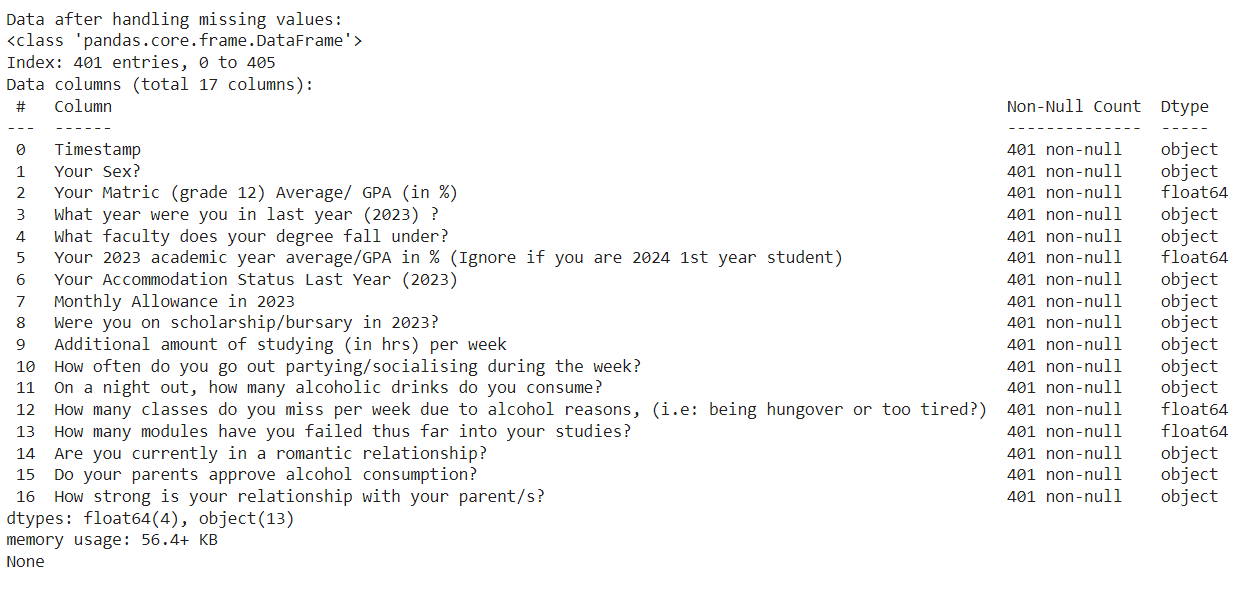
data['Your Matric (grade 12) Average/ GPA (in %)'] = data['Your Matric (grade 12) Average/ GPA (in %)'].fillna(data['Your Matric (grade 12) Average/ GPA (in %)'].mean())

data['Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)'] = data['Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)'].fillna(data['Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)'].mean())

data\_cleaned = data.dropna()

print("Data after handling missing values:")

print(data\_cleaned.info())



# Select only numeric columns

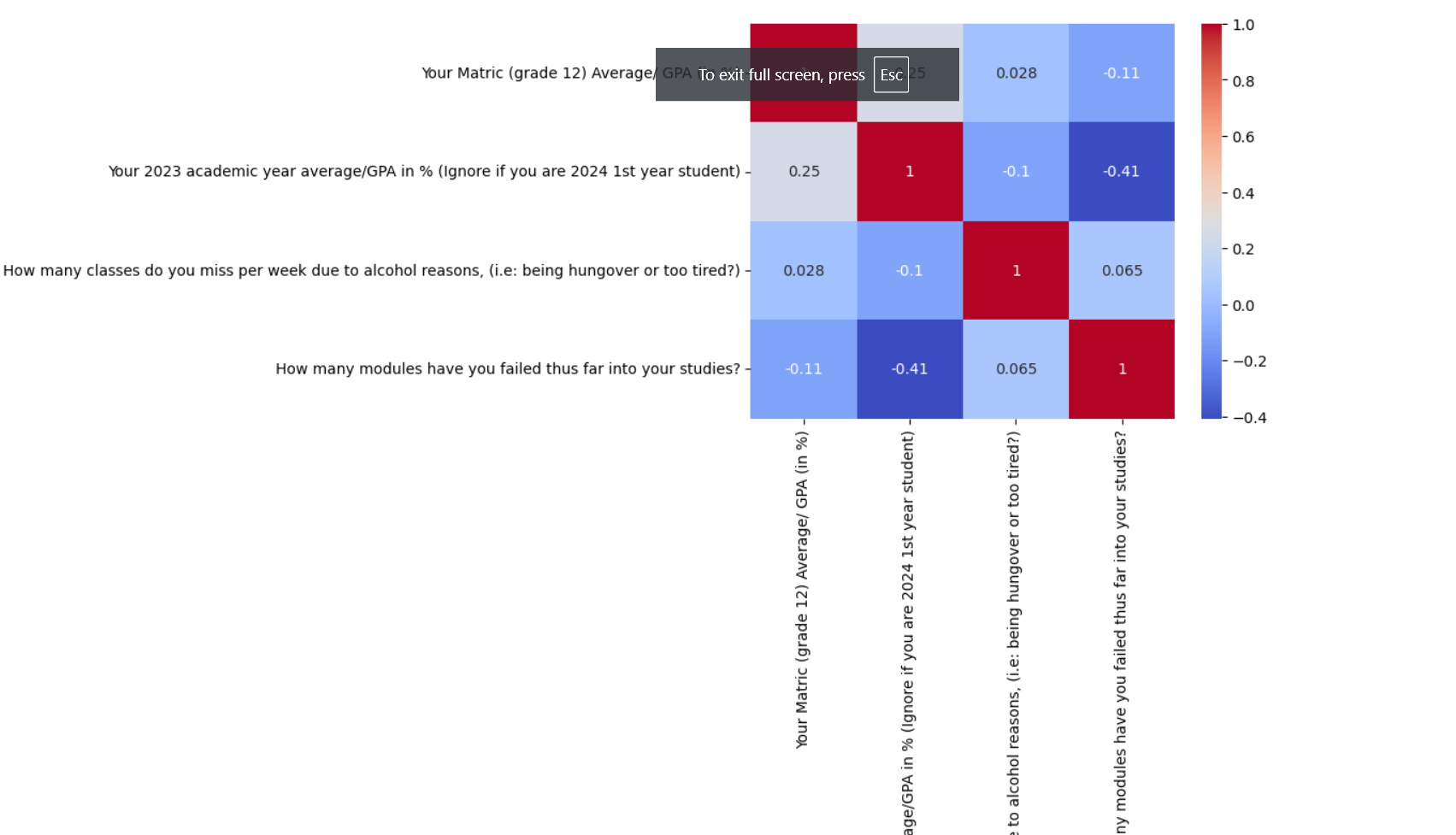
numerical\_data = data.select\_dtypes(include=['number'])

# Compute and visualize correlation matrix for numeric columns

corr\_matrix = numerical\_data.corr()

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

plt.show()



# Assuming 'data' is your pandas DataFrame with all the relevant columns

# List of required columns

required\_columns = [

'On a night out, how many alcoholic drinks do you consume?',

'Additional amount of studying (in hrs) per week',

'Monthly Allowance in 2023',

'How many classes do you miss per week due to alcohol reasons, (i.e: being hungover or too tired?)',

'Are you currently in a romantic relationship?',

'How often do you go out partying/socialising during the week?'

]

# Check for missing columns

missing\_columns = [col for col in required\_columns if col not in data.columns]

if missing\_columns:

print(f"Warning: The following required columns are missing: {missing\_columns}")

required\_columns = [col for col in required\_columns if col in data.columns]

# Define independent variables

X = data[required\_columns].copy()

# Convert 'On a night out, how many alcoholic drinks do you consume?' to numeric

def convert\_to\_numeric(value):

if isinstance(value, str) and '-' in value:

start, end = map(int, value.split('-'))

return (start + end) / 2

try:

return float(value)

except ValueError:

return None # Handle cases with invalid values

if 'On a night out, how many alcoholic drinks do you consume?' in X.columns:

X['On a night out, how many alcoholic drinks do you consume?'] = X['On a night out, how many alcoholic drinks do you consume?'].apply(convert\_to\_numeric)

# Label encoding categorical columns

label\_encoder = LabelEncoder()

if 'Are you currently in a romantic relationship?' in X.columns:

X['Are you currently in a romantic relationship?'] = label\_encoder.fit\_transform(X['Are you currently in a romantic relationship?'])

if 'How often do you go out partying/socialising during the week?' in X.columns:

X['How often do you go out partying/socialising during the week?'] = label\_encoder.fit\_transform(X['How often do you go out partying/socialising during the week?'])

# Define the dependent variable (target)

y = data['Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)'].copy()

# Convert GPA into a binary class (e.g., 1 for High GPA (>=60%), 0 for Low GPA (<60%))

y = y.apply(lambda x: 1 if x >= 63 else 0)

# Drop rows with NaN values

X = X.dropna()

y = y.loc[X.index] # Align y with X after dropping NaN rows

# Convert all values in X to numeric

X = X.apply(pd.to\_numeric, errors='coerce')

X = X.dropna() # Drop any remaining NaN rows

y = y.loc[X.index] # Align y with X again

# Ensure X and y have the same number of rows

print(f"Length of X: {len(X)}, Length of y: {len(y)}")

# Split the data into training and testing sets

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

# Initialize the logistic regression model

model = LogisticRegression(class\_weight='balanced')

# Fit the model on the training data

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

# Make predictions on the test data

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

# Calculate accuracy, precision, recall, F1 score, and confusion matrix

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

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

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

f1 = f1\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Print the results

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

print("Confusion Matrix:")

print(conf\_matrix)

# Example input for prediction

input\_data = pd.DataFrame([[3, 6, 5500, 1, 1]], columns=[

'On a night out, how many alcoholic drinks do you consume?',

'Additional amount of studying (in hrs) per week',

'Monthly Allowance in 2023',

'How many classes do you miss per week due to alcohol reasons, (i.e: being hungover or too tired?)',

'Are you currently in a romantic relationship?'

])

# Predicting probabilities and the class

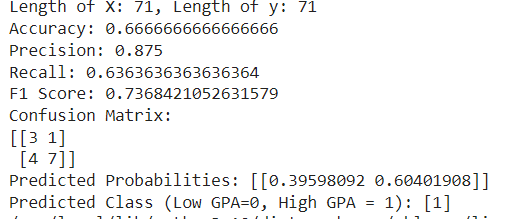
predicted\_probabilities = model.predict\_proba(input\_data)

predicted\_class = model.predict(input\_data)

# Output the results

print('Predicted Probabilities:', predicted\_probabilities)

print('Predicted Class (Low GPA=0, High GPA=1):', predicted\_class)



from sklearn.cluster import KMeans

# Define the features for clustering

features = data[['On a night out, how many alcoholic drinks do you consume?', 'Additional amount of studying (in hrs) per week']]

# Convert 'On a night out, how many alcoholic drinks do you consume?' to numeric

# Assuming the ranges represent an average, calculate the mid-point of the range

def convert\_to\_numeric(value):

  if isinstance(value, str) and '-' in value: # Check if the value is a string and contains '-'

    start, end = map(int, value.split('-'))

    return (start + end) / 2

  try:

    return float(value)

  except ValueError:

    return None # Handle cases with invalid values

# Apply the conversion to the correct column in features

features['On a night out, how many alcoholic drinks do you consume?'] = features['On a night out, how many alcoholic drinks do you consume?'].apply(convert\_to\_numeric)

# Instead of dropping rows, keep track of the original index

original\_index = features.index

# Convert features to numeric values before fitting

# Use errors='coerce' to handle any remaining non-numeric values and fillna to replace NaN with 0

features = features.apply(pd.to\_numeric, errors='coerce').fillna(0)

# Apply K-Means Clustering

kmeans = KMeans(n\_clusters=3)

kmeans.fit(features)

# Create a new Series for cluster labels with the original index

cluster\_labels = pd.Series(kmeans.labels\_, index=features.index)

# Reindex the cluster\_labels Series to match the original\_index

cluster\_labels = cluster\_labels.reindex(original\_index)

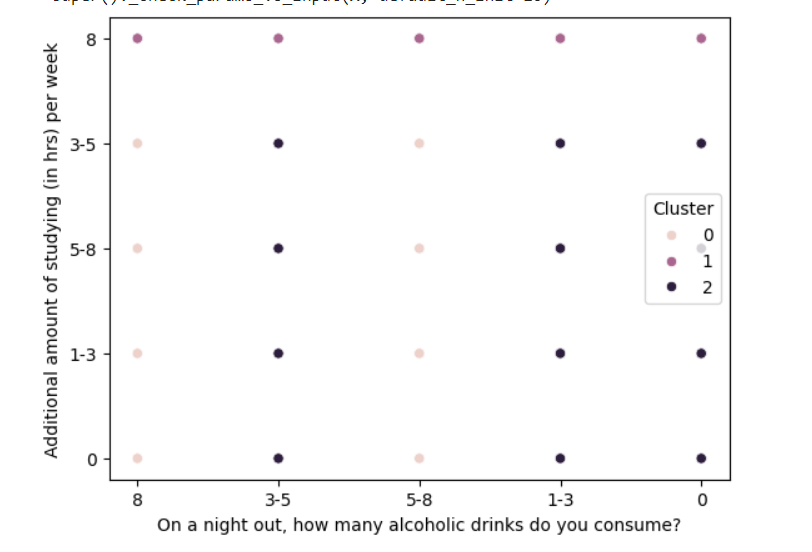
# Add cluster labels to the data using the new Series

data['Cluster'] = cluster\_labels

# Visualize the clusters

sns.scatterplot(x='On a night out, how many alcoholic drinks do you consume?', y='Additional amount of studying (in hrs) per week', hue='Cluster', data=data)

plt.show()



# Histogram of GPA

data['Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)'].hist()

plt.show()

# Scatter plot of alcohol consumption vs. GPA

sns.scatterplot(x='On a night out, how many alcoholic drinks do you consume?', y='Your 2023 academic year average/GPA in % (Ignore if you are 2024 1st year student)', data=data) # Use the correct column name from the DataFrame

plt.show()

