DEPARTMENT OF COMPUTER SCIENCE
RAJAGIRI COLLEGE OF SOCIAL SCIENCES
(Autonomous)
KALAMASSERY - KOCHI - 683104

MASTER OF COMPUTER APPLICATIONS

DATA ANALYSIS USING PYTHON LAB RECORD

NAME : KHADEEJA BEEVI C N

SEMESTER : THIRD

REGISTER NO. : 23204027

DEPARTMENT OF COMPUTER SCIENCE
RAJAGIRI COLLEGE OF SOCIAL SCIENCES
(Autonomous)
KALAMASSERY - KOCHI – 683104

MASTER OF COMPUTER APPLICATIONS

CERTIFICATE

NAME : KHADEEJA BEEVI C N

SEMESTER : THIRD

REGISTER NO. : 23204027

*Certified that this is a bonafide record of work done by **Khadeeja Beevi C N** in the Software Laboratory of Department of Computer Science, Rajagiri College of Social Sciences, Kalamassery.*

Mr. Diljith K Benny
Faculty in Charge

Dr.Bindiya M Varghese
Dean, Computer Science

Internal Examiner

External Examiner

Place : Kalamassery
Date : 22-09-2024

Table of Contents

Program #1.1	Date :
1.1 Create a simple calculator in Python.	

SOURCE CODE :

```
def add(x,y):
   return x+y

def sub(x,y):
   return x - y

def mul(x,y):
   return x*y

def div(x,y):
   return x/y

print("\n================================")
print("Select Operation:")
print("1.add")
print("2.sub")
print("3.mul")
print("4.div")

while True:
  choice = input("Enter yor choice (1/2/3/4)")

  if choice in ("1", "2", "3", "4"):
    try:
       num1 = float(input("Enter your first num:"))
       num2 = float(input("Enter your second num"))
    except ValueError:
       print("Invalid input. Enter a valid number")
       continue

    if choice == "1":
       print(num1, "+" , num2 , "=" ,add(num1,num2))
       continue
    elif choice == "2":
       print(num1 , "-" , num2 , "=" , sub(num1,num2))
       continue
    elif choice == "3":
       print(num1 , "*" , num2 , "=" , mul(num1,num2))
       continue
```

```
    elif choice == "4":
        print(num1 , "/" , num2 , "=" , div(num1,num2))
        continue

    nextCalcu = input("next calculation:(yes/no)")
    if nextCalcu == "no":
        break

else:
    print("Invalid input")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.1.py

==============================
Select Operation:
1.add
2.sub
3.mul
4.div
Enter yor choice (1/2/3/4)1
Enter your first num:3
Enter your second num6
3.0 + 6.0 = 9.0
Enter yor choice (1/2/3/4)2
Enter your first num:6
Enter your second num2
6.0 - 2.0 = 4.0
Enter yor choice (1/2/3/4)3
Enter your first num:6
Enter your second num2
6.0 * 2.0 = 12.0
Enter yor choice (1/2/3/4)4
Enter your first num:3
Enter your second num1
3.0 / 1.0 = 3.0
```

Program #1.2	Date :
An electric power distribution company charges domestic customers as follows: Conabove Rs 1.0f nj surcharge of 15% will be charged, and the minimum bill should be Rs. 100/-sumption unit Rate of charge	

SOURCE CODE:

```
def calculate_bill(units_consumed):
    # Rates based on unit consumption
    if units_consumed <= 100:
        total_bill = units_consumed * 3
    elif units_consumed <= 200:
        total_bill = (100 * 3) + (units_consumed - 100) * 5
    else:
        total_bill = (100 * 3) + (100 * 5) + (units_consumed - 200) * 7

    # Check if surcharge is needed (if units > 300)
    if units_consumed > 300:
        surcharge = total_bill * 0.15
        total_bill += surcharge

    # Minimum bill condition
    if total_bill < 100:
        total_bill = 100

    return total_bill

# Input from user
units = float(input("Enter the number of units consumed: "))
bill = calculate_bill(units)
print(f"Total bill: Rs. {bill:.2f}")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.2.py
Enter the number of units consumed: 200
Total bill: Rs. 800.00
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.3	Date :
0-200 Rs.0.50 p0per unit excess of 600 **1.3.1 If the bill exceeds Rs. 400, then a scenario mentioned above.**	

SOURCE CODE :

```
def calculate_bill(units_consumed):
    # Base rates for units consumed
    if units_consumed <= 200:
        total_bill = units_consumed * 0.50
    else:
        total_bill = (200 * 0.50) + (units_consumed - 200) * 1.00

    # If the bill exceeds Rs. 400, add 15% surcharge
    if total_bill > 400:
        surcharge = total_bill * 0.15
        total_bill += surcharge

    # Minimum bill condition
    if total_bill < 100:
        total_bill = 100

    return total_bill

# Input from user
units = float(input("Enter the number of units consumed: "))
bill = calculate_bill(units)
print(f"Total bill: Rs. {bill:.2f}")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.3.py
Enter the number of units consumed: 250
Total bill: Rs. 150.00
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.4	Date :
Print the pyramid oer unit **1.4.1 201-400 Rs. 0.65 per unit in excess of 200** **1.4.2 401-600 Rs 0.80 per unit excess of 400** **1.4.3 601 and**	

SOURCE CODE :

1.4.1

```
def print_pyramid_for_units():
    print("   0-200 units: Rs. 0.50 per unit")
    print("  201-400 units: Rs. 0.65 per unit (in excess of 200)")

    # Printing a pyramid to visually represent the range of 201-400 units
    rows = 5
    for i in range(1, rows + 1):
        # Spaces to align the pyramid
        print(' ' * (rows - i), end='')

        # Stars to represent units in excess of 200 (201-400 range)
        print('*' * (2 * i - 1))

    print("  Represents the range of 201-400 units at Rs. 0.65 per unit")

# Call the function to print the pyramid
print_pyramid_for_units()
```

OUTPUT

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.4_1.py
   0-200 units: Rs. 0.50 per unit
  201-400 units: Rs. 0.65 per unit (in excess of 200)
    *
   ***
  *****
 *******
*********
  Represents the range of 201-400 units at Rs. 0.65 per unit
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

1.4.2

```
def print_pyramid_for_401_600_units():
    print("   0-200 units: Rs. 0.50 per unit")
    print("  201-400 units: Rs. 0.65 per unit (in excess of 200)")
    print("  401-600 units: Rs. 0.80 per unit (in excess of 400)")

    # Printing a pyramid to visually represent the range of 401-600 units
    rows = 6
    for i in range(1, rows + 1):
        # Spaces to align the pyramid
        print(' ' * (rows - i), end='')

        # Stars to represent units in excess of 400 (401-600 range)
        print('*' * (2 * i - 1))

    print("  Represents the range of 401-600 units at Rs. 0.80 per unit")

# Call the function to print the pyramid
print_pyramid_for_401_600_units()
```

OUTPUT

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.4_2.py
   0-200 units: Rs. 0.50 per unit
  201-400 units: Rs. 0.65 per unit (in excess of 200)
  401-600 units: Rs. 0.80 per unit (in excess of 400)
     *
    ***
   *****
  *******
 *********
***********
  Represents the range of 401-600 units at Rs. 0.80 per unit
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

1.4.3

```
def print_pyramid_for_601_above_units():
    print("  601+ units: Rs. 1.00 per unit (in excess of 600)")

    # Printing a pyramid to visually represent units above 600
    rows = 7
    for i in range(1, rows + 1):
        # Spaces to align the pyramid
        print(' ' * (rows - i), end='')

        # Stars to represent units in excess of 600
        print('*' * (2 * i - 1))

    print("  Represents the range of 601+ units at Rs. 1.00 per unit")

# Call the function to print the pyramid
print_pyramid_for_601_above_units()
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.4_3.py
  601+ units: Rs. 1.00 per unit (in excess of 600)
      *
     ***
    *****
   *******
  *********
 ***********
*************
  Represents the range of 601+ units at Rs. 1.00 per unit
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.5	Date :
Create a Python program based on the numbers using *for* loops	

SOURCE CODE :

```
def print_numerical(rows):
    for i in range(1, rows + 1):
        for _ in range(rows-i):
            print(" ",end="")
        for j in range(1,2*i):
            print(j,end="")
        print()
while True:
    try:
        rows=(int(input("Enter the num of rows for the pyramid:")))
        if rows<=0:
            print("Please enter a positive integer")

        else:
            break
    except ValueError:
        print("Inavlid Input")
print_numerical(rows)
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.5.py
Enter the num of rows for the pyramid:6
     1
    123
   12345
  1234567
 123456789
1234567891011
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.6	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 :

```
def findSum():
    count =0
    sum =0
    for i in range(101,200):
        if i % 7 == 0:
            count +=1
            sum +=1
    print("Numbers of integers divisible by 7 between 100 and 200: ",count)
    print("Sum of integers divisible by 7 between 100 and 200: ",sum)
findSum()
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.6.py
Numbers of integers divisible by 7 between 100 and 200:  14
Sum of integers divisible by 7 between 100 and 200:  14
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.7	Date :
Write a recursive function to calculate the sum of numbers from 0 to 10	

SOURCE CODE :

```
def sum(x):
    if x==0:
        return 0
    else:
        return x +sum(x-1)
result = sum(10)
print("The sum of num from 0 to 10:", result)
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.7.py
The sum of num from 0 to 10: 55
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.8	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 rev_num(n):
    return int(str(n)[::-1])
def is_paliandrome(n):
    return str(n) == str(n)[::-1]
def add_paliandrome(n):
    while not is_paliandrome(n):
        revd_num = rev_num(n)
        print(f"{n} + {revd_num} = {n + revd_num}")
        n+=revd_num
    return n
number = int(input("Enter a num:"))
result = add_paliandrome(number)
print(f"The paliandrome is:{result}")
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.8.py
Enter a num:235
235 + 532 = 767
The paliandrome is:767
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.9	Date :
Write a menu-driven program that performs the following operations on strings	
1.9.1 Check if the String is a Substring of Another String	
1.9.2 Count Occurrences of Character	
1.9.3 Replace a substring with another substring	
1.9.4 Convert to Capital Letters	

SOURCE CODE :

```
# Function to check if one string is a substring of another
def check_substring(main_string, sub_string):
    if sub_string in main_string:
        print(f"'{sub_string}' is a substring of '{main_string}'")
    else:
        print(f"'{sub_string}' is not a substring of '{main_string}'")

# Function to count occurrences of a character in a string
def count_occurrences(string, char):
    count = string.count(char)
    print(f"The character '{char}' appears {count} times in the string.")

# Function to replace a substring with another substring
def replace_substring(string, old_substring, new_substring):
    new_string = string.replace(old_substring, new_substring)
    print(f"After replacing '{old_substring}' with '{new_substring}', the new string is: {new_string}")

# Function to convert the string to uppercase
def convert_to_upper(string):
    upper_string = string.upper()
    print(f"The string in uppercase is: {upper_string}")

# Main program
def main():
    while True:
        print("\nMenu:")
```

```
print("1. Check if a String is a Substring of Another String")
print("2. Count Occurrences of a Character")
print("3. Replace a Substring with Another Substring")
print("4. Convert String to Uppercase")
print("5. Exit")

# Take user choice
choice = input("Enter your choice (1-5): ")

if choice == '1':
    main_string = input("Enter the main string: ")
    sub_string = input("Enter the substring to check: ")
    check_substring(main_string, sub_string)

elif choice == '2':
    string = input("Enter the string: ")
    char = input("Enter the character to count: ")
    count_occurrences(string, char)

elif choice == '3':
    string = input("Enter the string: ")
    old_substring = input("Enter the substring to replace: ")
    new_substring = input("Enter the new substring: ")
    replace_substring(string, old_substring, new_substring)

elif choice == '4':
    string = input("Enter the string: ")
    convert_to_upper(string)

elif choice == '5':
    print("Exiting the program.")
    break
```

```
    else:
        print("Invalid choice! Please enter a number between 1 and 5.")
# Run the main program
main()
```

OUTPUT:

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.9.py

Menu:
1. Check if a String is a Substring of Another String
2. Count Occurrences of a Character
3. Replace a Substring with Another Substring
4. Convert String to Uppercase
5. Exit
Enter your choice (1-5): 1
Enter the main string: khadeeja
Enter the substring to check: eeja
'eeja' is a substring of 'khadeeja'

Menu:
1. Check if a String is a Substring of Another String
2. Count Occurrences of a Character
3. Replace a Substring with Another Substring
4. Convert String to Uppercase
5. Exit
Enter your choice (1-5): 2
Enter the string: khadeeja
Enter the character to count: a
The character 'a' appears 2 times in the string.

Menu:
1. Check if a String is a Substring of Another String
2. Count Occurrences of a Character
3. Replace a Substring with Another Substring
4. Convert String to Uppercase
5. Exit
Enter your choice (1-5): 3
Enter the string: khadeeja
Enter the substring to replace: hadeeja
Enter the new substring: aija
After replacing 'hadeeja' with 'aija', the new string is: kaija
```

```
Menu:
1. Check if a String is a Substring of Another String
2. Count Occurrences of a Character
3. Replace a Substring with Another Substring
4. Convert String to Uppercase
5. Exit
Enter your choice (1-5): 4
Enter the string: khadeeja
The string in uppercase is: KHADEEJA

Menu:
1. Check if a String is a Substring of Another String
2. Count Occurrences of a Character
3. Replace a Substring with Another Substring
4. Convert String to Uppercase
5. Exit
Enter your choice (1-5): 5
Exiting the program.
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.10	Date :
Write a function to find the factorial of a number but also store the factorials calculated in a dictionary	

SOURCE CODE :

```
fac_dict ={}
def fac(n):
    if n==0 or n==1:
        fac_dict[n] = 1
        return 1
    if n in fac_dict:
        return fac_dict[n]
    result = n * fac(n-1)
    fac_dict[n] = result
    return result
number = int(input("Enter a num to calculate factorial"))
fact = fac(number)
print(f"The factorial of {number} is {fact}")
print("\nStored factorial in dict: ",fac_dict)
```

OUTPUT :

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.10.py
Enter a num to calculate factorial5
The factorial of 5 is 120

Stored factorial in dict:  {1: 1, 2: 2, 3: 6, 4: 24, 5: 120}
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.11	Date :
Perform various set operations **1.11.1 Set Union** **1.11.2 Set Intersection** **1.11.3 Set Difference**	

SOURCE CODE :

```
# Define two sets

set1 = {1, 2, 3, 4, 5}

set2 = {4, 5, 6, 7, 8}

# 1. Set Union

union_result = set1.union(set2)

print(f"Union of Set1 and Set2: {union_result}")

# 2. Set Intersection

intersection_result = set1.intersection(set2)

print(f"Intersection of Set1 and Set2: {intersection_result}")

# 3. Set Difference (set1 - set2)

difference_result = set1.difference(set2)

print(f"Difference of Set1 and Set2 (Set1 - Set2): {difference_result}")

# 3. Set Difference (set2 - set1)

difference_result_reverse = set2.difference(set1)

print(f"Difference of Set2 and Set1 (Set2 - Set1): {difference_result_reverse}")
```

OUTPUT :

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.11.py
Union of Set1 and Set2: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection of Set1 and Set2: {4, 5}
Difference of Set1 and Set2 (Set1 - Set2): {1, 2, 3}
Difference of Set2 and Set1 (Set2 - Set1): {8, 6, 7}
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.12	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 :

```
def student_with_highmarks():
    N = int(input("Enter the number of students:"))
    students={}
    for i in range(N):
        print(f"\nEnter details for student {i+1}:")
        name = input("Enter name: ")
        roll_no = input("Enter roll number: ")
        total_mark = int(input("Enter total marks: "))

        students[roll_no] = {
            "name": name,
            "roll_no": roll_no,
            "total_mark": total_mark
        }
    highest_mark_student = max(students.values(),key=lambda x:x["total_mark"])\
    print("\nDetails 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 Mark: {highest_mark_student['total_mark']}")
student_with_highmarks()
```

OUTPUT :

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.12.py
Enter the number of students:2

Enter details for student 1:
Enter name: khadeeja
Enter roll number: 27
Enter total marks: 97

Enter details for student 2:
Enter name: Faris
Enter roll number: 17
Enter total marks: 98

Details of the student with the highest total mark:
Name: Faris
Roll No: 17
Total Mark: 98
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.13	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 copied from {source_file} to {destination_file} successfully.")
    except FileNotFoundError:
        print(f"Error: {source_file} not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

source_file = input("Enter the source file name: ")
destination_file = input("Enter the destination file name: ")
copy_file(source_file, destination_file
```

OUTPUT

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.13.py
Enter the source file name: a.txt
Enter the destination file name: b.txt
Contents copied from a.txt to b.txt successfully.
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.14	Date :
Use the OS module to perform **1.14.1 Create a directory** **1.14.2 Directory Listing** **1.14.3 Search for ".py" files** **1.14.4 Remove a particular file**	

SOURCE CODE

```
import os

# Function to create a directory
def create_directory(directory_name):
    try:
        os.mkdir(directory_name)
        print(f"Directory '{directory_name}' created successfully.")
    except FileExistsError:
        print(f"Directory '{directory_name}' already exists.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Function to list contents of a directory
def list_directory(directory_path):
    try:
        files = os.listdir(directory_path)
        print(f"Contents of '{directory_path}':")
        for file in files:
            print(file)
    except FileNotFoundError:
        print(f"Directory '{directory_path}' not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Function to search for '.py' files in a directory
def search_py_files(directory_path):
    try:
        print(f"Searching for .py files in '{directory_path}':")
```

```
        for root, dirs, files in os.walk(directory_path):
            for file in files:
                if file.endswith('.py'):
                    print(os.path.join(root, file))
    except Exception as e:
        print(f"An error occurred: {e}")

# Function to remove a particular file
def remove_file(file_path):
    try:
        if os.path.isfile(file_path):
            os.remove(file_path)
            print(f"File '{file_path}' removed successfully.")
        else:
            print(f"File '{file_path}' does not exist.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Menu-driven program
def main():
    while True:
        print("\n--- OS Module Operations ---")
        print("1. Create a directory")
        print("2. Directory listing")
        print("3. Search for '.py' files")
        print("4. Remove a particular file")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '1':
            directory_name = input("Enter the directory name to create: ")
            create_directory(directory_name)
```

```
    elif choice == '2':
        directory_path = input("Enter the directory path to list: ")
        list_directory(directory_path)

    elif choice == '3':
        directory_path = input("Enter the directory path to search for '.py' files: ")
        search_py_files(directory_path)

    elif choice == '4':
        file_path = input("Enter the file path to remove: ")
        remove_file(file_path)

    elif choice == '5':
        print("Exiting the program.")
        break

    else:
        print("Invalid choice. Please try again.")

# Call the main function
if __name__ == "__main__":
    main()
```

OUTPUT

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.14.py

--- OS Module Operations ---
1. Create a directory
2. Directory listing
3. Search for '.py' files
4. Remove a particular file
5. Exit
Enter your choice (1-5): 1
Enter the directory name to create: khadeeja
Directory 'khadeeja' created successfully.

--- OS Module Operations ---
1. Create a directory
2. Directory listing
3. Search for '.py' files
4. Remove a particular file
5. Exit
```

```
Enter your choice (1-5): 2
Enter the directory path to list: C:\Users\hp\OneDrive\Desktop\Python Record
Contents of 'C:\Users\hp\OneDrive\Desktop\Python Record':
1.1.py
1.10.py
1.11.py
1.12.py
1.13.py
1.14.py
1.2.py
1.3.py
1.4_1.py
1.4_2.py
1.4_3.py
1.5.py
1.6.py
1.7.py
1.8.py
1.9.py
a.txt
b.txt
khadeeja

--- OS Module Operations ---
1. Create a directory
2. Directory listing
3. Search for '.py' files
4. Remove a particular file
5. Exit
```

```
Enter your choice (1-5): 3
Enter the directory path to search for '.py' files: C:\Users\hp\OneDrive\Desktop\Python Record
Searching for .py files in 'C:\Users\hp\OneDrive\Desktop\Python Record':
C:\Users\hp\OneDrive\Desktop\Python Record\1.1.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.10.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.11.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.12.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.13.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.14.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.2.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.3.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.4_1.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.4_2.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.4_3.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.5.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.6.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.7.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.8.py
C:\Users\hp\OneDrive\Desktop\Python Record\1.9.py

--- OS Module Operations ---
1. Create a directory
2. Directory listing
3. Search for '.py' files
4. Remove a particular file
5. Exit
Enter your choice (1-5): 4
Enter the file path to remove: b.txt
File 'b.txt' removed successfully.
```

```
--- OS Module Operations ---
1. Create a directory
2. Directory listing
3. Search for '.py' files
4. Remove a particular file
5. Exit
Enter your choice (1-5): 5
Exiting the program.
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

Program #1.15	Date :
Create a simple banking application by using inheritance	

SOURCE CODE

```
class BankAccount:
    def __init__(self, account_number, account_holder, balance=0.0):
        self.account_number = account_number
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ₹{amount:.2f}. New balance is ₹{self.balance:.2f}.")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"Withdrew ₹{amount:.2f}. New balance is ₹{self.balance:.2f}.")
            else:
                print("Insufficient funds.")
        else:
            print("Withdrawal amount must be positive.")

    def display_balance(self):
        print(f"Account Balance: ₹{self.balance:.2f}")

class SavingsAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, interest_rate=0.03):
        super().__init__(account_number, account_holder, balance)
        self.interest_rate = interest_rate
```

```
    def apply_interest(self):
        interest = self.balance * self.interest_rate
        self.deposit(interest)
        print(f"Applied interest: ₹{interest:.2f}.")

class CheckingAccount(BankAccount):
    def __init__(self, account_number, account_holder, balance=0.0, overdraft_limit=1000.0):
        super().__init__(account_number, account_holder, balance)
        self.overdraft_limit = overdraft_limit

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance + self.overdraft_limit:
                self.balance -= amount
                print(f"Withdrew ₹{amount:.2f}. New balance is ₹{self.balance:.2f}.")
            else:
                print("Exceeds overdraft limit.")
        else:
            print("Withdrawal amount must be positive.")

if __name__ == "__main__":
    savings = SavingsAccount("SA123456", "Alice", 1000.0)
    savings.deposit(500)
    savings.withdraw(200)
    savings.apply_interest()
    savings.display_balance()

    checking = CheckingAccount("CA123456", "Bob", 500.0)
    checking.deposit(300)
    checking.withdraw(700)
    checking.withdraw(2000)
```

```
checking.display_balance()
```

OUTPUT

```
PS C:\Users\hp\OneDrive\Desktop\Python Record> python 1.15.py
Deposited ₹500.00. New balance is ₹1500.00.
Withdrew ₹200.00. New balance is ₹1300.00.
Deposited ₹39.00. New balance is ₹1339.00.
Applied interest: ₹39.00.
Account Balance: ₹1339.00
Deposited ₹300.00. New balance is ₹800.00.
Withdrew ₹700.00. New balance is ₹100.00.
Exceeds overdraft limit.
Account Balance: ₹100.00
PS C:\Users\hp\OneDrive\Desktop\Python Record>
```

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 :

```
Connection to MySQL DB successful
Query executed successfully
Query executed successfully
(1, 'John Doe', 28, 'Male', 'USA')
(2, 'Jane Smith', 25, 'Female', 'Canada')
(3, 'John Doe', 28, 'Male', 'USA')
(4, 'Jane Smith', 25, 'Female', 'Canada')
PS C:\Python_Record>
```

	id	name	age	gender	nationality
Edit Copy Delete	1	John Doe	28	Male	USA
Edit Copy Delete	2	Jane Smith	25	Female	Canada
Edit Copy Delete	3	John Doe	28	Male	USA
Edit Copy Delete	4	Jane Smith	25	Female	Canada

Check all *With selected:* Edit Copy Delete Export

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 :

```
Connection to SQLite DB successful
Query executed successfully
Query executed successfully
Users in the database:
(1, 'John Doe', 28, 'Male', 'USA')
Query executed successfully
Query executed successfully
Users after deletion:
PS C:\Python_Record>
```

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 :

```
Connection to MySQL DB successful
Query executed successfully
Query executed successfully
Users in the database:
(2, 'Alice', 30, 'Female', 'USA')
Query executed successfully
Users after update:
(2, 'Alice', 35, 'Female', 'USA')
Query executed successfully
Users after deletion:
PS C:\Python_Record>
```

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 :

LOGIN

ABC-Name

Submit

Welcome, ABC-Name!

Your login was successful.

Back to Login

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

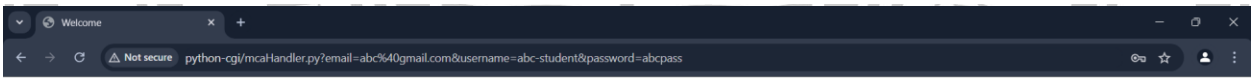
```
#!C:\wamp64\www\python-cgi\venv\Scripts\python.exe

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:

REGISTER - MCA admission 2024

abc@gmail.com
abc-student
•••••••
register



Welcome, abc-student!

Your Registration was successful! check verification sent on abc@gmail.com

Continue With registration

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:

CRUD OPERATIONS

Create

newuser

•••••••

Create

Read

newuser

Read

Update

newuser

updateduser

••••••••••••

Update

Delete

updateduser

Delete

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:

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

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:

```
original array: [[1 2 3 4]
 ['a' 'b' 4 7]
 [11 12 13 14]]
Filtered Array: [[ 1.  2.  3.  4.]
 [11. 12. 13. 14.]]
```

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:

```
Original array shape: (1, 3, 1)
Squeezed array shape: (3,)
Squeezed array:
 [1 2 3]
```

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:

```
Values to check: [3, 5, 10]
Presence in the array: [ True  True False]
```

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:

```
3D 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]]]

2D Diagonals:
Diagonal from slice 0:
[1 5 9]
Diagonal from slice 1:
[10 14 18]
Diagonal from slice 2:
[19 23 27]
```

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:

```
Original array:
 [[3 1 2]
 [6 4 5]]

Sorted along the first axis (axis=0):
 [[3 1 2]
 [6 4 5]]

Sorted along the last axis (axis=1):
 [[1 2 3]
 [4 5 6]]

Sorted flattened array:
 [1 2 3 4 5 6]
```

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:

```
Original structured array:
 [('Alice', 5.5, 2) ('Bob', 6. , 1) ('Charlie', 5.7, 2) ('David', 5.9, 1)
 ('Eve', 5.4, 2)]

Sorted structured array:
 [('David', 5.9, 1) ('Bob', 6. , 1) ('Eve', 5.4, 2) ('Alice', 5.5, 2)
 ('Charlie', 5.7, 2)]
```

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:

```
Original complex array:
 [3.+2.j 1.+4.j 2.+3.j 1.+2.j 2.+1.j]

Sorted complex array:
 [1.+2.j 1.+4.j 2.+1.j 2.+3.j 3.+2.j]
```

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:

```
Original array:
 [[3 1 2]
 [6 4 5]
 [1 7 8]
 [9 0 3]]

Array sorted by column 1:
 [[9 0 3]
 [3 1 2]
 [6 4 5]
 [1 7 8]]
```

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:

```
Original array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Sum of diagonal elements: 15
```

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

```
Matrix A:
 [[1 2 3]
 [4 5 6]]

Matrix B:
 [[ 7  8]
 [ 9 10]
 [11 12]]

Matrix Multiplication using np.dot():
 [[ 58  64]
 [139 154]]

Matrix Multiplication using @ operator:
 [[ 58  64]
 [139 154]]
```

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:

```
Matrix A:
 [[1.+2.j 2.+3.j]
 [3.+4.j 4.+5.j]]

Matrix B:
 [[5.+6.j 6.+7.j]
 [7.+8.j 8.+9.j]]

Matrix Multiplication Result:
 [[-17. +53.j -19. +61.j]
 [-21.+105.j -23.+121.j]]
```

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:

```
Inner Product of vector_a and vector_b: 32

Outer Product of vector_a and vector_b:
 [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]

Cross Product of vector_a and vector_b: [-3  6 -3]

Inner Product (Dot Product) of matrix_a and matrix_b:
 [[19 22]
 [43 50]]

Outer Product of the first columns of matrix_a and matrix_b:
 [[ 5  7]
 [15 21]]
```

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:

```
Data 1:
 [1 2 3 4 5]
Data 2:
 [5 4 3 2 1]

Covariance Matrix:
 [[ 2.5 -2.5]
 [-2.5  2.5]]
```

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:

```
Covariance Matrix:
 [[4 2]
 [2 3]]

Correlation Matrix:
 [[1.         0.57735027]
 [0.57735027 1.        ]]
```

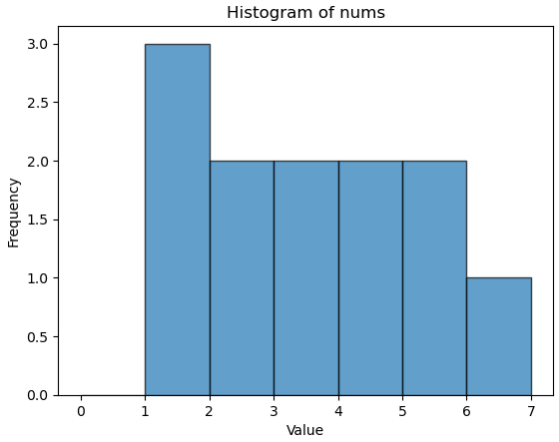
Program #4.1.16	Date :
Write a NumPy program to compute the histogram of nums against the bins.	

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]
```



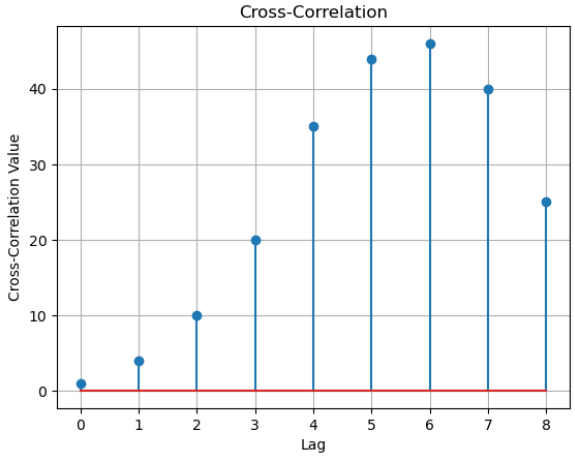
Program #4.1.17	Date :
Write a NumPy program to compute the cross-correlation of two given arrays.	

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:

```
Original Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Mean along the second axis: [2. 5. 8.]
Standard Deviation along the second axis: [0.81649658 0.81649658 0.81649658]
Variance along the second axis: [0.66666667 0.66666667 0.66666667]
```

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:

```
Original Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

80th Percentile along the second axis: [2.6 5.6 8.6]
```

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:

```
Addition of two Series:
0    15
1    30
2    45
3    60
4    75
dtype: int64

Subtraction of two Series:
0     5
1    10
2    15
3    20
4    25
dtype: int64

Multiplication of two Series:
0      50
1     200
2     450
3     800
4    1250
dtype: int64

Division of two Series:
0    2.0
1    2.0
2    2.0
3    2.0
4    2.0
dtype: float64
```

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:

```
Dictionary: {'a': 10, 'b': 12, 'c': 30}
Series: a    10
b    12
c    30
dtype: int64
```

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:

```
0    10
1    20
2    30
3    40
Name: Column1, dtype: int64
```

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:

```
0    1
1    2
2    3
3    4
4    5
5    6
6    7
7    8
dtype: object
```

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:

```
5    6
6    7
7    8
dtype: int64
```

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:

```
0    1
1    2
2    3
2    6
3    7
4    8
dtype: int64
```

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:

```
4    3
2    2
5    2
1    1
3    1
Name: count, dtype: int64
```

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:

```
0     apple
1    banana
2     grape
3      kiwi
4      pear
5     mango
dtype: object
```

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:

```
index of minimum value 3
index of max value 2
```

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:

```
      name  age
0  varghese   21
1      kuku   22
2      aami   23
```

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:

```
no of rows: 3
no of columns 3
```

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:

```
original data frame      name  rollno  age
0  varghese      23   22
1      kuku      12   21
2      aami      45   20
DataFtame after adding new row:       name  rollno  age
0  varghese      23   22
1      kuku      12   21
2      aami      45   20
3       roy      62   24
```

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:

```
DataFrame has been written to 'output_file.csv' using a tab separator.
```

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:

```
Original DataFrame with NaN values:
      name  score  age
0     John   85.0   20
1    Emily    NaN   21
2  Michael   78.0   19
3     Sara    NaN   22
4    David   95.0   20

DataFrame after replacing NaN values with zeros in 'score' column:
      name  score  age
0     John   85.0   20
1    Emily    0.0   21
2  Michael   78.0   19
3     Sara    0.0   22
4    David   95.0   20

DataFrame after dropping rows with index 1 and 3:
      name  score  age
0     John   85.0   20
2  Michael   78.0   19
4    David   95.0   20
```

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:

```
Original DataFrame:
      name  score  age
0     John     85   20
1    Emily     92   21
2  Michael     78   19
3     Sara     88   22
4    David     95   20

Shuffled DataFrame:
      name  score  age
0     John     85   20
1  Michael     78   19
2     Sara     88   22
3    Emily     92   21
4    David     95   20
```

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:

```
Original DataFrame:
      name  score  age
0     John     85   20
1    Emily     92   21
2  Michael     78   19
3     Sara     88   22
4    David     95   20

Row where the value of 'score' is maximum:
name     David
score       95
age         20
Name: 4, dtype: object
```

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:

```
Original DataFrame:
      name  score  age
0     John     85   20
1    Emily     92   21
2  Michael     78   19
3     Sara     88   22
4    David     95   20

The column 'score' is present in the DataFrame.
```

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:

```
Empty DataFrame:
Empty DataFrame
Columns: [name, score, age]
Index: []

DataFrame after appending data:
      name score age
0     John    85  20
1    Emily    92  21
2  Michael    78  19
3     Sara    88  22
4    David    95  20
```

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:

```
Original DataFrame:
             Name  Age
0  Alberto Franco   18
1    Gino Mcneill   22
2     Ryan Parkes   40
3    Eesha Hinton   50
4    Syed Wharton   80

DataFrame with Age Categories:
             Name  Age Age Category
0  Alberto Franco   18  Young Adult
1    Gino Mcneill   22  Young Adult
2     Ryan Parkes   40        Adult
3    Eesha Hinton   50       Senior
4    Syed Wharton   80          NaN
```

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:

```
DataFrame with Random Values:
   Random Numbers    Floats
0              52  0.866176
1              93  0.601115
2              15  0.708073
3              72  0.020584
4              61  0.969910
5              21  0.832443
6              83  0.212339
7              87  0.181825
8              75  0.183405
9              75  0.304242

DataFrame with Missing Values:
     A      B
0  1.0    NaN
1  2.0    two
2  NaN  three
3  4.0    NaN
4  5.0   five

DataFrame with Datetime Values:
                        Date     Value
0 2024-09-15 10:41:25.355991  0.524756
1 2024-09-16 10:41:25.355991  0.431945
2 2024-09-17 10:41:25.355991  0.291229
3 2024-09-18 10:41:25.355991  0.611853
4 2024-09-19 10:41:25.355991  0.139494
5 2024-09-20 10:41:25.355991  0.292145
6 2024-09-21 10:41:25.355991  0.366362
7 2024-09-22 10:41:25.355991  0.456070
8 2024-09-23 10:41:25.355991  0.785176
9 2024-09-24 10:41:25.355991  0.199674
```

```
DataFrame with Mixed Values:
   Integers Strings  Booleans
0         1     one      True
1         2     two     False
2         3   three      True
3         4    four     False
4         5    five      True
```

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:

```
Student Data 1:
  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 Data 2:
  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

Combined Student Data:
  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
5         s4  Scarlette Fisher    201
6         s5  Carla Williamson    200
7         s6       Dante Morse    198
8         s7    Kaiser William    219
9         s8   Madeeha Preston    201
```

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:

```
Student Data 1:
  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 Data 2:
  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

Combined Student Data (along columns):
  student_id              name  marks student_id              name  marks
0         s1  Danniella Fenton    200         s4  Scarlette Fisher    201
1         s2      Ryder Storey    210         s5  Carla Williamson    200
2         s3      Bryce Jensen    190         s6       Dante Morse    198
3         s4         Ed Bernal    222         s7    Kaiser William    219
4         s5       Kwame Morin    199         s8   Madeeha Preston    201
```

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

```
Combined Student Data:
  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
5         s4  Scarlette Fisher    201
6         s5  Carla Williamson    200
7         s6       Dante Morse    198
8         s7    Kaiser William    219
9         s8   Madeeha Preston    201

Merged Data (on common column 'student_id'):
  student_id              name  marks  exam_id
0         s1  Danniella Fenton    200       23
1         s2      Ryder Storey    210       45
2         s3      Bryce Jensen    190       12
3         s4         Ed Bernal    222       67
4         s5       Kwame Morin    199       21
5         s4  Scarlette Fisher    201       67
6         s5  Carla Williamson    200       21
7         s7    Kaiser William    219       55
8         s8   Madeeha Preston    201       33
```

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:

```
Combined Student Data:
  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
5         s4  Scarlette Fisher    201
6         s5  Carla Williamson    200
7         s6       Dante Morse    198
8         s7    Kaiser William    219
9         s8   Madeeha Preston    201

Merged Data (matching records from both sides):
  student_id              name  marks  exam_id
0         s1  Danniella Fenton    200       23
1         s2      Ryder Storey    210       45
2         s3      Bryce Jensen    190       12
3         s4         Ed Bernal    222       67
4         s5       Kwame Morin    199       21
5         s4  Scarlette Fisher    201       67
6         s5  Carla Williamson    200       21
7         s7    Kaiser William    219       55
8         s8   Madeeha Preston    201       33
```

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:

```
Group for school code s001:
  school class           name date_Of_Birth  age  height  weight  address
0   s001     V  Alberto Franco    15/05/2002   12     173      35  street1
3   s001    VI    Eesha Hinton    25/09/1998   13     167      30  street1

Group for school code s002:
  school class          name date_Of_Birth  age  height  weight  address
1   s002     V  Gino Mcneill    17/05/2002   12     192      32  street2
4   s002     V  Gino Mcneill    11/05/2002   14     151      31  street2

Group for school code s003:
  school class         name date_Of_Birth  age  height  weight  address
2   s003    VI  Ryan Parkes    16/02/1999   13     186      33  street3

Group for school code s004:
  school class          name date_Of_Birth  age  height  weight  address
5   s004    VI  David Parkes    15/09/1997   12     159      32  street4

Type of GroupBy object: <class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

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:

school	mean	min	max
s001	12.5	12	13
s002	13.0	12	14
s003	13.0	13	13
s004	12.0	12	12

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:

```
   Id  type     book  count
0   1    10     Math      1
1   1    11  Physics      1
2   1    12  Physics      1
3   1    20     Math      1
4   2    14  English      1
5   2    15  English      1
6   2    21  English      1
```

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:

	school	mean_age	min_age	max_age
0	s001	12.5	12	13
1	s002	13.0	12	14
2	s003	13.0	13	13
3	s004	12.0	12	12

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.

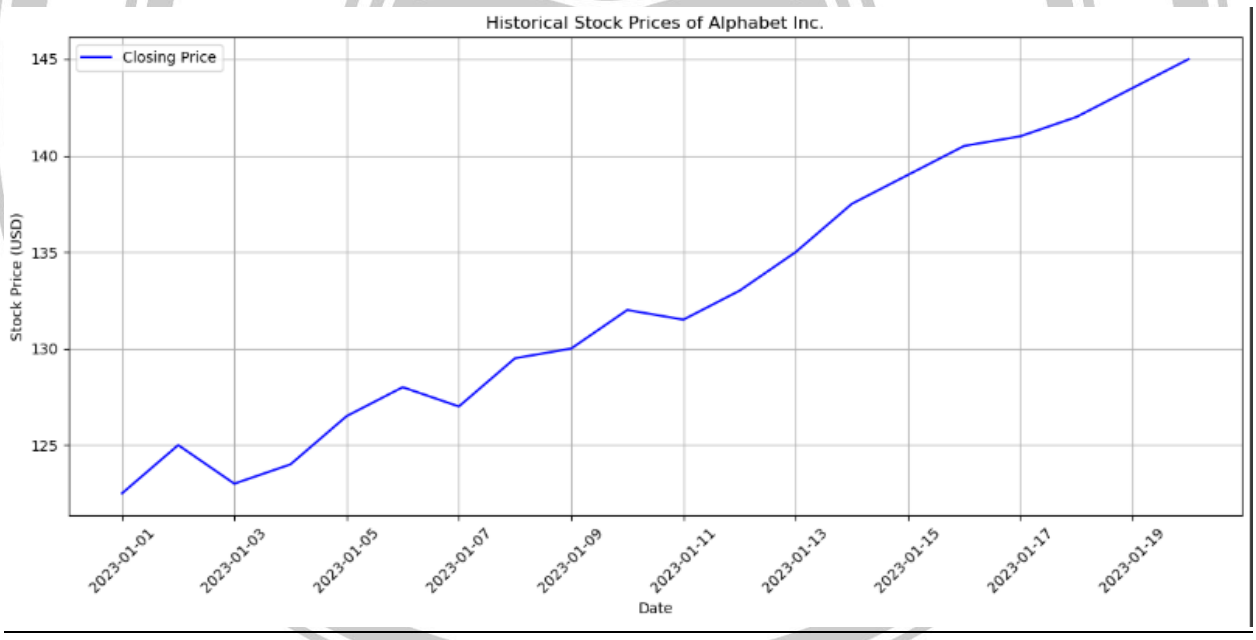
4.4.1.4 Create a scatter plot of the trading volume/stock prices of Alphabet Inc. stock between two specific dates.

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:

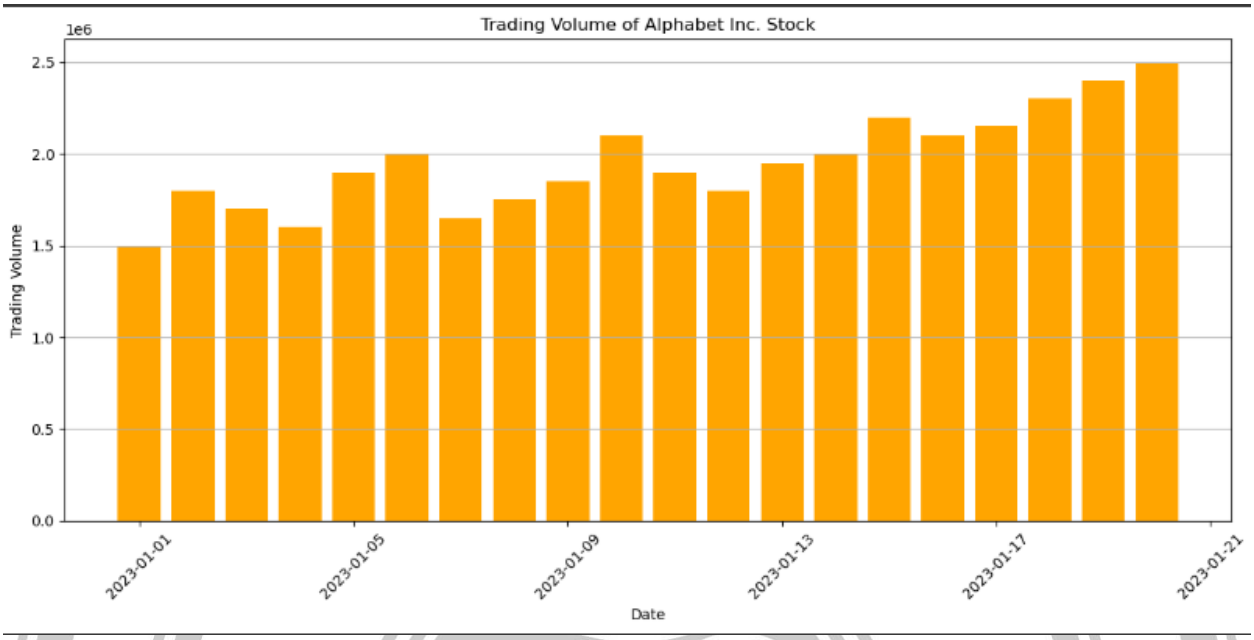


SOURCE CODE 1.2:

```
# 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:

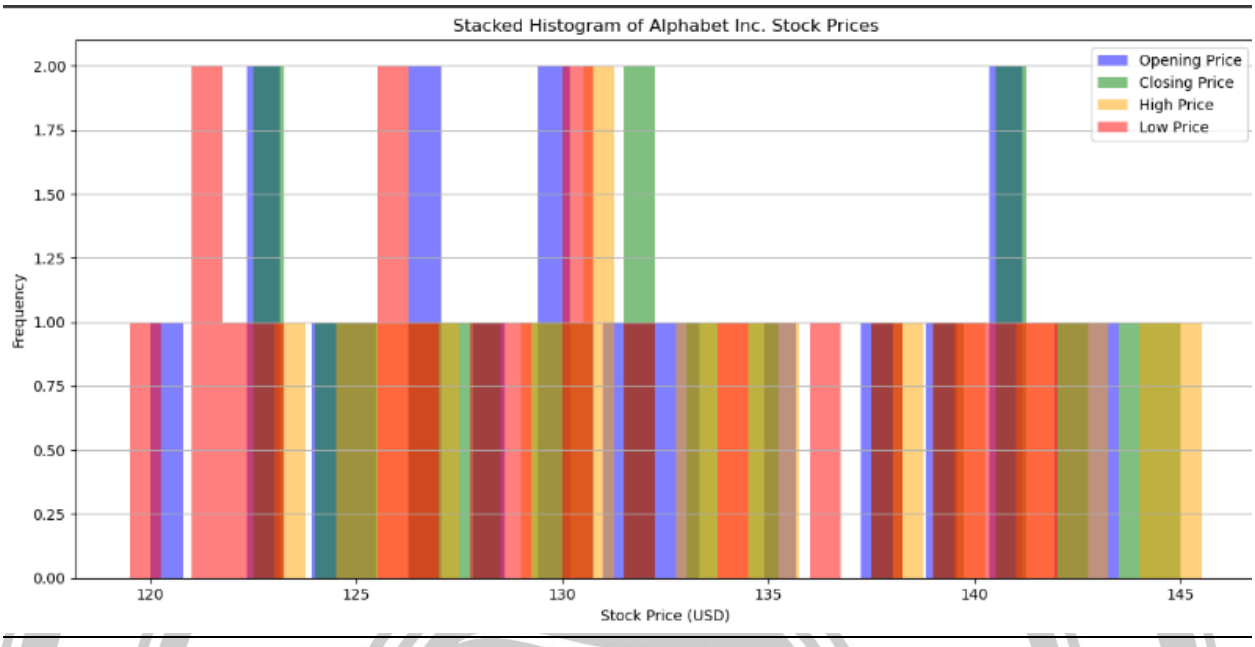


SOURCE CODE 1.3:

```
# 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:

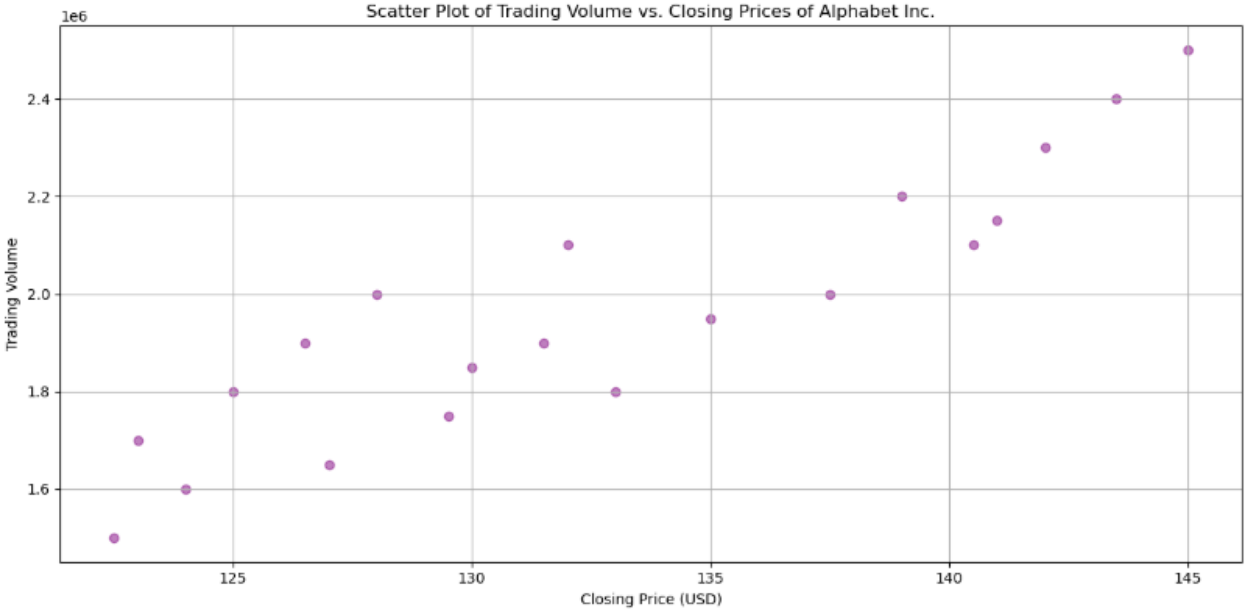


SOURCE CODE 1.4:

```
# 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

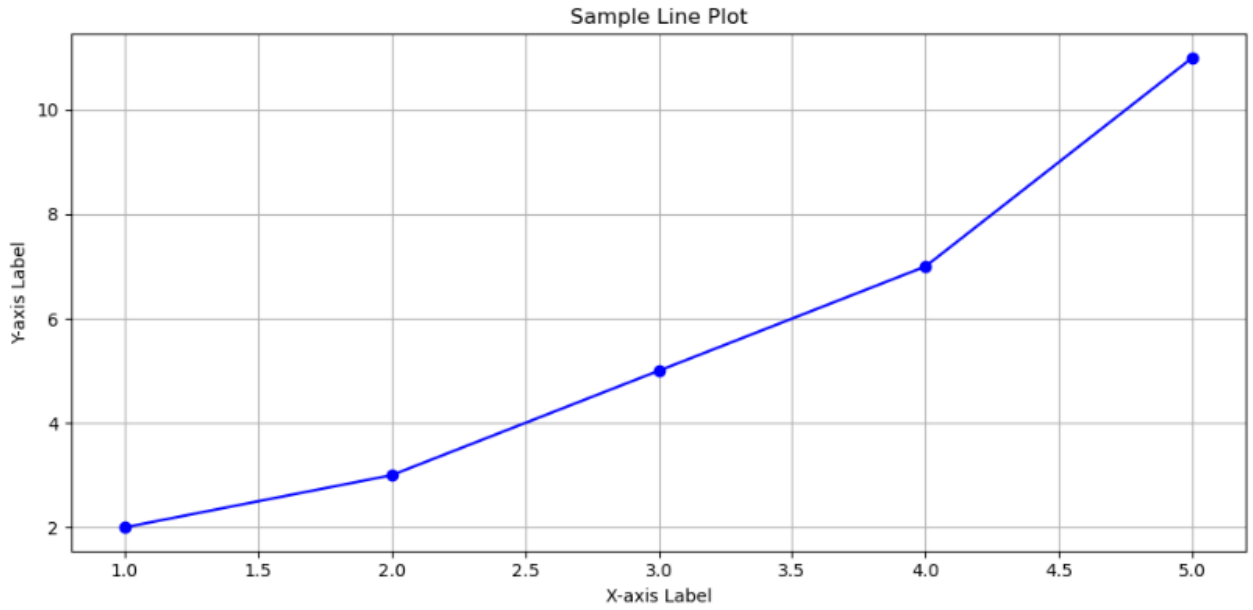


Program #4.4.2	Date :
Write a Python program to draw a line with a suitable label on the x-axis, y-axis, and title.	

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:


Sample Line Plot
Y-axis Label
X-axis Label
10
8
6
4
2
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

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.

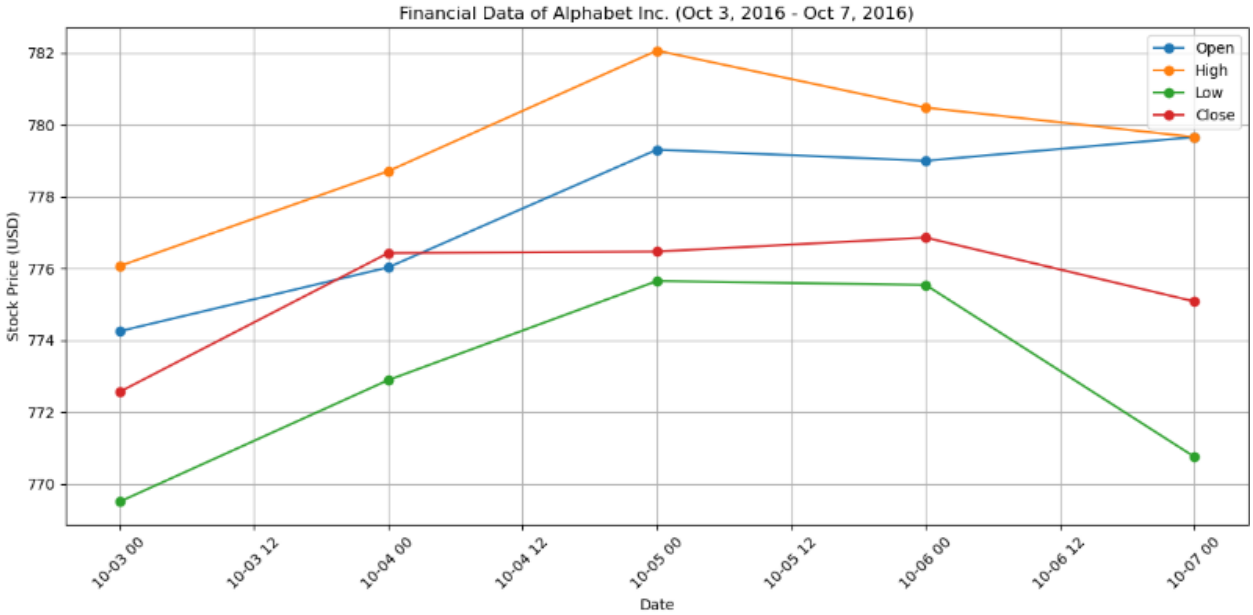
Date,Open,High,Low,Close
10-03-16,774.25,776.065002,769.5,772.559998
10-04-16,776.030029,778.710022,772.890015,776.429993
10-05-16,779.309998,782.070007,775.650024,776.469971
10-06-16,779,780.47998,775.539978,776.859985
10-07-16,779.659973,779.659973,770.75,775.080017

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)
```

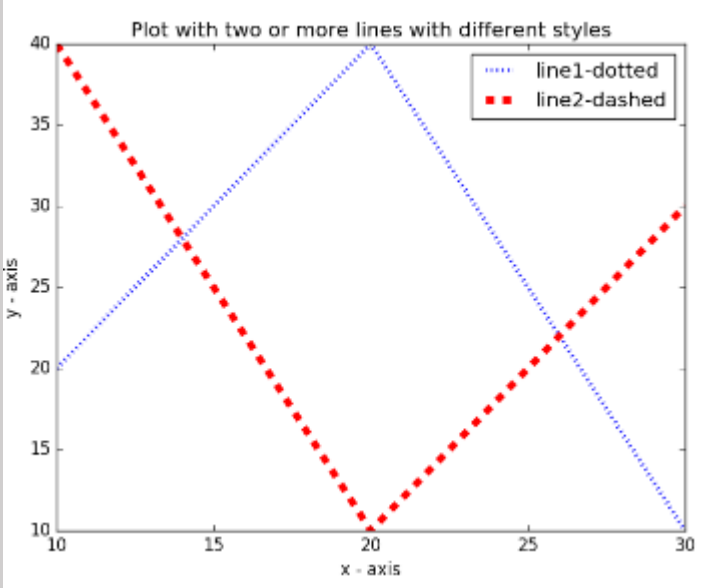
```
# Plotting the financial data
plt.figure(figsize=(12, 6))

# 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:


Financial Data of Alphabet Inc. (Oct 3, 2016 - Oct 7, 2016)
Open
High
Low
Close
Stock Price (USD)
782
780
778
776
774
772
770
10-03 00
10-03 12
10-04 00
10-04 12
10-05 00
10-05 12
10-06 00
10-06 12
10-07 00
Date

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:

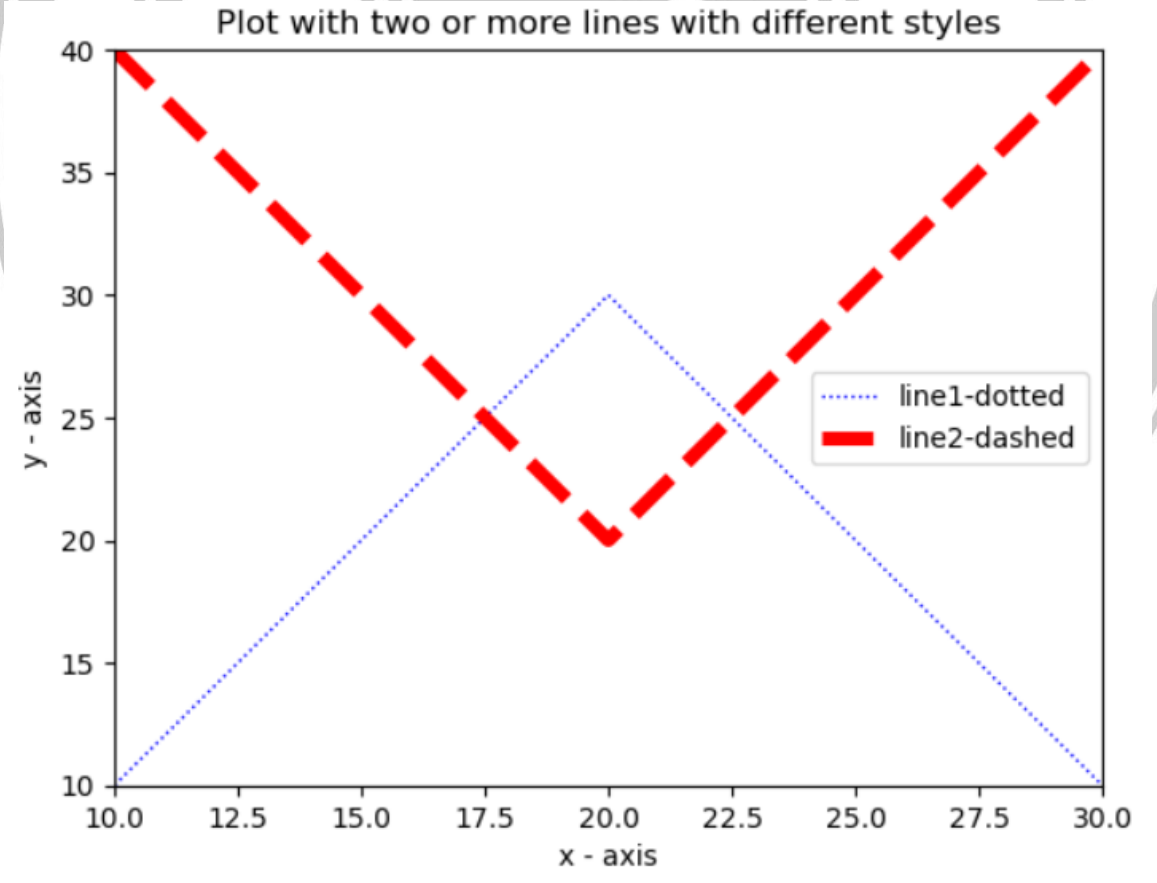


SOURCE CODE:

```
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



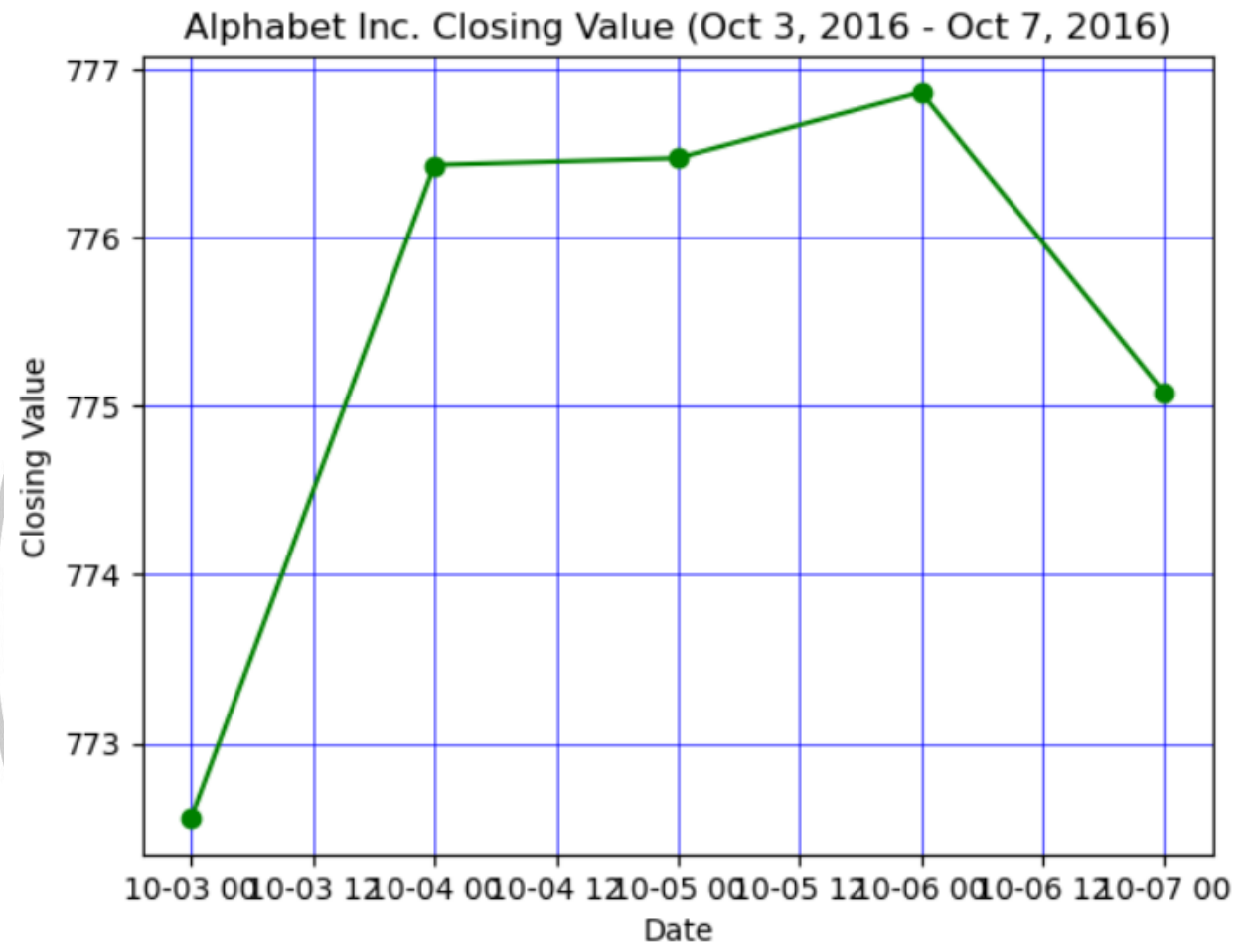
Program #4.4.5	Date :
Write a Python program to display the grid and draw line charts of the closing value of Alphabet Inc. between October 3, 2016, and October 7, 2016. Customized the grid lines with linestyle -, width 0.5, and color blue. **Date,Close** **03-10-16,772.559998** **04-10-16,776.429993**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

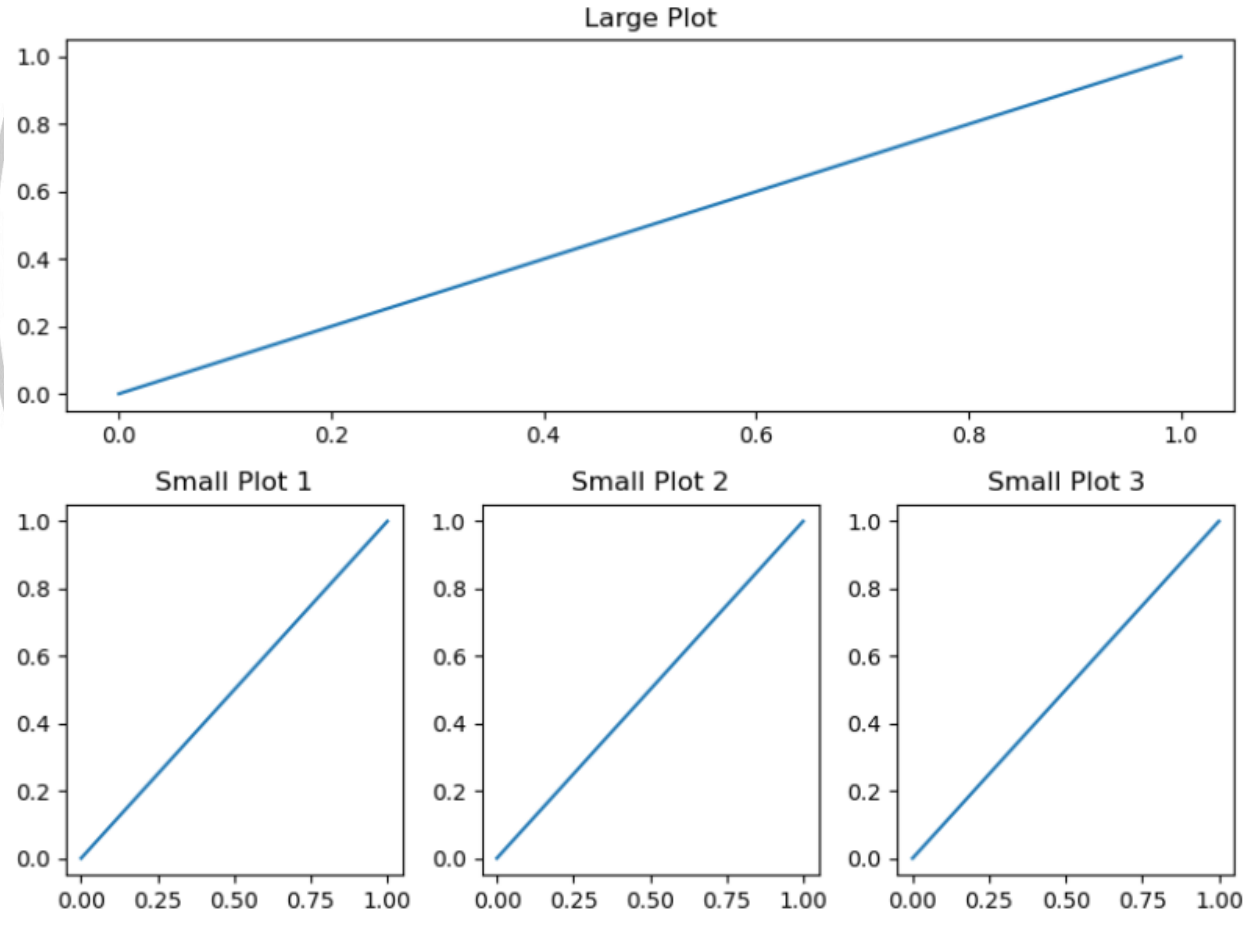
Write a Python program to create multiple plots as in the screenshot (use any method).

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')
```

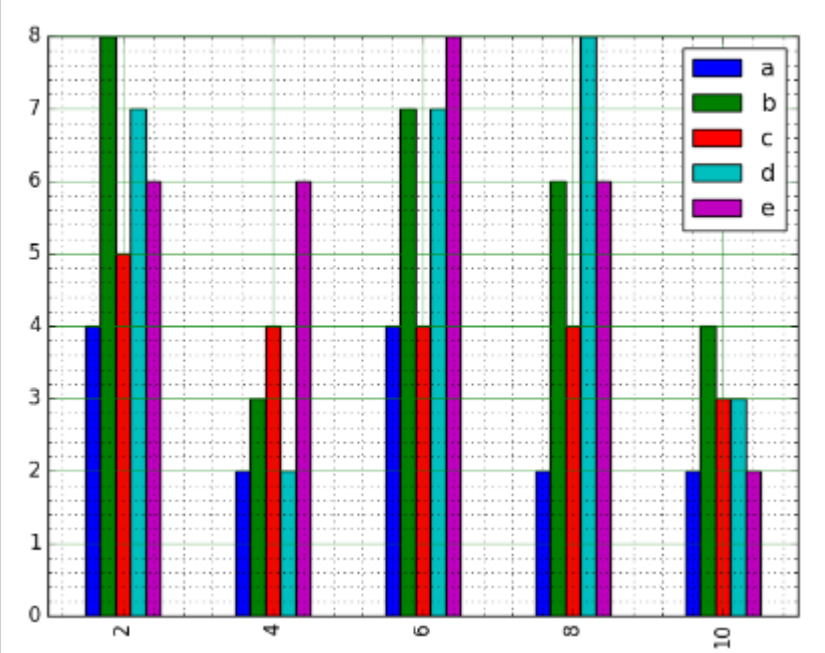
```
ax3.plot([0, 1], [0, 1])
ax3.set_title('Small Plot 2')
ax4.plot([0, 1], [0, 1])
ax4.set_title('Small Plot 3')
# Adjust layout
plt.tight_layout()
# Show plot
plt.show()
```

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:

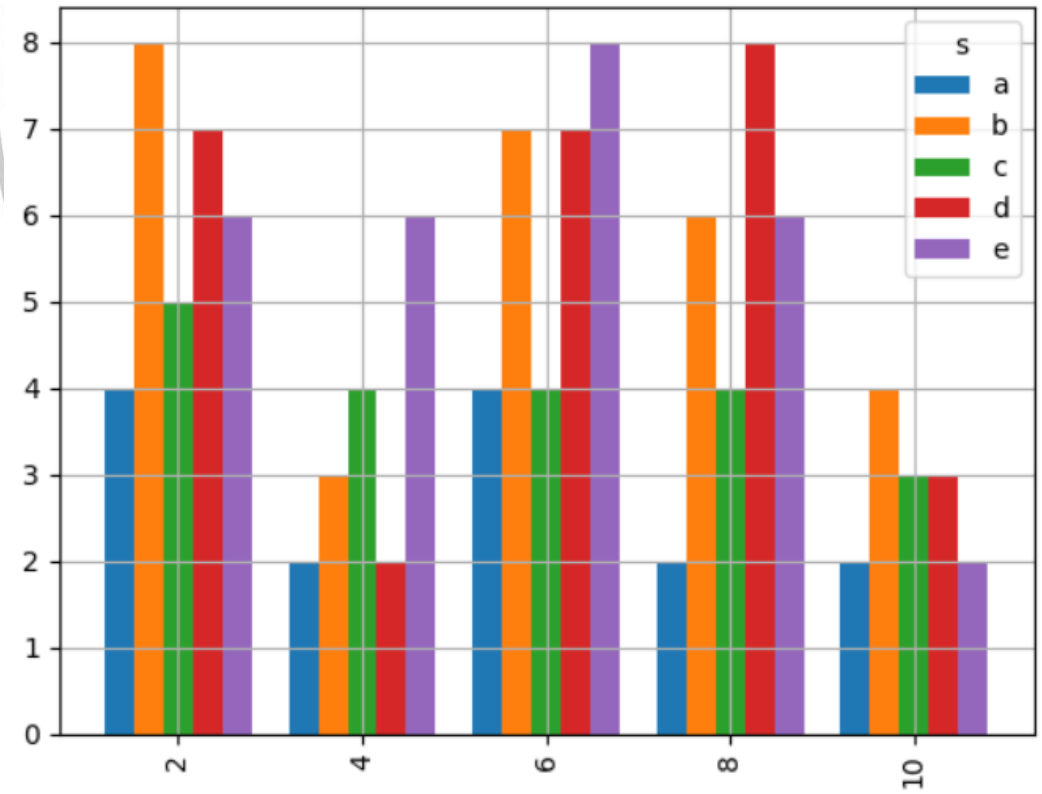


SOURCE CODE:

```
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],
```

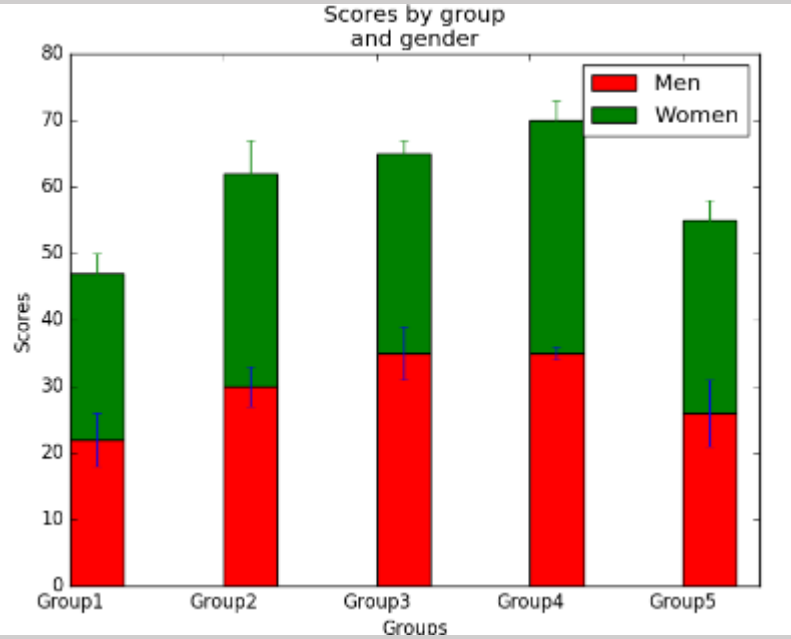
```
    'd': [7, 2, 7, 8, 3],
    'e': [6, 6, 8, 6, 2]
}
# Create DataFrame
index = [2, 4, 6, 8, 10]
df = pd.DataFrame(data, index=index)
# Plotting the bar plot
df.plot(kind='bar', width=0.8)
# Adding legend, labels, and grid
plt.legend(title="s", loc='best')
plt.grid(True)
plt.show()
```

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:

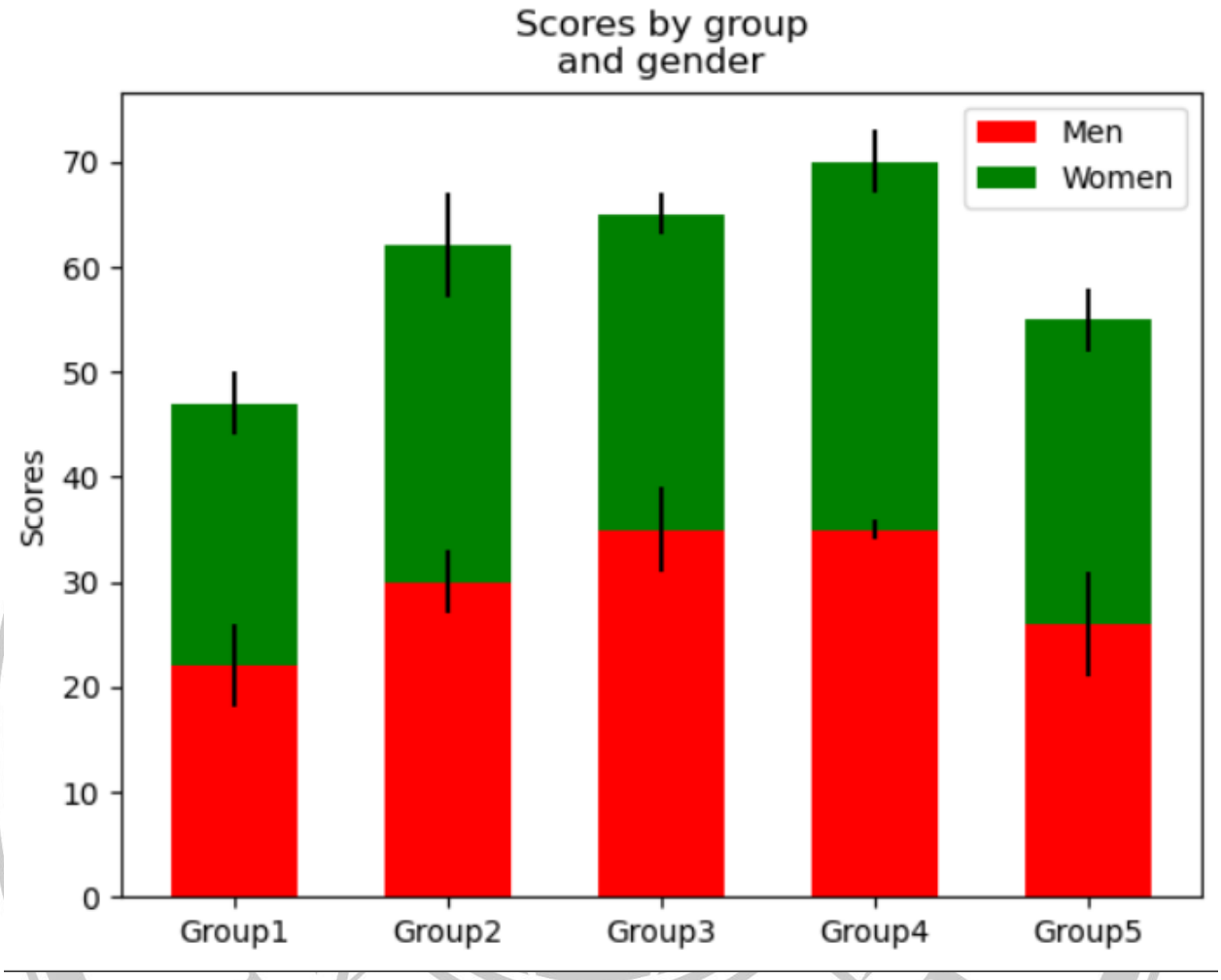


SOURCE CODE:

```
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)
```

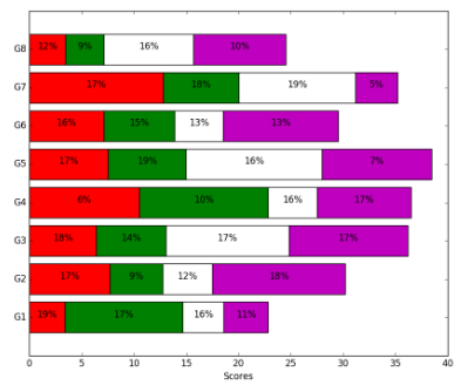
```
# Group labels
groups = ('Group1', 'Group2', 'Group3', 'Group4', 'Group5')

# 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:


Scores by group
and gender
Men
Women
Scores
0
10
20
30
40
50
60
70
Group1
Group2
Group3
Group4
Group5

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:



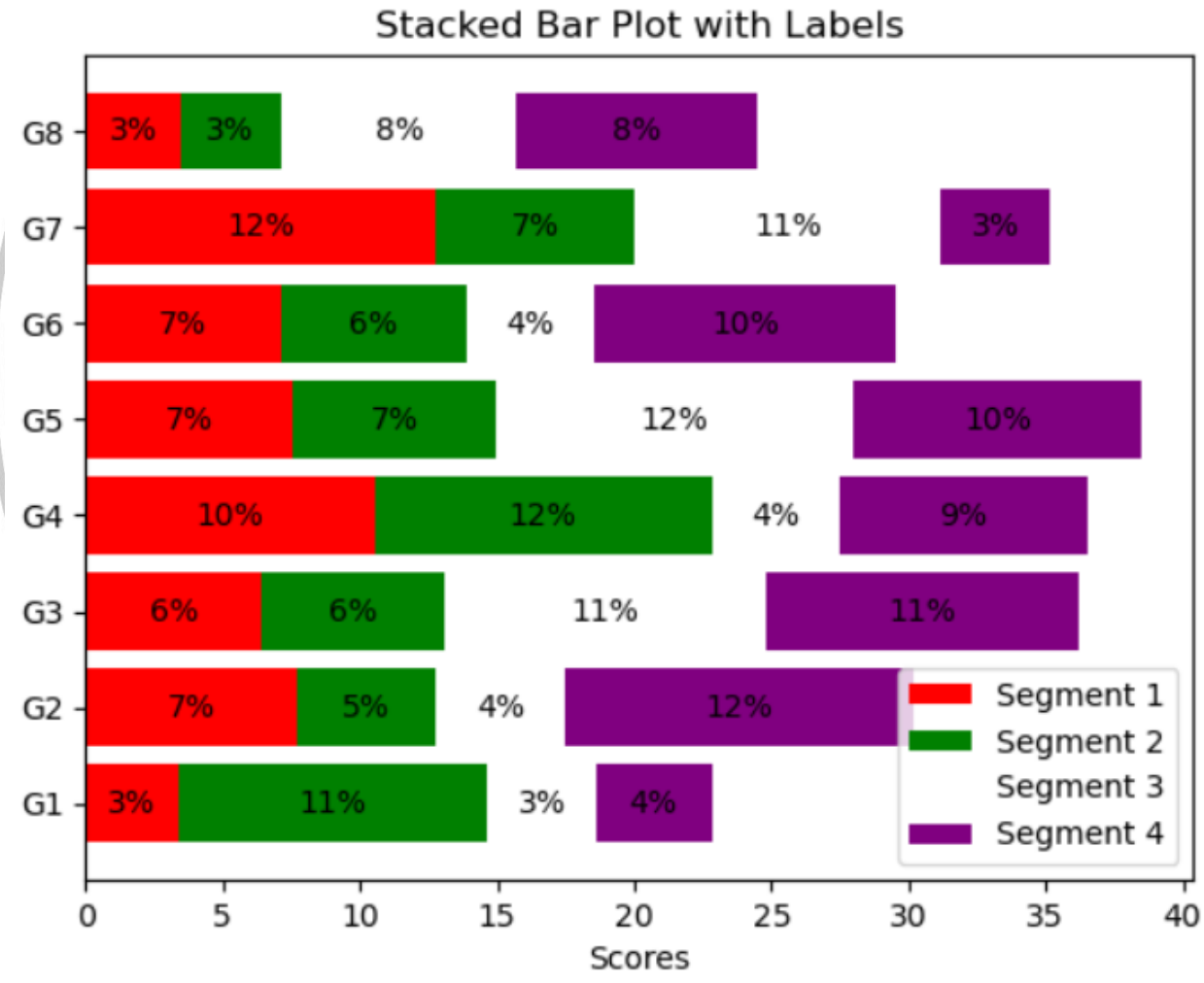
SOURCE CODE:

```
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')
```

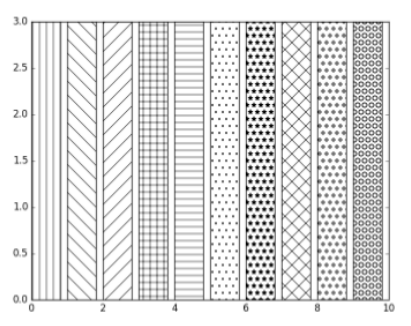
```
# Display the legend
ax.legend()

# 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:

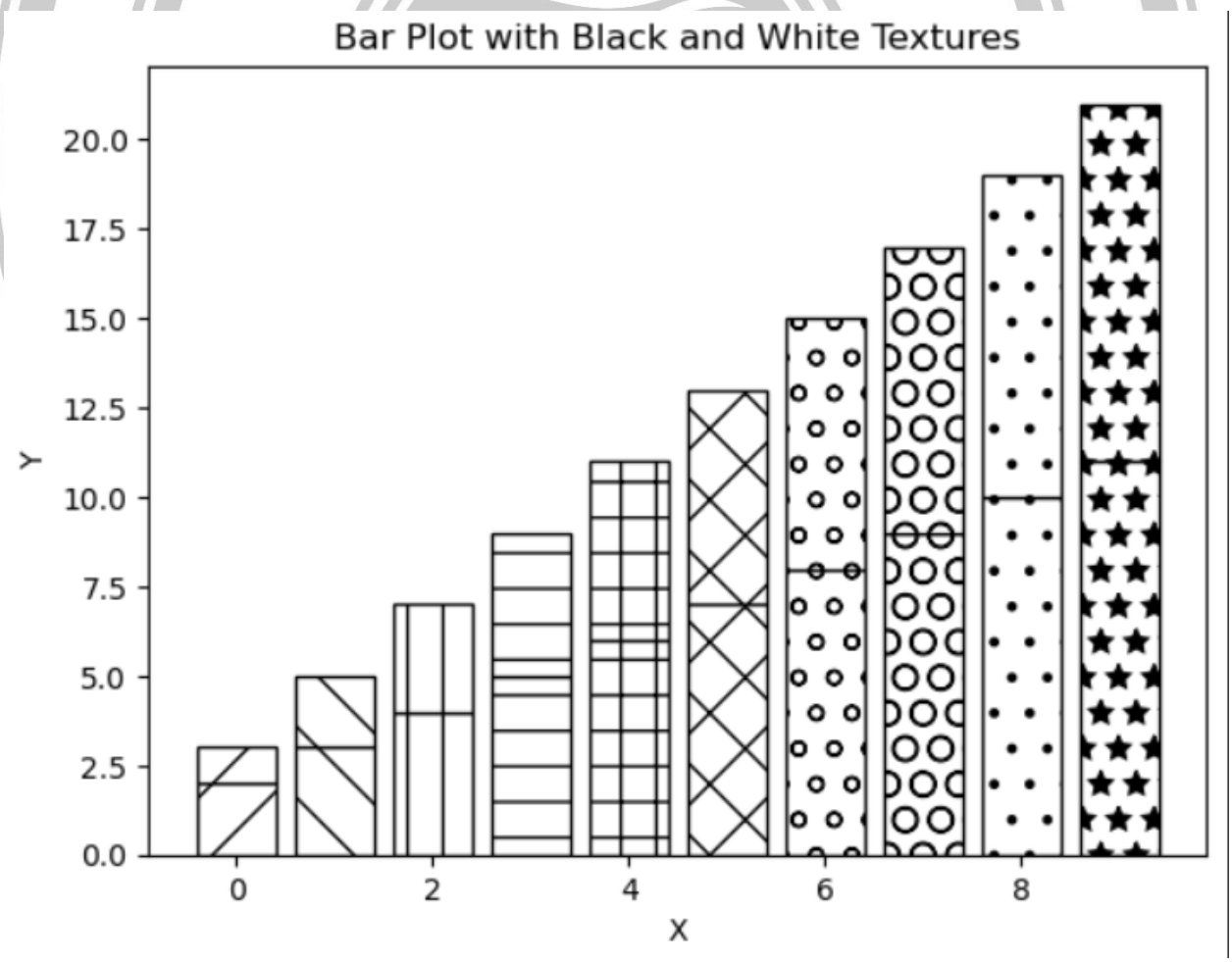


SOURCE CODE:

```
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 :
5.1. Handle the given dataset (Data.csv) with adequate preprocessing steps mentioned and visualize the dataset with appropriate graphs. **5.1.1 Handle Missing Data Values** **5.1.2 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'])
```

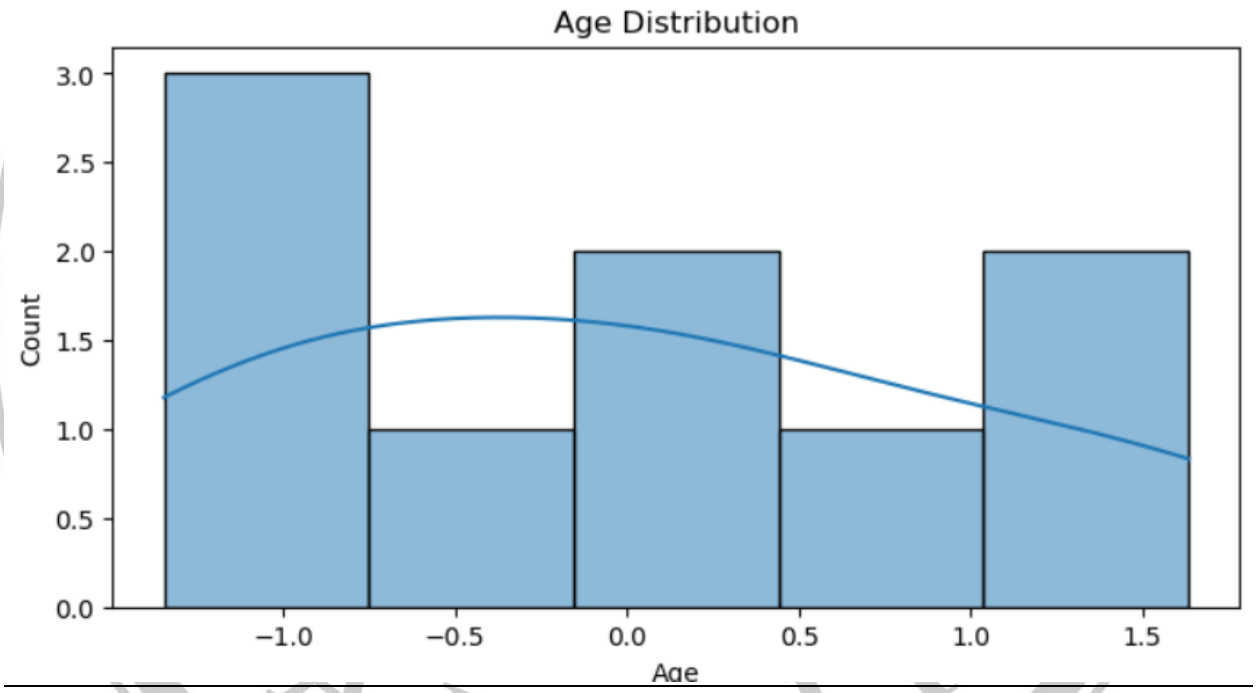
```
# Encode 'Country' using OneHotEncoding (creates separate columns for each category)
data = pd.get_dummies(data, columns=['Country'], drop_first=True)  # drop_first to avoid dummy variable trap

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)
```

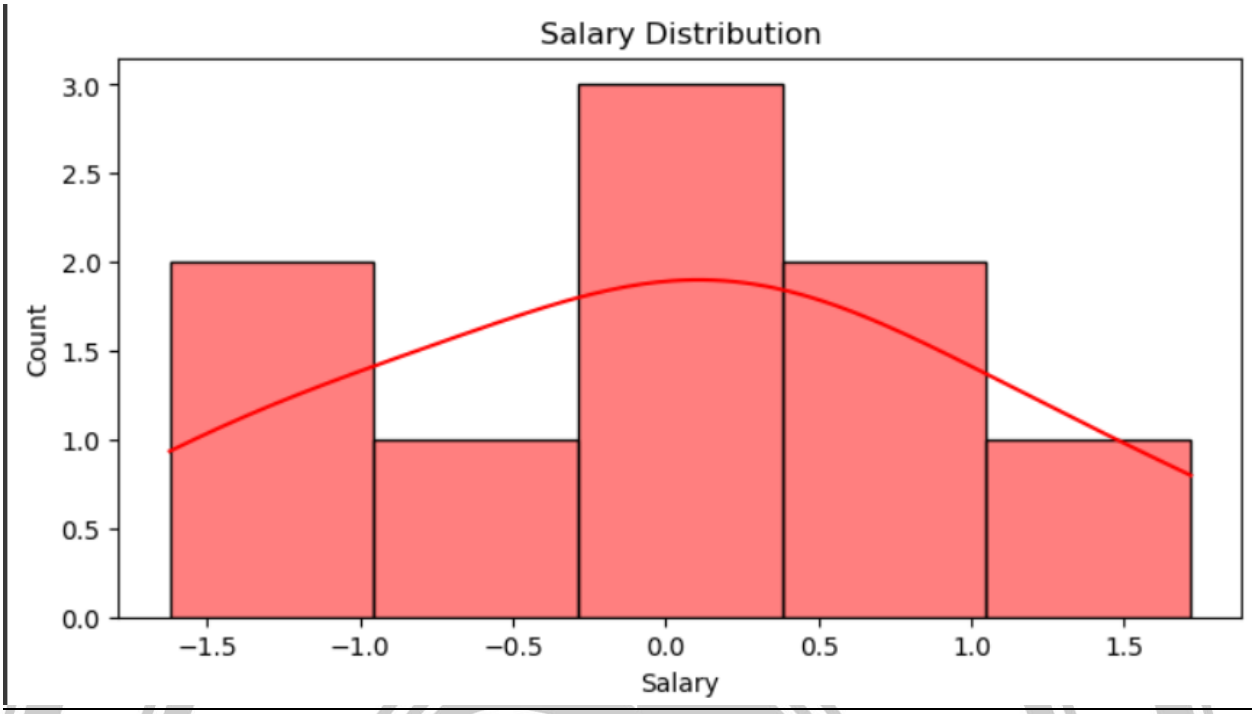
```
plt.figure(figsize=(6, 4))
sns.countplot(x='Purchased', data=data)
plt.title('Count of Purchases (Yes/No)')
plt.show()

# 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()
```

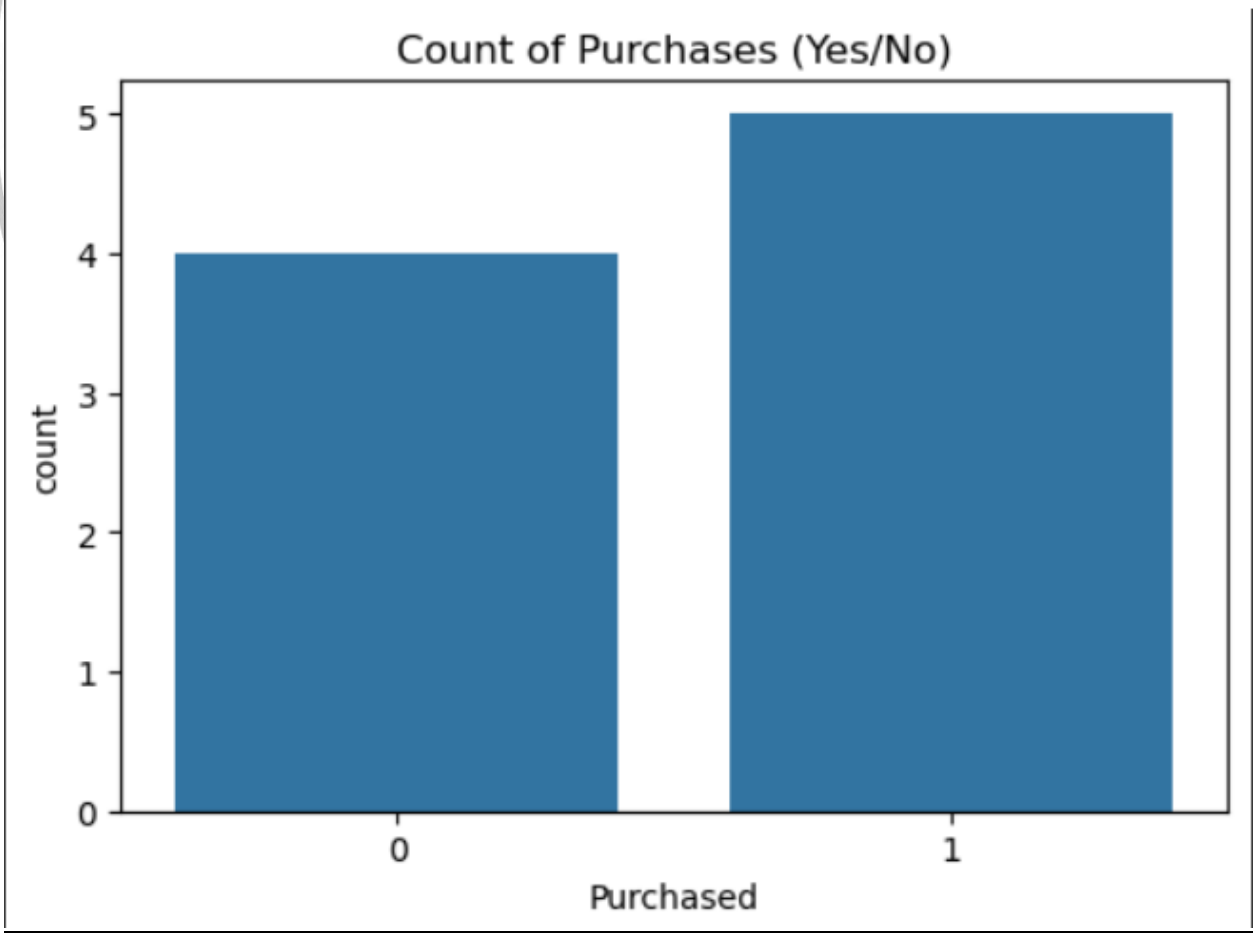
OUTPUT:

```
Original Dataset:
      Name   Age   Salary Country Purchased
0    John  28.0  50000.0     USA       Yes
1    Emma  32.0      NaN  Canada        No
2   Lucas  41.0  62000.0     USA       Yes
3  Sophia  29.0  58000.0  France        No
4  Olivia   NaN  54000.0     USA       Yes
5    Liam  35.0  52000.0  France        No
6     Ava  30.0  60000.0  Canada       Yes
7    Noah  40.0  65000.0  France        No
8     Mia  36.0      NaN     USA       Yes

Dataset after handling missing values:
      Name     Age        Salary Country Purchased
0    John  28.000  50000.000000     USA       Yes
1    Emma  32.000  57285.714286  Canada        No
2   Lucas  41.000  62000.000000     USA       Yes
3  Sophia  29.000  58000.000000  France        No
4  Olivia  33.875  54000.000000     USA       Yes
5    Liam  35.000  52000.000000  France        No
6     Ava  30.000  60000.000000  Canada       Yes
7    Noah  40.000  65000.000000  France        No
8     Mia  36.000  57285.714286     USA       Yes

Dataset after encoding categorical variables:
      Name     Age        Salary  Purchased  Country_France  Country_USA
0    John  28.000  50000.000000          1           False         True
1    Emma  32.000  57285.714286          0           False        False
2   Lucas  41.000  62000.000000          1           False         True
3  Sophia  29.000  58000.000000          0            True        False
4  Olivia  33.875  54000.000000          1           False         True
5    Liam  35.000  52000.000000          0            True        False
6     Ava  30.000  60000.000000          1           False        False
7    Noah  40.000  65000.000000          0            True        False
```

```
8     Mia  36.000  57285.714286          1           False         True

Dataset after scaling features:
      Name       Age        Salary  Purchased  Country_France  Country_USA
0     John -1.348310 -1.622709e+00          1           False         True
1     Emma -0.430312 -1.620536e-15          0           False        False
2    Lucas  1.635185  1.049988e+00          1           False         True
3   Sophia -1.118811  1.590891e-01          0            True        False
4   Olivia  0.000000 -7.318098e-01          1           False         True
5     Liam  0.258187 -1.177259e+00          0            True        False
6      Ava -0.889311  6.045385e-01          1           False        False
7     Noah  1.405685  1.718162e+00          0            True        False
8      Mia  0.487687 -1.620536e-15          1           False         True
```




Salary Distribution
Count
Salary
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Count of Purchases (Yes/No)
count
Purchased
5
4
3
2
1
0
0
1

Program #5.2	Date :

5.2. Using the given dataset (dirtydata.csv),

- **5.2.1 Handle the data with empty cells (Use dropna() and fillna())**
- **5.2.2 Replace the empty cells using mean, median, and mode.**
- **5.2.3 Handle the data in the wrong format.**
- **5.2.4 Handle the wrong data from the dataset.**
- **5.2.5 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:

```
Original Dataset:
      Name   Age      Salary Date_of_Joining Country
0     John  25.0         NaN      2022-03-01     USA
1     Emma  32.0       54000      wrong_date  Canada
2    Lucas   NaN       62000      2020-06-15     USA
3   Sophia  29.0       48000      2019-12-10  France
4   Olivia  28.0       54000             NaN     USA
5     Liam  35.0  wrong_data      2018-07-01  France
6      Ava  30.0       60000      2017-09-23  Canada
7     Noah  40.0       65000      2016-11-04  France
8      Mia  36.0       60000      2018-07-01     USA

Dataset after dropping rows with missing values:
      Name   Age      Salary Date_of_Joining Country
1     Emma  32.0       54000      wrong_date  Canada
3   Sophia  29.0       48000      2019-12-10  France
5     Liam  35.0  wrong_data      2018-07-01  France
6      Ava  30.0       60000      2017-09-23  Canada
7     Noah  40.0       65000      2016-11-04  France
8      Mia  36.0       60000      2018-07-01     USA

Dataset after filling missing values with 'Unknown':
      Name      Age      Salary Date_of_Joining Country
0     John     25.0     Unknown      2022-03-01     USA
1     Emma     32.0       54000      wrong_date  Canada
2    Lucas  Unknown       62000      2020-06-15     USA
3   Sophia     29.0       48000      2019-12-10  France
4   Olivia     28.0       54000         Unknown     USA
5     Liam     35.0  wrong_data      2018-07-01  France
6      Ava     30.0       60000      2017-09-23  Canada
7     Noah     40.0       65000      2016-11-04  France
8      Mia     36.0       60000      2018-07-01     USA
```

```
Dataset after filling missing values using mean for Age and median for Salary:
      Name     Age   Salary Date_of_Joining Country
0     John  25.000  60000.0      2022-03-01     USA
1     Emma  32.000  54000.0      wrong_date  Canada
2    Lucas  31.875  62000.0      2020-06-15     USA
3   Sophia  29.000  48000.0      2019-12-10  France
4   Olivia  28.000  54000.0             NaN     USA
5     Liam  35.000  60000.0      2018-07-01  France
6      Ava  30.000  60000.0      2017-09-23  Canada
7     Noah  40.000  65000.0      2016-11-04  France
8      Mia  36.000  60000.0      2018-07-01     USA

Dataset after handling wrong date format:
      Name     Age   Salary Date_of_Joining Country
0     John  25.000  60000.0      2022-03-01     USA
1     Emma  32.000  54000.0             NaT  Canada
2    Lucas  31.875  62000.0      2020-06-15     USA
3   Sophia  29.000  48000.0      2019-12-10  France
4   Olivia  28.000  54000.0             NaT     USA
5     Liam  35.000  60000.0      2018-07-01  France
6      Ava  30.000  60000.0      2017-09-23  Canada
7     Noah  40.000  65000.0      2016-11-04  France
8      Mia  36.000  60000.0      2018-07-01     USA
```

```
Duplicate entries:
 Empty DataFrame
Columns: [Name, Age, Salary, Date_of_Joining, Country]
Index: []

Dataset after removing duplicates:
      Name     Age   Salary Date_of_Joining Country
0     John  25.000  60000.0      2022-03-01     USA
1     Emma  32.000  54000.0             NaT  Canada
2    Lucas  31.875  62000.0      2020-06-15     USA
3   Sophia  29.000  48000.0      2019-12-10  France
4   Olivia  28.000  54000.0             NaT     USA
5     Liam  35.000  60000.0      2018-07-01  France
6      Ava  30.000  60000.0      2017-09-23  Canada
7     Noah  40.000  65000.0      2016-11-04  France
8      Mia  36.000  60000.0      2018-07-01     USA
```

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:

```
Original Cricketer Dataset:
                Name  Age   Runs  Wickets  Matches
0       Virat Kohli   34  12000        4      250
1      Rohit Sharma   36  10000        8      200
2          MS Dhoni   39  10500        1      350
3  Sachin Tendulkar   47  18426      154      463
4     Hardik Pandya   28   3500       55       80
5   Ravindra Jadeja   32   2400      185      100
6      Shubman Gill   24   1800        0       30
7      Rishabh Pant   26   2500        0       50
8      Yuvraj Singh   41   8700      111      300

Cricketer Dataset with 'Experience Category':
                Name  Age   Runs  Wickets  Matches Experience Category
0       Virat Kohli   34  12000        4      250           Mid-level
1      Rohit Sharma   36  10000        8      200         Experienced
2          MS Dhoni   39  10500        1      350         Experienced
3  Sachin Tendulkar   47  18426      154      463         Experienced
4     Hardik Pandya   28   3500       55       80           Mid-level
5   Ravindra Jadeja   32   2400      185      100           Mid-level
6      Shubman Gill   24   1800        0       30               Young
7      Rishabh Pant   26   2500        0       50           Mid-level
8      Yuvraj Singh   41   8700      111      300         Experienced
```

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,

5.4.1 Draw the line of linear regression
5.4.2 Evaluate how well the data fit in linear regression.
5.4.3 Predict the speed of a 10-year-old car.

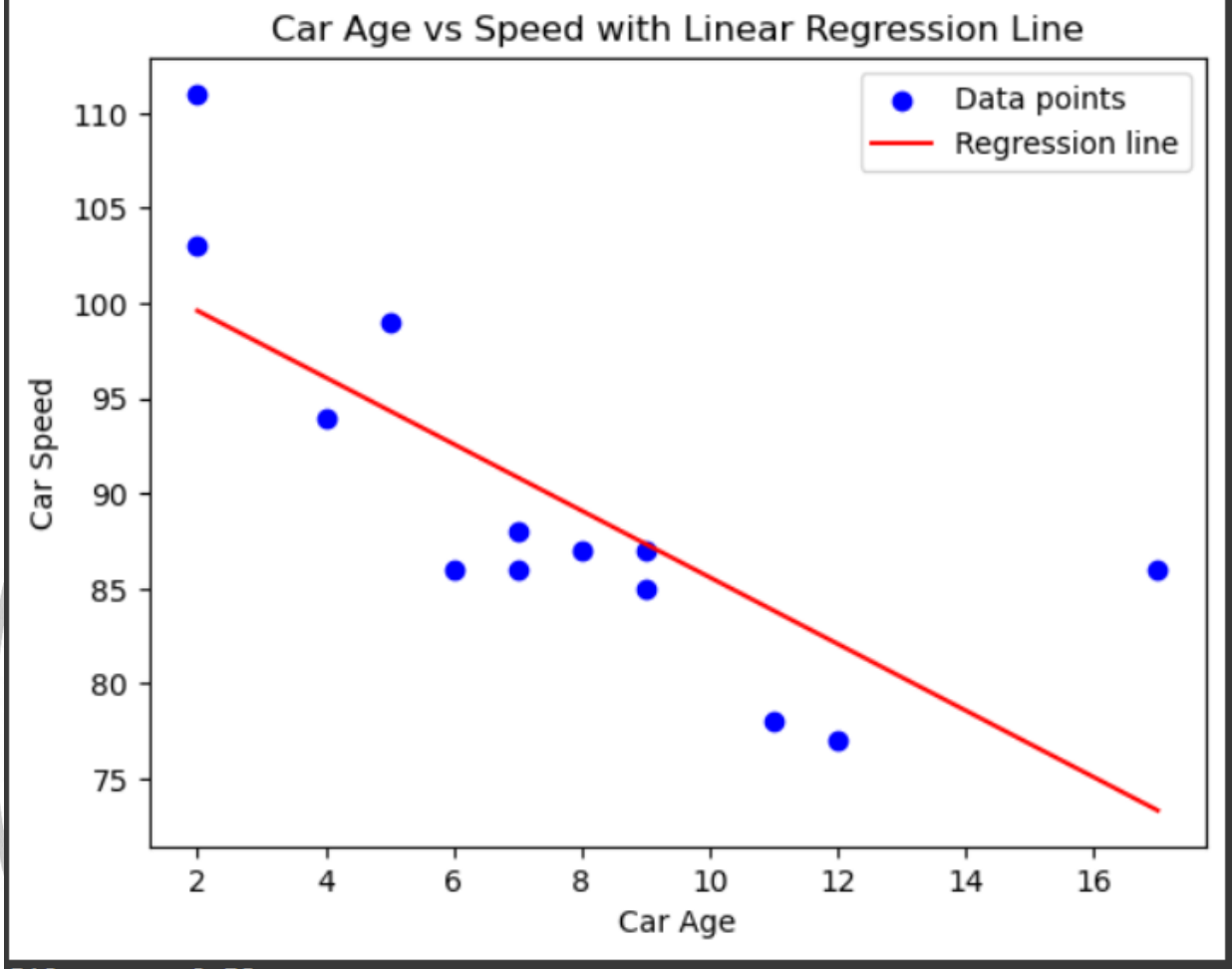
SOURCE CODE:

```
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:

```
Regression Line: y = -1.75x + 103.11
```




```
R^2 score: 0.58
Predicted speed for a 10-year-old car: 85.59 mph
```

Program #5.5	Date :

Using the dataset (cars.csv),

5.5.1 Predict the CO2 emissions of a car with a weight of 2300 kg and volume of 1300 cm3.

5.5.2 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:

```
   Weight  Volume  CO2_Emissions
0    1500    1200            130
1    1600    1300            150
2    1700    1400            170
3    1800    1500            190
4    1900    1600            210
Coefficients: [0.10759261 0.09240739]
Intercept: -142.27778367854444
Predicted CO2 emissions for a car with weight 2300 kg and volume 1300 cm3: 225.31 g/km
```

Program #5.6	Date :

Using the insurance dataset (insurance.csv) with adequate preprocessing steps,

5.6.1 Visualize the correlation among variables using a heatmap.

5.6.2 Create a linear regression model.

5.6.3 Evaluate the model. (Find MSE and R_square.)

5.6.4 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}')
```

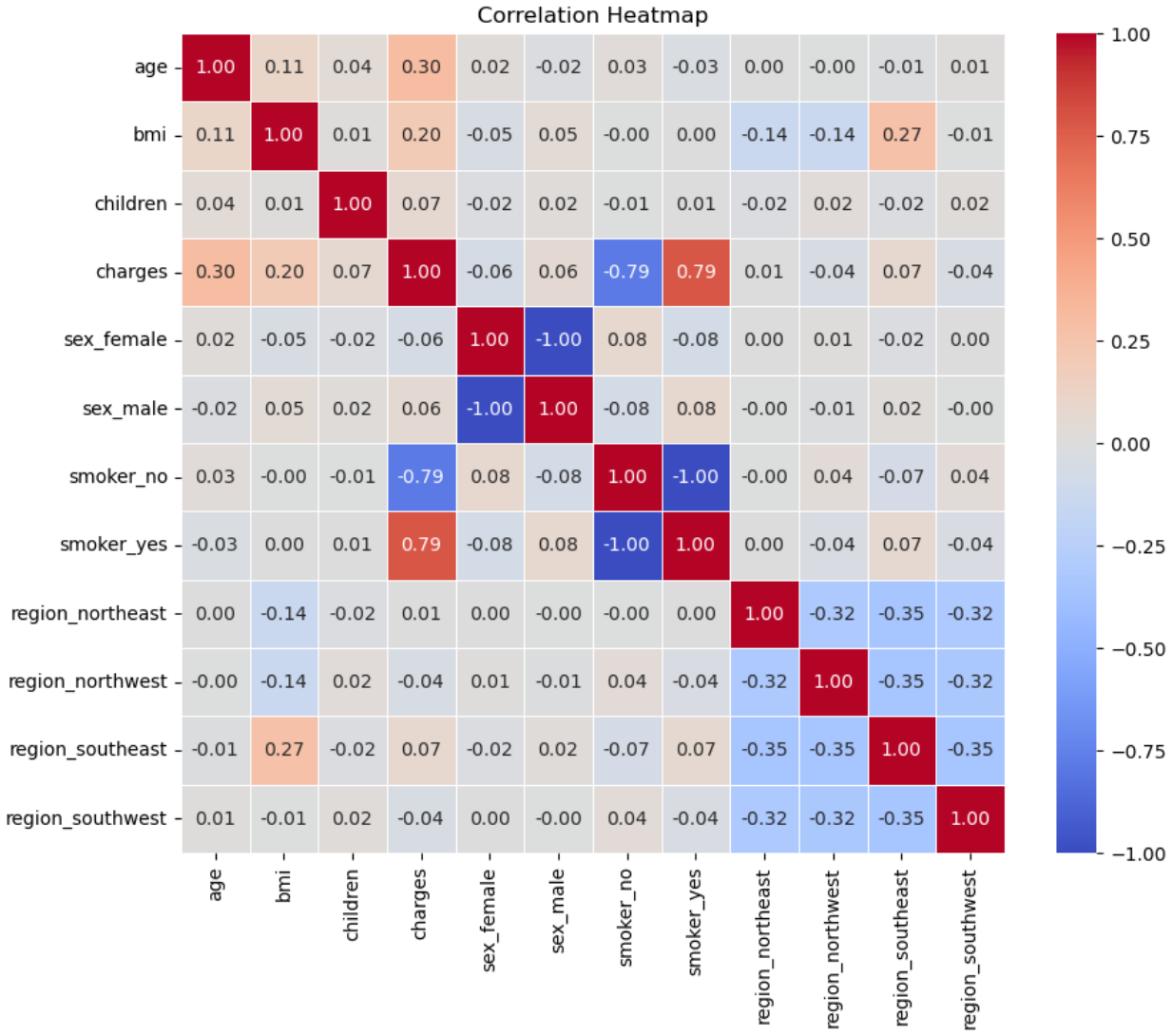
Step 5.6.4 : Predict the charges for a person with an age of 30, a BMI of 32.00, and who is a smoker.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

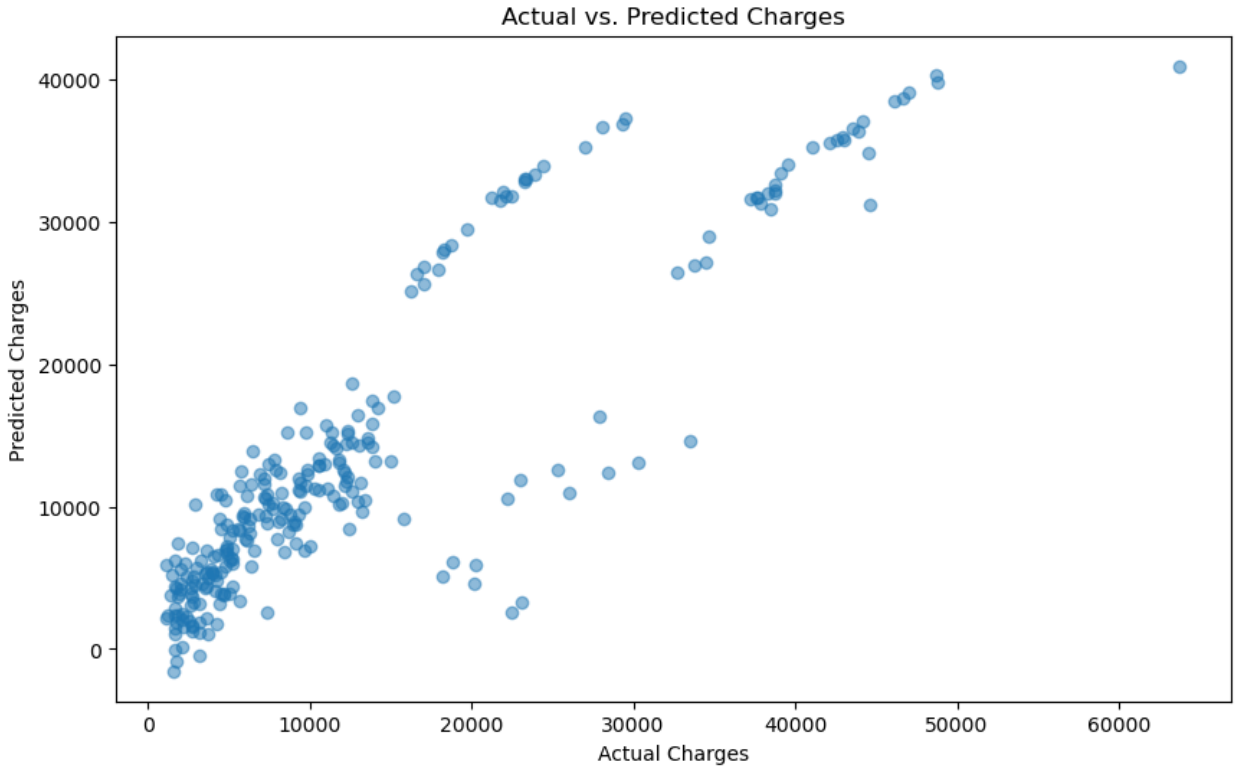
```
from sklearn.metrics import mean_squared_error, r2_score

# 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:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
None
   age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520
```




```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
None
   age     sex     bmi  children smoker     region      charges
0   19  female  27.900         0    yes  southwest  16884.92400
1   18    male  33.770         1     no  southeast   1725.55230
2   28    male  33.000         3     no  southeast   4449.46200
3   33    male  22.705         0     no  northwest  21984.47061
4   32    male  28.880         0     no  northwest   3866.85520
Mean Squared Error (MSE): 33596915.85136147
Mean Absolute Error (MAE): 4181.194473753643
R-squared (R2): 0.7835929767120723
```




```
Mean Squared Error (MSE): 33596915.85136147
R-squared (R2): 0.7835929767120723
```

```
Predicted charges: $29,737.97
```

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.)	

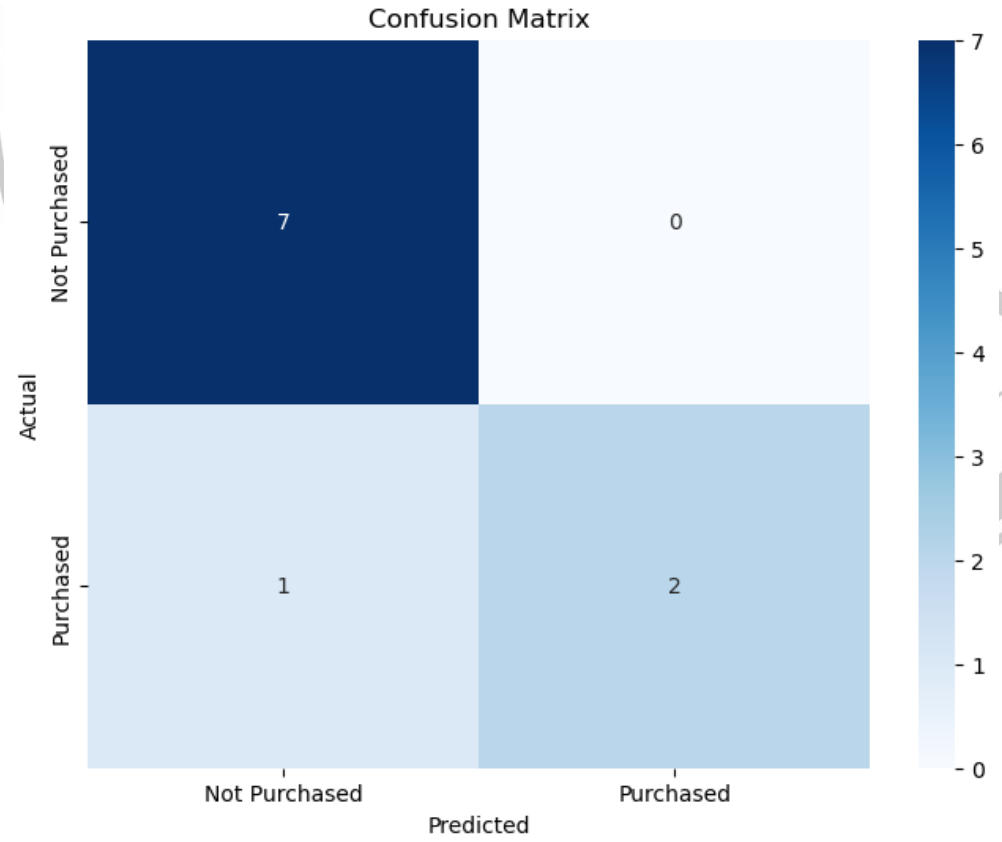
SOURCE CODE:

```
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:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             49 non-null     int64
 1   Income          49 non-null     int64
 2   Gender          49 non-null     object
 3   Education       49 non-null     object
 4   Marital_Status  49 non-null     object
 5   Purchased       49 non-null     int64
dtypes: int64(3), object(3)
memory usage: 2.4+ KB
None
   Age  Income  Gender    Education Marital_Status  Purchased
0   29   56000  Female    Bachelors        Married          0
1   34   72000    Male      Masters         Single          1
2   41   62000  Female          PhD        Married          1
3   22   48000    Male  High School         Single          0
4   37   75000  Female    Bachelors        Married          1
Accuracy: 0.90
Precision: 1.00
Recall: 0.67
F1 Score: 0.80
```



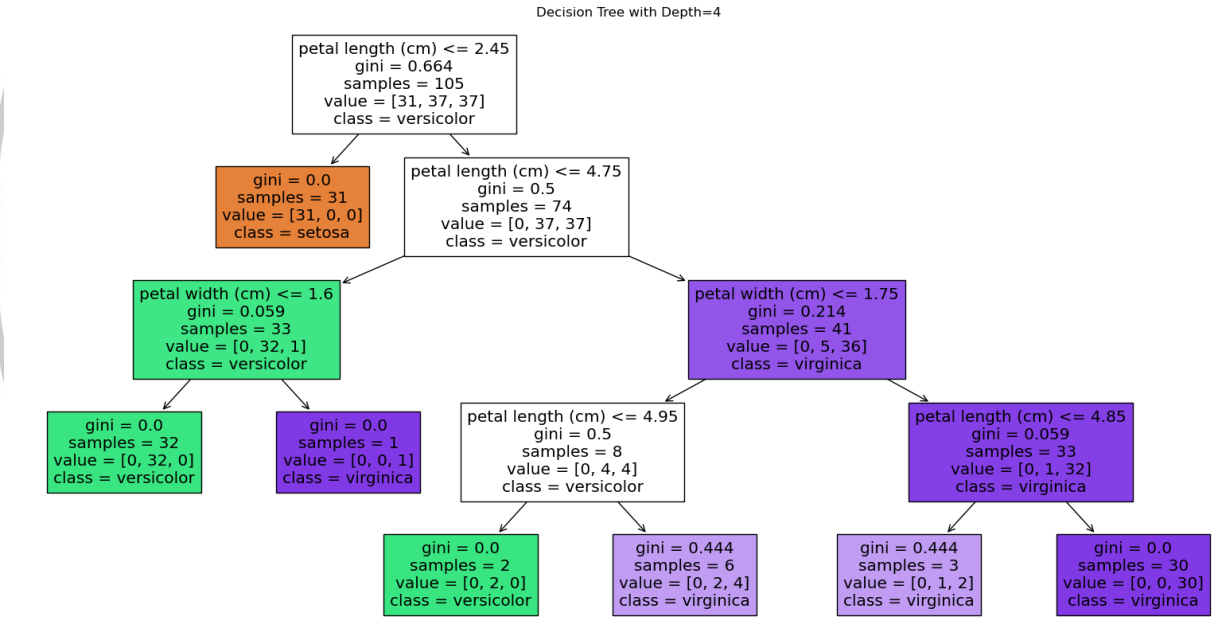
Program #5.8	Date :
Use the Iris dataset to visualize a decision tree with a depth=4 and save the plot as a PNG file. Also, print the confusion matrix and generate the classification report.	

SOURCE CODE:

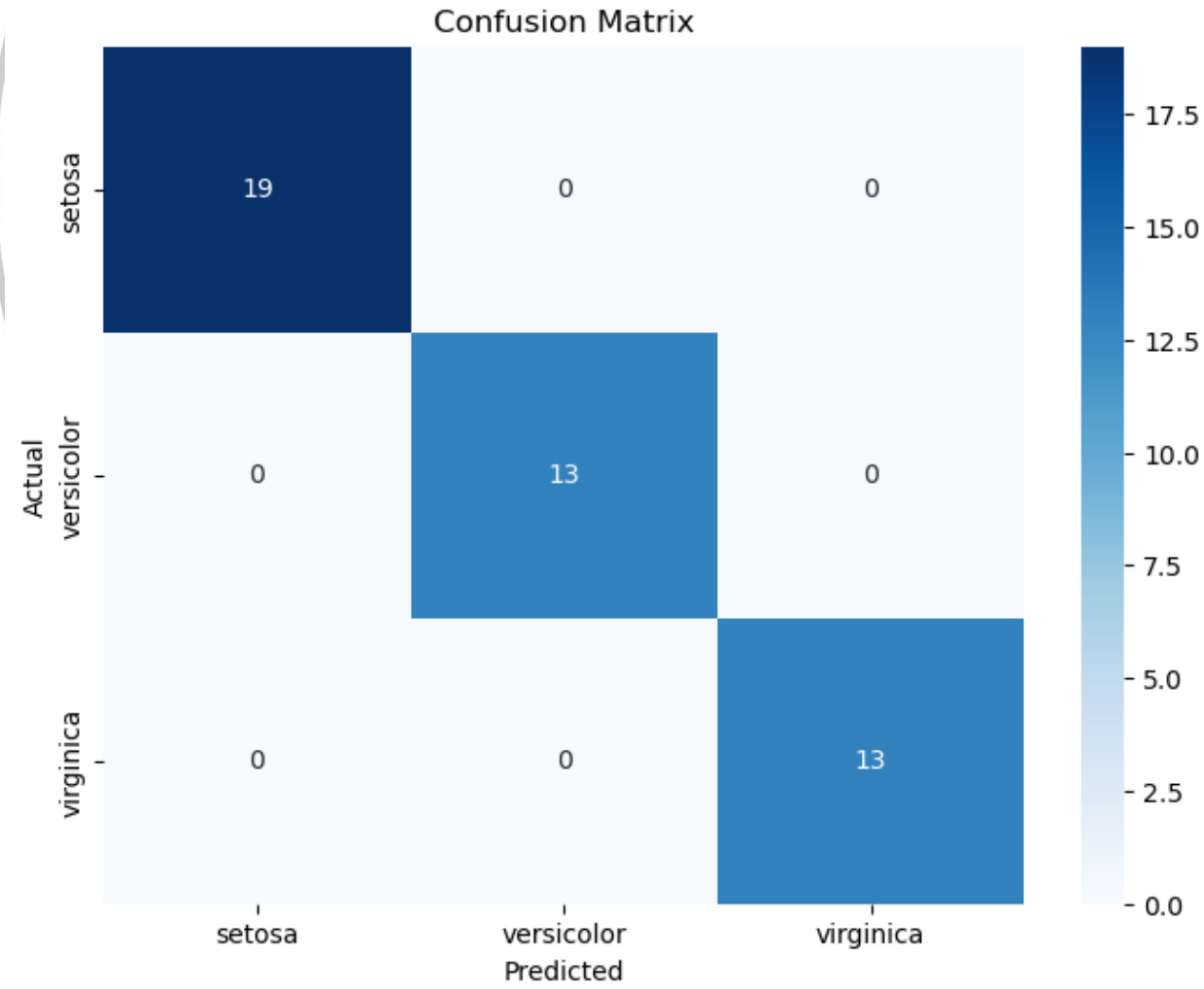
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# 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:



```
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



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.	

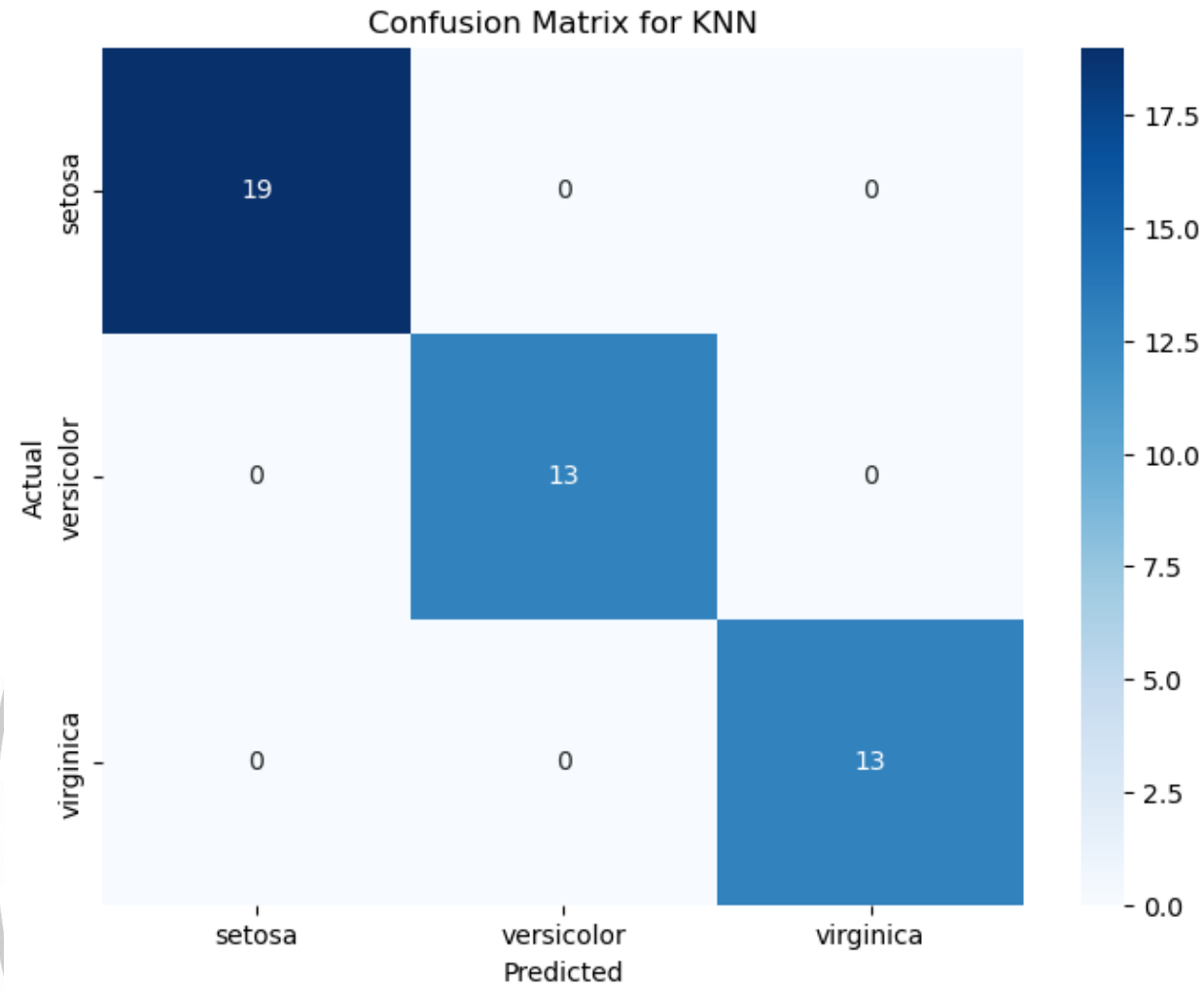
SOURCE CODE:

```
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:

```
Accuracy of KNN model with k=5: 1.00
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```


Confusion Matrix for KNN
setosa
versicolor
virginica
Actual
19
0
0
0
13
0
0
0
13
setosa
versicolor
virginica
Predicted
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

Program #5.10	Date :
Analyze the given dataset (gym_data.csv) using RandomForestRegressor and visualize the 'Effect of n_estimators.	

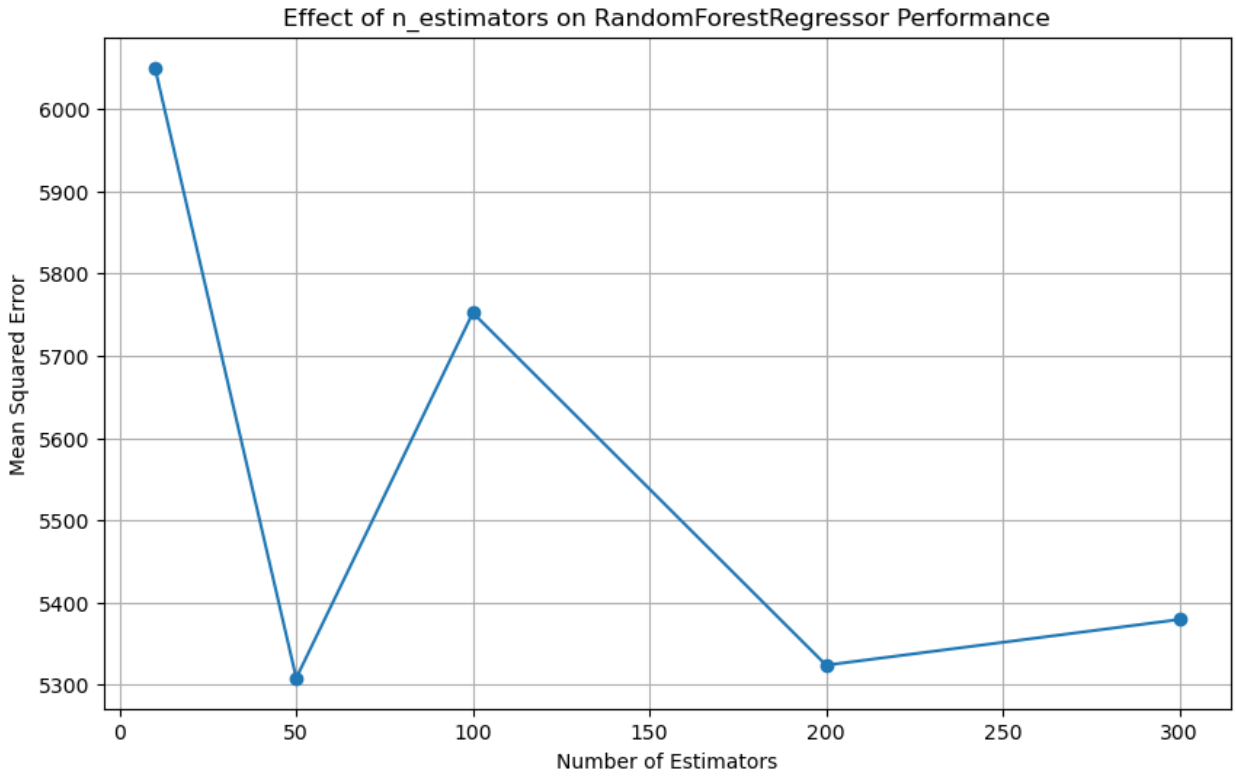
SOURCE CODE:

```
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:

```
   feature1  feature2  feature3  target
0        25        10         5     200
1        30        15         6     250
2        35        20         7     300
3        40        25         8     350
4        45        30         9     400
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   feature1  16 non-null     int64
 1   feature2  16 non-null     int64
 2   feature3  16 non-null     int64
 3   target    16 non-null     int64
dtypes: int64(4)
memory usage: 644.0 bytes
None
n_estimators: 10, MSE: 6050.00
n_estimators: 50, MSE: 5308.20
n_estimators: 100, MSE: 5752.10
n_estimators: 200, MSE: 5323.89
n_estimators: 300, MSE: 5379.34
```


Effect of n_estimators on RandomForestRegressor Performance
Mean Squared Error
6000
5900
5800
5700
5600
5500
5400
5300
0
50
100
150
200
250
300
Number of Estimators

Program #5.11	Date :
Visualize a 3-dimensional cluster using the given dataset 'Mall_Customers.csv', where no_of_clusters = 5.	

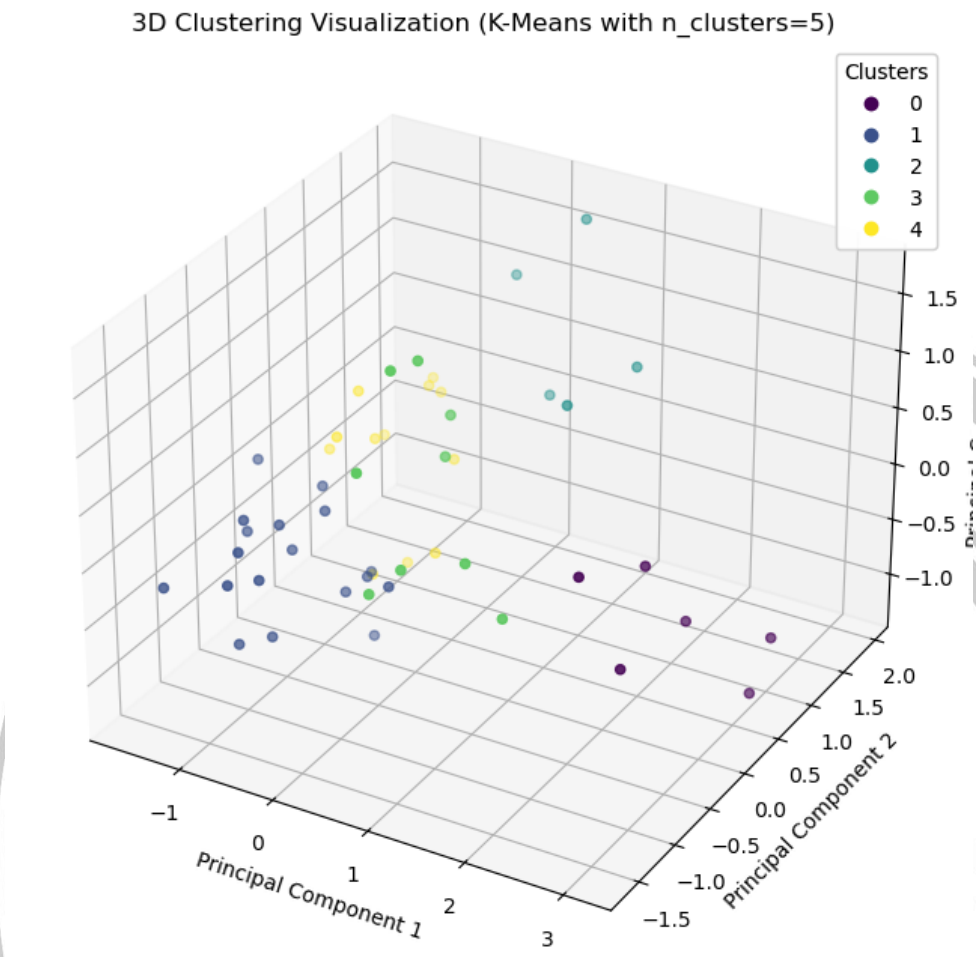
SOURCE CODE:

```
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:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Female	19	15	39
1	2	Female	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40


3D Clustering Visualization (K-Means with n_clusters=5)
Clusters
0
1
2
3
4
1.5
1.0
0.5
0.0
−0.5
−1.0
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Principal Component 2
−1
0
1
2
3
Principal Component 1

Program #5.12	Date :

Using the dataset provided (Online Retail.xlsx),

5.12.1 Split the data according to the region of the transaction.

5.12.2 Build the models using the apriori algorithm.

5.12.3 Develop the association rules.

5.12.4 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:

```
   InvoiceNo StockCode                          Description  Quantity  \
0     536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1     536365     71053                  WHITE METAL LANTERN         6
2     536365    84406B           CREAM CUPID'S BOW TEA COSY         8
3     536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4     536366     84879       RED WOOLLY HOTTIE WHITE HEART.         6

      InvoiceDate  UnitPrice  CustomerID         Country
0  12/1/2010 8:26       2.55       17850  United Kingdom
1  12/1/2010 8:26       3.39       17850  United Kingdom
2  12/1/2010 8:26       2.75       17850  United Kingdom
3  12/1/2010 8:26       3.39       17850  United Kingdom
4  12/1/2010 8:28       3.39       17851  United Kingdom
Frequent Itemsets:
    support                             itemsets
0  0.166667           (CREAM CUPID'S BOW TEA COSY)
1  0.166667  (KNITTED UNION FLAG HOT WATER BOTTLE)
2  0.166667        (MINT GREEN RABBIT NIGHT LIGHT)
3  0.166667             (PINK FAIRY CAKE GIFT BOX)
4  0.166667               (RED RABBIT NIGHT LIGHT)
Association Rules:
                         antecedents  \
0           (CREAM CUPID'S BOW TEA COSY)
1  (KNITTED UNION FLAG HOT WATER BOTTLE)
2           (CREAM CUPID'S BOW TEA COSY)
...
1   0.166667              (KNITTED UNION FLAG HOT WATER BOTTLE)
20  0.166667  (WHITE METAL LANTERN, WHITE HANGING HEART T-LI...
19  0.166667  (CREAM CUPID'S BOW TEA COSY, WHITE METAL LANTE...
18  0.166667  (CREAM CUPID'S BOW TEA COSY, WHITE METAL LANTE...
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Program #6	Date :
Project	

SOURCE CODE

Model.py

```
#pip install transformers -U
#pip install datasets
from datasets import load_dataset
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score,confusion_matrix,
ConfusionMatrixDisplay
import torch
import matplotlib.pyplot as plt
from transformers import TrainingArguments, Trainer

dataset = load_dataset("AiresPucrs/toxic-comments")
x = dataset['train']['comment_text']
y = dataset['train']['toxic']

from transformers import BertTokenizer, BertForSequenceClassification
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=2)
model = model.to('cuda')

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(x, y, test_size=0.2,stratify=y)
X_train_tokenized = tokenizer(X_train, padding=True, truncation=True, max_length=512)
X_val_tokenized = tokenizer(X_val, padding=True, truncation=True, max_length=512)
X_train_tokenized.keys()

class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
```

```
        return len(self.encodings["input_ids"])
train_dataset = Dataset(X_train_tokenized, y_train)
val_dataset = Dataset(X_val_tokenized, y_val)

def compute_metrics(p):
    print(type(p))
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    cm = confusion_matrix(y_true=labels, y_pred=pred)
    disp = ConfusionMatrixDisplay(confusion_matrix = cm , display_labels = ['Toxic','Non-Toxic'])
    f1 = f1_score(y_true=labels, y_pred=pred)
    disp.plot()
    plt.show()

    return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
#!pip install transformers==4.17
args = TrainingArguments(
    output_dir="output",
    num_train_epochs=1,
    per_device_train_batch_size=8

)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
trainer.evaluate()
```

modeltest.py

```
from transformers import BertForSequenceClassification, BertTokenizer
import torch

# Load saved model and tokenizer
model = BertForSequenceClassification.from_pretrained(r'Z:\Python_Dataanalytics
```

```
project\SafeSpeak\Toxic Model')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
#new_comment = "that was good point"
# new_comment = "go to hell"
new_comment=input("Comment : ")
encoded_comment = tokenizer(new_comment, padding='max_length', truncation=True,
max_length=128, return_tensors='pt')
output = model(**encoded_comment)
probabilities = torch.nn.functional.softmax(output.logits, dim=-1)
predicted_label = torch.argmax(probabilities, dim=-1)
if predicted_label == 0:
   print("Toxic comment.")
else:
   print("Non-Toxic comment")

print("Predicted Label:", predicted_label.item()
```

Performance Metrics

OUTPUT

```
Comment : Go to hell
Toxic comment.
Predicted Label: 0
PS Z:\Python_Dataanalytics project\SafeSpeak>
```