Lab Manual

ADVANCED PYTHON PROGRAMMING

PANDIT DEENDAYAL ENERGY UNIVERSITY GANDHINAGAR, GUJARAT, INDIA

SCHOOL OF TECHNOLOGY

Computer Science & Engineering

LAB File (2024-25)

Advanced Python Programming for Emerging Application's (23CP301P)

Student Name: Patel Honey Ashishkumar

Enrollment No.: 22BCP402

Semester: 5

Division: 6

Group: 11

Table of Contents

LAB Assignment 1: Product Review from text file

Consider a scenario where you are working as a data scientist for a large e-commerce company. Your team is responsible for analyzing customer feedback data, which is stored in multiple text files. Each text file contains customer reviews for different product categories. Your task is to write a Python script that performs the following operations:

Read the contents of all the text files in a given

directory. For each review, extract the following

information:

- Customer ID (a 6-digit alphanumeric code)
- Product ID (a 10-digit alphanumeric code)
- Review date (in the format "YYYY-MM-DD")
- Review rating (an integer between 1 and 5)
- Review text (the actual feedback provided by the customer)

Calculate the average review rating for each product and store it in a dictionary where the product ID is the key and the average rating is the value.

Determine the top 3 products with the highest average review ratings.

Create a new text file named "summary.txt" and write the following information into it:

- The total number of reviews processed.
- The total number of valid reviews (reviews with all required information extracted successfully).
- The total number of invalid reviews (reviews with missing or incorrect information).
- The product ID and average rating of the top 3 products with the highest average ratings.

Your Python script should be robust, handling any potential errors or exceptions during the file handling process. Additionally, you should implement efficient algorithms to handle large volumes of data without consuming excessive memory or processing time.

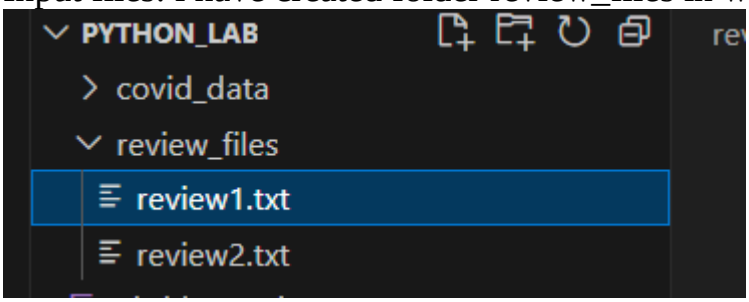
Write the Python script to achieve the above objectives and provide detailed comments explaining each step of your implementation.

Source code:

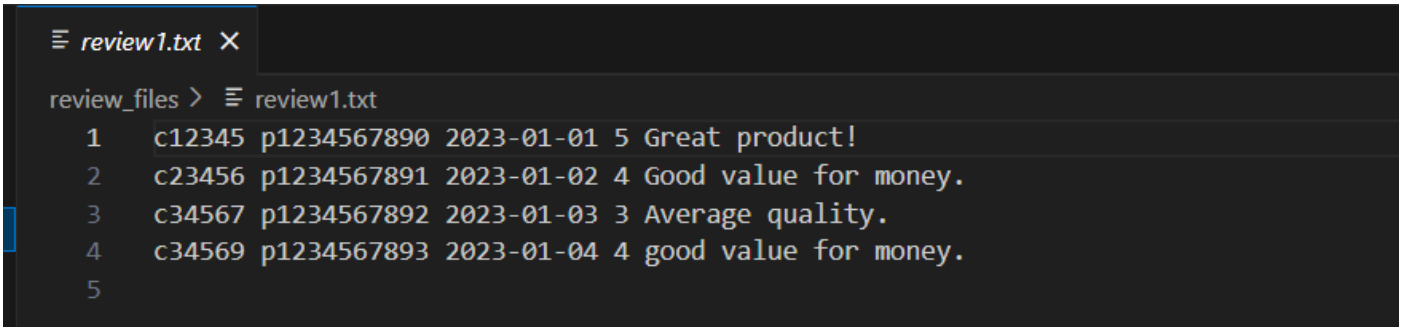
```
import os

# Function to read all reviews from a file
def read_all_reviews(file_path):
    reviews = []
    invalid_reviews = 0

    with open(file_path, 'r') as f:
        for line in f:
            parts = line.split(' ', 4)
            if len(parts) == 5:
                custID = parts[0]
                prodID = parts[1]
                revdate = parts[2]
                revrate = parts[3]
                revtext = parts[4]
                try:
                    revrate = int(revrate)
                    reviews.append((custID, prodID, revdate, revrate, revtext))
                except ValueError:
                    invalid_reviews += 1
            else:
                invalid_reviews += 1

    return reviews, invalid_reviews

# Function to calculate average ratings
def calculate_average_ratings(reviews):
    product_ratings = {}

    for review in reviews:
        prodID = review[1]
        revrate = review[3]

        if prodID not in product_ratings:
            product_ratings[prodID] = []
        product_ratings[prodID].append(revrate)

    average_ratings = {prodID: sum(ratings) / len(ratings) for prodID, ratings in
```

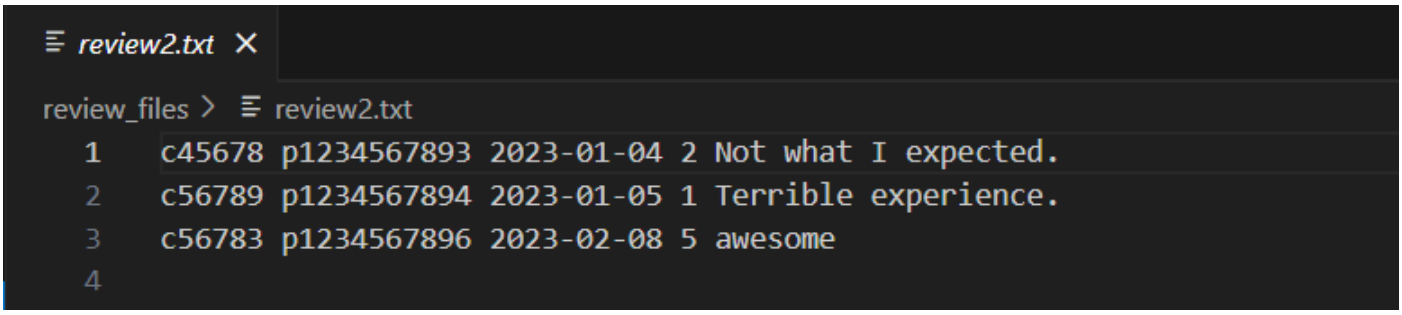
```
product_ratings.items()}

    return average_ratings

# Function to write results to a file
def write_results(filename, total_reviews, total_valid, total_invalid, top_products):
    with open(filename, 'w') as f:
        f.write('Summary Report\n')
        f.write('Total reviews: ' + str(total_reviews) + '\n')
        f.write('Total valid reviews: ' + str(total_valid) + '\n')
        f.write('Total invalid reviews: ' + str(total_invalid) + '\n')
        f.write('Top 3 products:\n')
        for prod in top_products:
            f.write('Product ID: ' + prod[0] + ' Average rating: ' + str(prod[1]) + '\n')

# Main function
def main():
    review_directory = 'review_files'  # Directory containing the review files
    review_files = [os.path.join(review_directory, file) for file in
os.listdir(review_directory)]

    all_reviews = []
    total_invalid_reviews = 0

    for file in review_files:
        reviews, invalid_reviews = read_all_reviews(file)
        all_reviews.extend(reviews)
        total_invalid_reviews += invalid_reviews

    total_reviews = len(all_reviews)
    average_ratings = calculate_average_ratings(all_reviews)
    sorted_avg_ratings = sorted(average_ratings.items(), key=lambda x: x[1], reverse=True)

    top_products = sorted_avg_ratings[:3]

    # Writing results to summary.txt
    write_results('summary.txt', total_reviews, total_reviews - total_invalid_reviews,
total_invalid_reviews, top_products)

    # Print confirmation message
    print("Name:Honey Patel")
    print("Roll number: 22BCP402")
    print("summary.txt file has been generated successfully!")

if __name__ == "__main__":
    main()
```

Input files: i have created folder review_files in which all files are stored.

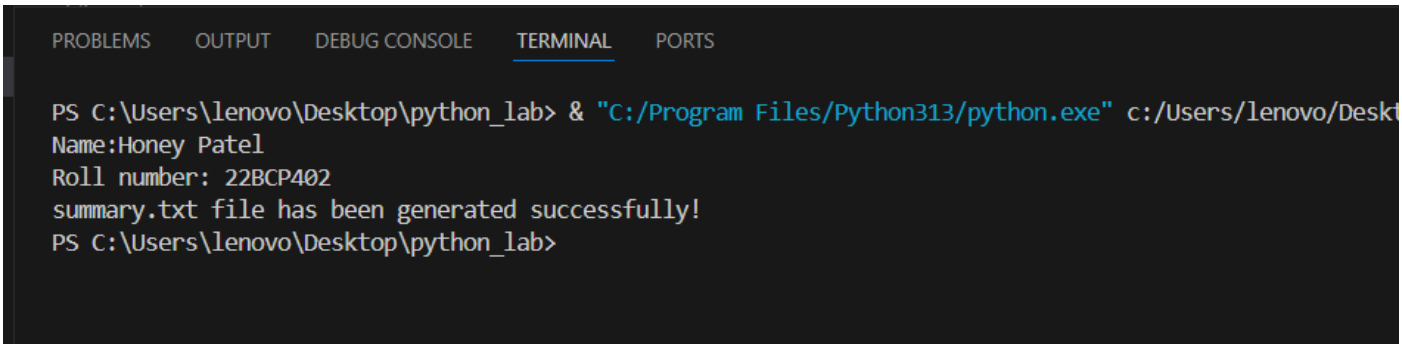


Input file(review1.txt, review2.txt)





<u>Output:</u> summary.txt is created.



Lab1.py summary.txt

summary.txt

```
Summary Report
Total reviews: 7
Total valid reviews: 7
Total invalid reviews: 0
Top 3 products:
Product ID: p1234567890 Average rating: 5.0
Product ID: p1234567896 Average rating: 5.0
Product ID: p1234567891 Average rating: 4.0
```

LAB Assignment 2: Railway Ticket Reservation System

You are tasked with developing a railway ticket reservation system for a busy rail network. The system should handle ticket booking, seat availability, and generate reports for the railway administration. Your task is to implement a Python program that provides the following functionalities:

Load Train Data: The program should read the train data from a CSV file named "trains.csv." Each row in the CSV file represents a train with the following information:

- Train ID (a unique alphanumeric code)
- Train Name
- Source Station
- Destination Station
- Total Seats (total number of seats available on the train)

Load Passenger Data: The program should read the passenger data from a CSV file named "passengers.csv." Each row in the CSV file represents a passenger with the following information:

- Passenger Name
- Train ID (the ID of the train the passenger wants to book a ticket on)
- Number of Tickets (the number of tickets the passenger wants to book)

Check Seat Availability: Given the train ID and the number of tickets requested by a passenger, the program should check if there are enough seats available on the specified train for booking. If seats are available, the booking should be confirmed, and the total fare for the booking should be calculated as per the fare rules (you can define fare rules based on distance, class, etc.).

Update Seat Availability: After confirming the booking, the program should update the seat availability for the corresponding train.

Generate Reports:

Report 1: The program should generate a report showing the details of all the trains, including their names, source stations, destination stations, and the total number of seats available on each train.

Report 2: The program should generate a report showing the total revenue earned from each train based on the total number of confirmed bookings and their respective fares.

Handle Errors: The program should handle various types of errors gracefully, such as invalid train IDs, invalid passenger names, insufficient seats, etc., and provide appropriate error messages.

Note:

You can assume that the passenger data in "passengers.csv" will not exceed the available seats on

any train. You can design the fare rules based on your preference and mention them clearly in the

program.

Write the Python program to implement the above functionalities for the railway ticket reservation system. Use comments to explain each step of your implementation and provide sample CSV files ("trains.csv" and "passengers.csv") for testing the program.

Source code:

```
import csv

# Function to load train data from "trains.csv"
def load_train_data():
    trains = []
    with open("trains.csv", "r") as file:
        reader = csv.DictReader(file)
        #keys are taken from the first row (the header row) and values are the data from the respective row

        for row in reader:
            #extract the column Train ID and then assign it to key train_id ,We can append a new dictionary to the list
of dictionaries by using the Python append() method

            trains.append({
                "train_id": row["Train ID"],
                "train_name": row["Train Name"],
                "source_station": row["Source Station"],
                "destination_station": row["Destination Station"],
                "total_seats": int(row["Total Seats"]),
                "available_seats": int(row["Available Seats"]),
                "fare_per_ticket": int(row["Total fare"])
            })
    return trains

# Function to load passenger data from "passengers.csv"
def load_passenger_data():
    passengers = []
    with open("passengers.csv", "r") as file:
        reader = csv.DictReader(file)
        for row in reader:
            # Directly use expected column names

            passengers.append({
                "passenger_name": row["Passenger Name"],
                "train_id": row["Train ID"],
                "tickets": int(row["Number of Tickets"])
            })
    return passengers

# Function to check seat availability and book tickets, check if seat is availability for passengers and books a tickes
if possible
def check_and_book_tickets(trains, passenger):
    for train in trains:
        if train["train_id"] == passenger["train_id"]:  # Checks if the train ID in the current train matches the train ID
in the passenger's booking.

            if train["available_seats"] >= passenger["tickets"]: #if the available seat is greater then passengers requested
ticket
                #if yes then
                train["available_seats"] -= passenger["tickets"]  #it will do changes in available tickes=available tickets-
passenger booked tickets
```

```
            total_fare = train["fare_per_ticket"] * passenger["tickets"] #calculating the total
cost=ticketcost*passenger's num of tickets book
            print("Booking confirmed for %s. Total fare: %d" % (passenger['passenger_name'], total_fare))
            return total_fare
        else:
            print("Insufficient seats available for %s." % passenger['passenger_name'])  #if availableseat not greater
than passengers tickets then
            return 0
    print("Train ID %s not found." % passenger['train_id'])
    return 0

# Function to generate train report
def generate_train_report(trains):
    print("Train Report:")
    for train in trains:
        print(f"Train ID: {train['train_id']}, Name: {train['train_name']}, "
              f"Source: {train['source_station']}, Destination: {train['destination_station']}, "
              f"Available Seats: {train['available_seats']}")
# Function to generate revenue report
def generate_revenue_report(trains, total_revenue):
    print("Revenue Report:")
    for train in trains:
        print(f"Train ID: {train['train_id']}, Name: {train['train_name']}, "
              f"Total Revenue: {total_revenue[train['train_id']]}")

# Main function
def main():
    trains = load_train_data()
    passengers = load_passenger_data()

    # Initialize total_revenue dictionary using a for loop
    total_revenue = {}  #this will track the total revenue generated by each train
    for train in trains:   #etrates through each trsin in trains list
        train_id = train["train_id"]   #retrive the trin id
        total_revenue[train_id] = 0   #initializa for this train total revenue as 0

    for passenger in passengers:
        fare = check_and_book_tickets(trains, passenger)   #this function return the total_fare if booking is successful
        if fare > 0:    #so if fare greather than 0 then calculate the total revenue
            total_revenue[passenger["train_id"]] += fare  #Adds the fare to the total revenue for the corresponding train
in the total_revenue

    generate_train_report(trains)   # this function to generate the report
    generate_revenue_report(trains, total_revenue)
    print("Nmae: Patel Honey")
    print("Roll number: 22BCP402")


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

Passengers.csv – input file

```
passengers.csv
Passenger Name,Train ID,Number of Tickets
Saanvi,T123,2
Rajvi,T321,1
Ashish,T789,4
Niya,T456,3
honey,T123,7
aksh,T123,2
keni,T789,7
```

Trains.csv

```
trains.csv
Train ID,Train Name,Source Station,Destination Station,Total Seats,Available Seats,Total fare
T123,Vnade bharat,City A,City B,200,198,500
T456,Gujrat mail,City B,City C,150,147,1200
T789,Shatabdi ,Ahemedabad,Mohaali,100,96,1400
T321,Tejas Express,City D,City E,180,179,200
T654,Rajdhani Express,City E,City F,250,250,1500
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/Users/lenovo/Desktop/python_lab/railway.py
Booking confirmed for Saanvi. Total fare: 1000
Booking confirmed for Rajvi. Total fare: 200
Booking confirmed for Ashish. Total fare: 5600
Booking confirmed for Niya. Total fare: 3600
Booking confirmed for honey. Total fare: 3500
Booking confirmed for aksh. Total fare: 1000
Booking confirmed for keni. Total fare: 9800
Train Report:
Train ID: T123, Name: Vnade bharat, Source: City A, Destination: City B, Available Seats: 187
Train ID: T456, Name: Gujrat mail, Source: City B, Destination: City C, Available Seats: 144
Train ID: T789, Name: Shatabdi , Source: Ahemedabad, Destination: Mohaali, Available Seats: 85
Train ID: T321, Name: Tejas Express, Source: City D, Destination: City E, Available Seats: 178
Train ID: T654, Name: Rajdhani Express, Source: City E, Destination: City F, Available Seats: 250
Revenue Report:
Train ID: T123, Name: Vnade bharat, Total Revenue: 5500
Train ID: T456, Name: Gujrat mail, Total Revenue: 3600
Train ID: T789, Name: Shatabdi , Total Revenue: 15400
Train ID: T321, Name: Tejas Express, Total Revenue: 200
Train ID: T654, Name: Rajdhani Express, Total Revenue: 0
Nmae: Patel Honey
Roll number: 22BCP402
PS C:\Users\lenovo\Desktop\python_lab>
```

Lab Assignment 3: CSV Manipulation

Your task is to write a Python program that reads this CSV file, calculates the average score for each student, and then creates a new CSV file named "student_average_grades.csv”

- Steps to Solve

1. Read the data from "student_grades.csv" using CSV file handling in Python.
2. For each student, calculate their average score across all subjects (Maths, Science, and English).
3. Create average functions to calculate the average for each student.
4. Store the student's name and their corresponding average score in a new dictionary.
5. Write the data from the dictionary into a new CSV file named "student_average_grades.csv" with two columns: "Name" and "Average."

CSV files:

Student_grades.csv – input file

student_grades.csv

```
Name,Maths,Science,English
Honey,85,78,92
Sweta,88,90,85
Aksh,70,80,65
Keni,95,100,98
Niya,60,65,70
```

Source code:

```
import csv

# Step 1: Read the data from "student_grades.csv"

def read_grades(file_name):
    with open(file_name, mode='r') as file:
        reader = csv.DictReader(file)
        grades = []
        for row in reader:
            grades.append({
                'Name': row['Name'],
                'Maths': float(row['Maths']),
                'Science': float(row['Science']),
                'English': float(row['English'])
```

```
            })
    return grades

# Step 3: Create average function to calculate the average for each student

def calculate_average(grades):
    for student in grades:
        student['Average'] = (student['Maths'] + student['Science'] + student['English']) /3


# Step 4: Store the student's name and their corresponding average score in a new
dictionary

def create_average_dict(grades):
    average_dict = []
    for student in grades:
        average_dict.append({'Name': student['Name'], 'Average': round(student['Average'],
2)})
    return average_dict

# Step 5: Write the data from the dictionary into a new CSV file

def write_averages(file_name, averages):
    with open(file_name, mode='w', newline='') as file:
        writer = csv.DictWriter(file, fieldnames=['Name', 'Average'])
        writer.writeheader()
        for student in averages:
            writer.writerow(student)

# Main function

def main():
    grades = read_grades('student_grades.csv')
    calculate_average(grades)
    average_dict = create_average_dict(grades)
    write_averages('student_average_grades.csv', average_dict)
     print("Name: Honey Patel")
    print("Roll number: 22BCP402")

    print("Average grades have been written to 'student_average_grades.csv'")
if __name__ == "__main__":
    main()
```

Student_avaerage_grades.csv file is created in my current directory.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/Users/lenovo/Desktop/python_lab/StudentCSV.py
Name: Honey Patel
Roll number: 22BCP402
Average grades have been written to 'student_average_grades.csv'
PS C:\Users\lenovo\Desktop\python_lab>
```

student_average_grades.csv

```
Name,Average
Honey,85.0
Sweta,87.67
Aksh,71.67
Keni,97.67
Niya,65.0
```

Lab Assignment 4 :CSV In depth

You are working as a data engineer for a large retail company. Your team is responsible for processing and analyzing sales data from multiple stores across the country. The sales data is stored in CSV files, and each file represents sales data for a specific month and year. Each CSV file has the following columns:

- Date (in the format "YYYY-MM-DD")
- Store ID (a unique alphanumeric code)
- Product ID (a unique alphanumeric code)
- Quantity sold (an integer representing the number of products sold on that date)

The "product_names.csv" file has two columns: "Product ID" and "Product Name," and it contains the mapping for all products in the sales data.

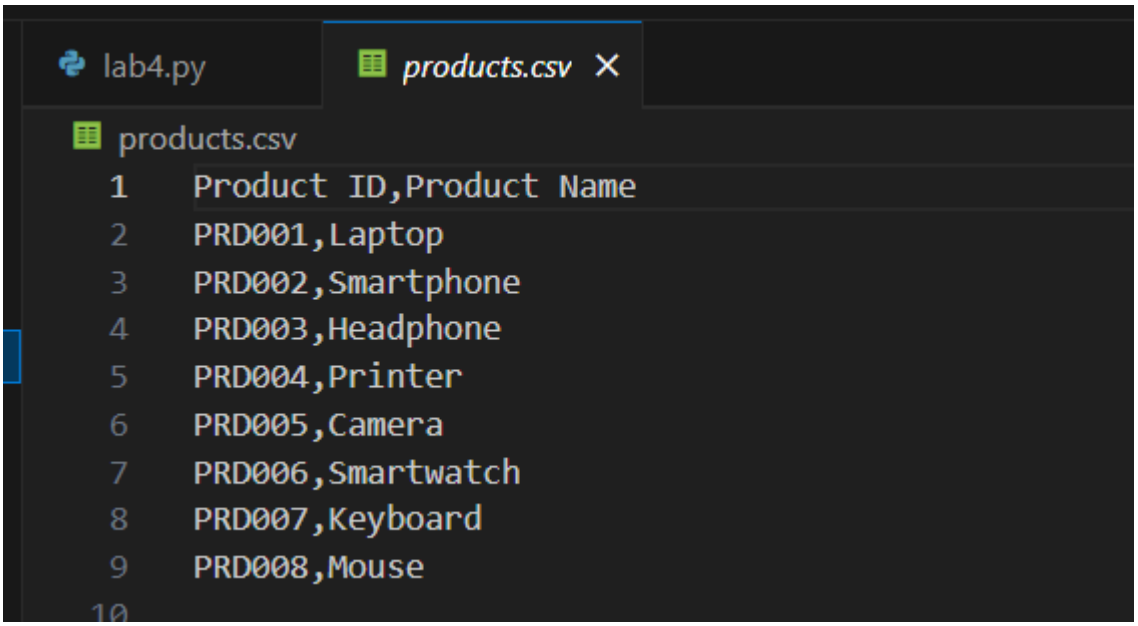
Your task is to write a Python program that performs the following operations:

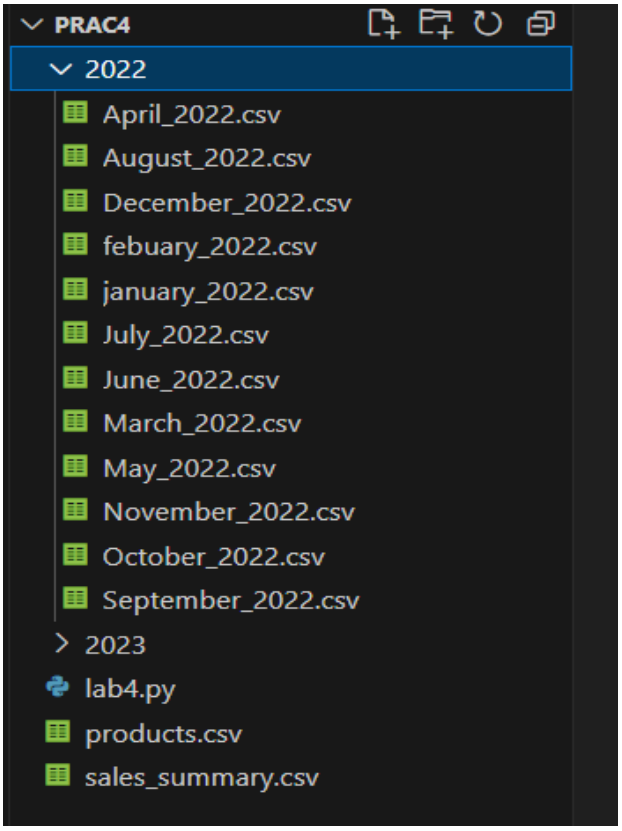
- Read the sales data from all the CSV files in a given directory and its subdirectories.
- Calculate the total sales (quantity sold) for each product across all stores and all months.
- Determine the top 5 best-selling products in terms of the total quantity sold.

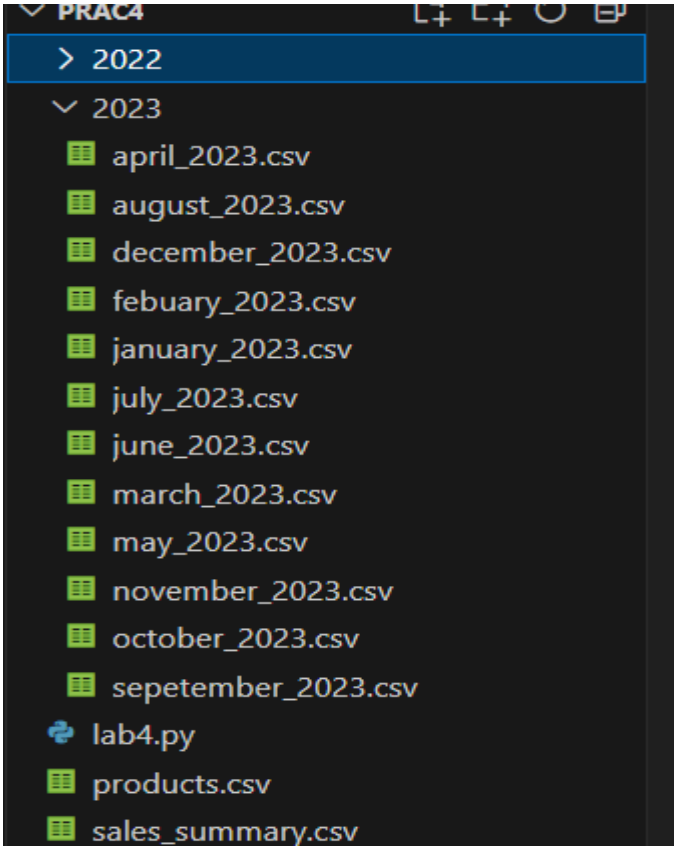
Create a new CSV file named "sales_summary.csv" and write the following information into it:

- Product ID
- Product Name
- Total Quantity Sold
- Average Quantity Sold per month (considering all months available in the data)

Input files:
products.csv

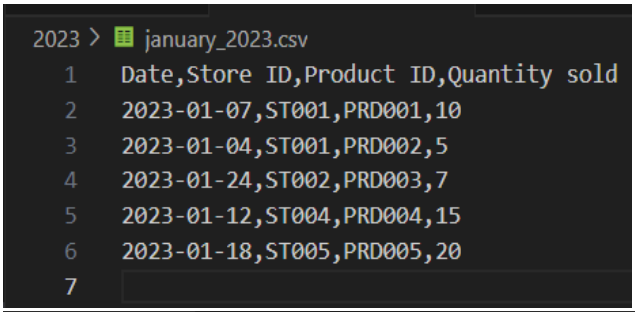


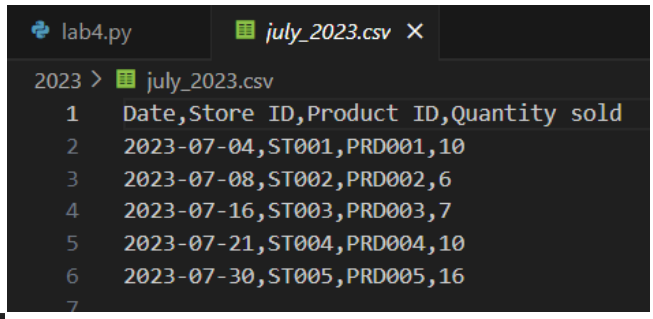


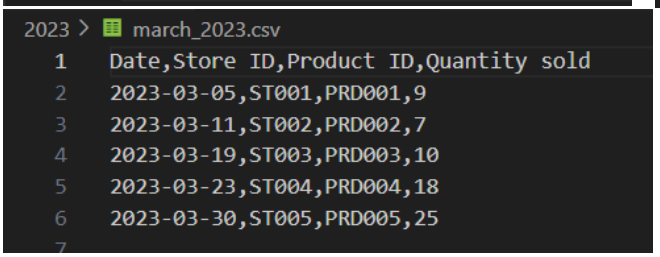


Here, I have created folder 2022,2023 folder in which I have listed for all the months.

2023 data:







Like this, I have created for all the months for 2023 year.

Same for this, I have created for 2022 data:

lab4.py December_2022.csv

2022 > December_2022.csv

```
Date,Store ID,Product ID,Quantity sold
2022-12-04,ST001,PRD001,12
2022-12-11,ST002,PRD002,9
2022-12-17,ST003,PRD003,11
2022-12-22,ST004,PRD004,14
2022-12-30,ST005,PRD005,15
```

lab4.py September_2022.csv

2022 > September_2022.csv

```
Date,Store ID,Product ID,Quantity sold
2022-09-02,ST001,PRD001,15
2022-09-07,ST002,PRD002,12
2022-09-16,ST003,PRD003,6
2022-09-21,ST004,PRD004,13
2022-09-29,ST005,PRD005,18
```

Source code:

```
import os
import pandas as pd

# Debugging lines
print("Current working directory:", os.getcwd())
print("Directory contents:", os.listdir())

# Function to read all sales data from CSV files in a directory and its subdirectories
def read_sales_data(directory):
    all_data = []
    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith('.csv'):
                file_path = os.path.join(root, file)
                data = pd.read_csv(file_path)
                all_data.append(data)
    return pd.concat(all_data, ignore_index=True)

# Function to calculate total and average sales per product
def calculate_sales_summary(sales_data, product_names):
    try:
        # Group by Product ID and calculate total quantity sold
        total_sales = sales_data.groupby('Product ID')['Quantity sold'].sum().reset_index()

        # Merge with product names
        summary = pd.merge(total_sales, product_names, on='Product ID', how='left')

        # Convert 'Date' column to datetime format if it exists
        if 'Date' in sales_data.columns:
            sales_data['Date'] = pd.to_datetime(sales_data['Date'])
            # Calculate the average quantity sold per month
            total_months = sales_data['Date'].dt.to_period('M').nunique()  # Unique months
            summary['Average Quantity Sold per Month'] = summary['Quantity sold'] / total_months
        else:
            summary['Average Quantity Sold per Month'] = summary['Quantity sold']  # No date info
available

        return summary
    except Exception as e:
        print(f"Error in calculating sales summary: {e}")
        return pd.DataFrame()  # Return an empty DataFrame on error
```

```
# Function to determine the top 5 best-selling products
def get_top_selling_products(sales_summary):
    return sales_summary.sort_values(by='Quantity sold', ascending=False).head(5)

# Main function to execute the operations
def main():
    try:
        # Set the directories to '2011' and '2012' within 'prac4'
        base_directory = os.getcwd()
        directories = [os.path.join(base_directory, '2023'), os.path.join(base_directory, '2022')]

        # Debugging lines to check paths
        print("Current working directory:", base_directory)
        for directory in directories:
            print(f"Checking directory {directory}")
            print("Files in the directory:", os.listdir(directory))

        # Read the product names file
        product_names = pd.read_csv('products.csv')

        # Read sales data from both directories
        all_sales_data = []
        for directory in directories:
            sales_data = read_sales_data(directory)
            all_sales_data.append(sales_data)

        # Combine sales data from both folders
        combined_sales_data = pd.concat(all_sales_data, ignore_index=True)

        # Calculate the sales summary
        sales_summary = calculate_sales_summary(combined_sales_data, product_names)

        # Get the top 5 best-selling products
        top_selling_products = get_top_selling_products(sales_summary)

        # Save the sales summary to a CSV file
        top_selling_products.to_csv('sales_summary.csv', index=False)

        # Print the top 5 best-selling products
        print("\nTop 5 Best-Selling Products:")
        print(top_selling_products[['Product ID', 'Product Name', 'Quantity sold', 'Average Quantity Sold per
Month']].to_string(index=False))

        print("\nSales summary has been saved to 'sales_summary.csv'.")
         print("Name: Honey Patel")
         print("Roll number: 22BCP402")

    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

Output:

```
PS C:\Users\lenovo\Desktop\python_lab\prac4> & "C:/Program Files/Python313/python.exe" c:/Users/lenovo/Desktop/python_lab/prac4/lab4.py
Current working directory: C:\Users\lenovo\Desktop\python_lab\prac4
Directory contents: ['2022', '2023', 'lab4.py', 'products.csv', 'sales_summary.csv']
Current working directory: C:\Users\lenovo\Desktop\python_lab\prac4
Checking directory C:\Users\lenovo\Desktop\python_lab\prac4\2023
Files in the directory: ['april_2023.csv', 'august_2023.csv', 'december_2023.csv', 'febuary_2023.csv', 'january_2023.csv', 'july_2023.csv', 'june_2023.csv', 'march_20
23.csv', 'may_2023.csv', 'november_2023.csv', 'october_2023.csv', 'sepetember_2023.csv']
Checking directory C:\Users\lenovo\Desktop\python_lab\prac4\2022
Files in the directory: ['April_2022.csv', 'August_2022.csv', 'December_2022.csv', 'febuary_2022.csv', 'january_2022.csv', 'July_2022.csv', 'June_2022.csv', 'March_20
22.csv', 'May_2022.csv', 'November_2022.csv', 'October_2022.csv', 'September_2022.csv']

Top 5 Best-Selling Products:
Product ID Product Name  Quantity sold  Average Quantity Sold per Month
    PRD005       Camera            371                       15.458333
    PRD004      Printer            300                       12.500000
    PRD001       Laptop            275                       11.458333
    PRD002   Smartphone            210                        8.750000
    PRD003    Headphone            206                        8.583333

Sales summary has been saved to 'sales_summary.csv'.
Name: Honey Patel
Roll number: 22BCP402
PS C:\Users\lenovo\Desktop\python_lab\prac4>
```

Sales_summary has been created in folder:

lab4.py | sales_summary.csv

sales_summary.csv

```
Product ID,Quantity sold,Product Name,Average Quantity Sold per Month
PRD005,371,Camera,15.458333333333334
PRD004,300,Printer,12.5
PRD001,275,Laptop,11.458333333333334
PRD002,210,Smartphone,8.75
PRD003,206,Headphone,8.583333333333334
```

Lab Assignment 5 : JSON Handling

You are working as a data scientist for a healthcare organization, and your team has been tasked with analyzing COVID-19 data from multiple countries. The data is stored in JSON files, with each file representing the daily COVID-19 statistics for a specific country. Each JSON file has the following structure:

{ "country": "Country Name", "date":

"YYYY-MM-DD",

"confirmed_cases": { "total": 1000, "new": 50 },

"deaths": { "total": 20, "new": 2 },

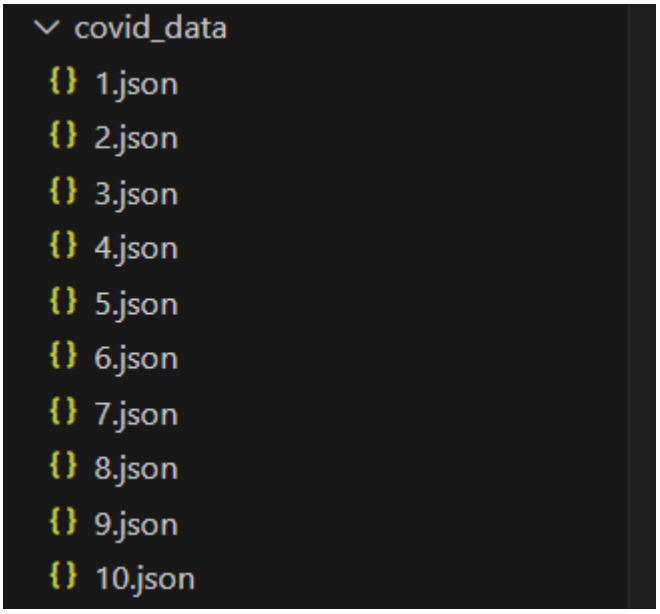
"recovered": { "total": 800, "new": 30 }

}

Your task is to write a Python program that performs the following operations:

1. Read COVID-19 data from all JSON files in a given directory and its subdirectories.
2. Calculate and display the following statistics for each country:
    1. Total confirmed cases.
    2. Total deaths.
    3. Total recovered cases.
    4. Total active cases (total confirmed cases minus total deaths and total recovered).
3. Determine the top 5 countries with the highest number of confirmed cases and the lowest number of confirmed cases.
4. Generate a summary report in JSON format that includes the statistics for all countries and save it to a file named "covid19_summary.json".
5.

Input files: I have created covid_data folder in which writing all the json file.



covid_data > {} 1.json > ...

```
{
    "country": "Russia",
    "date": "2024-09-18",
    "confirmed_cases": {"total": 60876, "new": 1305 },
    "deaths": {"total": 2958,"new": 59},
    "recovered": {"total": 55324,"new": 1223 }
}
```

covid.py {} 2.json

covid_data > {} 2.json > {} recovered

```
{
    "country": "Philippines",
    "date": "2024-09-03",
    "confirmed_cases": {
        "total": 14215,
        "new": 615
    },
    "deaths": {
        "total": 1892,
        "new": 6
    },
    "recovered": {
        "total": 9775,
        "new": 506
    }
}
```

covid.py {} 3.json

covid_data > {} 3.json > ...

```
{
    "country": "Indonesia",
    "date": "2024-09-15",
    "confirmed_cases": {
        "total": 93006,
        "new": 1091
    },
    "deaths": {
        "total": 2730,
        "new": 93
    },
    "recovered": {
        "total": 57038,
        "new": 587
    }
}
```

covid.py {} 4.json

covid_data > {} 4.json > ...

```
{
    "country": "Japan",
    "date": "2024-09-20",
    "confirmed_cases": {
        "total": 81989,
        "new": 666
    },
    "deaths": {
        "total": 1535,
        "new": 90
    },
    "recovered": {
        "total": 19260,
        "new": 516
    }
}
```

covid.py {} 5.json

covid_data > {} 5.json > {} recovered

```

{
    "country": "Vietnam",
    "date": "2024-09-16",
    "confirmed_cases": {
        "total": 17585,
        "new": 1465
    },
    "deaths": {
        "total": 3991,
        "new": 16
    },
    "recovered": {
        "total": 8987,
        "new": 976
    }
}
```

Like this, I have created 10 json file.

Source code:

```
import os
import json

# Step 1: Load all JSON files from a directory and subdirectories
def load_covid_data(data_directory):
    covid_data = []
    for root, dirs, files in os.walk(data_directory):
        for file in files:
            if file.endswith(".json"):
                file_path = os.path.join(root, file)
                try:
                    with open(file_path, 'r') as f:
                        data = json.load(f)
                        covid_data.append(data)
                except json.JSONDecodeError:
                    print(f"Error decoding JSON in file: {file_path}")
                except Exception as e:
                    print(f"Error loading file {file_path}: {e}")
    return covid_data

# Step 2: Calculate summary statistics for each country
def calculate_statistics(covid_data):
    country_stats = {}

    for data in covid_data:
        # Ensure the 'country' key exists
        if 'country' not in data:
            print(f"Missing 'country' key in data: {data}")  # Print data with missing key
            continue  # Skip this entry if 'country' is missing

        country = data['country']
        if country not in country_stats:
            country_stats[country] = {
                'total_confirmed': 0,
                'total_deaths': 0,
                'total_recovered': 0
            }

        # Access nested data and handle missing fields gracefully
        confirmed = data.get('confirmed_cases', {}).get('total', 0)
        deaths = data.get('deaths', {}).get('total', 0)
        recovered = data.get('recovered', {}).get('total', 0)
```

```
        country_stats[country]['total_confirmed'] += confirmed
        country_stats[country]['total_deaths'] += deaths
        country_stats[country]['total_recovered'] += recovered

    # Calculate total active cases for each country
    for country, stats in country_stats.items():
        stats['total_active'] = stats['total_confirmed'] - stats['total_deaths'] -
stats['total_recovered']

    return country_stats

# Step 3: Determine the top 5 countries with highest and lowest confirmed cases
def get_top_countries(country_stats):
    sorted_countries = sorted(country_stats.items(), key=lambda x: x[1]['total_confirmed'],
reverse=True)

    top_5_highest = sorted_countries[:5]
    top_5_lowest = sorted_countries[-5:]

    return top_5_highest, top_5_lowest

# Step 4: Generate the summary report and save it to a JSON file
def generate_summary_report(country_stats, top_5_highest, top_5_lowest,
output_filename="covid19_summary.json"):
    summary = {
        "COVID-19 Statistics by Country": [
            f"{country}: Confirmed: {stats['total_confirmed']}, Deaths: {stats['total_deaths']},
Recovered: {stats['total_recovered']}, Active: {stats['total_active']}"
            for country, stats in country_stats.items()
        ],
        "Top 5 Countries with Highest Confirmed Cases": [
            f"{country}: {stats['total_confirmed']}" for country, stats in top_5_highest
        ],
        "Top 5 Countries with Lowest Confirmed Cases": [
            f"{country}: {stats['total_confirmed']}" for country, stats in top_5_lowest
        ]
    }

    with open(output_filename, 'w') as f:
        json.dump(summary, f, indent=4)

# Step 5: Print top 5 countries with highest and lowest confirmed cases
def print_top_countries(top_5_highest, top_5_lowest):
    print("\nTop 5 Countries with Highest Confirmed Cases:")
    for country, stats in top_5_highest:
        print(f"{country}: {stats['total_confirmed']}")
```

```
    print("\nTop 5 Countries with Lowest Confirmed Cases:")
    for country, stats in top_5_lowest:
        print(f"{country}: {stats['total_confirmed']}")

# Step 6: Main function to run the entire process
def main(data_directory):
    try:
        covid_data = load_covid_data(data_directory)
        country_stats = calculate_statistics(covid_data)
        top_5_highest, top_5_lowest = get_top_countries(country_stats)

        # Generate summary file
        generate_summary_report(country_stats, top_5_highest, top_5_lowest)

        # Print top 5 countries with highest and lowest confirmed cases

        print("\nPatel Honey Ashishkumar")
        print("\n22BCP402")

        print_top_countries(top_5_highest, top_5_lowest)

        print("\nSummary file 'covid19_summary.json' has been generated successfully!")
    except Exception as e:
        print(f"An error occurred: {e}")

# Set the data directory to the path where the JSON files are located
data_directory = "C:\\Users\\lenovo\\Desktop\\python_lab"

# Run the main function
main(data_directory)
```

Output:

```
Patel Honey Ashishkumar

22BCP402

Top 5 Countries with Highest Confirmed Cases:
Bangladesh: 145468
Japan: 137725
Vietnam: 114336
Indonesia: 113355
Russia: 60876

Top 5 Countries with Lowest Confirmed Cases:
Japan: 137725
Vietnam: 114336
Indonesia: 113355
Russia: 60876
Philippines: 14215

Summary file 'covid19_summary.json' has been generated successfully!
PS C:\Users\lenovo\Desktop\python_lab>
```

Covid19_summary.json:

covid.py | covid19_summary.json

covid19_summary.json > [ ] Top 5 Countries with Highest Confirmed Cases


```
{
    "COVID-19 Statistics by Country": [
        "Russia: Confirmed: 60876, Deaths: 2958, Recovered: 55324, Active: 2594",
        "Indonesia: Confirmed: 113355, Deaths: 5912, Recovered: 72947, Active: 34496",
        "Philippines: Confirmed: 14215, Deaths: 1892, Recovered: 9775, Active: 2548",
        "Japan: Confirmed: 137725, Deaths: 1755, Recovered: 35543, Active: 100427",
        "Vietnam: Confirmed: 114336, Deaths: 7751, Recovered: 45183, Active: 61402",
        "Bangladesh: Confirmed: 145468, Deaths: 5481, Recovered: 59398, Active: 80589"
    ],
    "Top 5 Countries with Highest Confirmed Cases": [
        "Bangladesh: 145468",
        "Japan: 137725",
        "Vietnam: 114336",
        "Indonesia: 113355",
        "Russia: 60876"
    ],
    "Top 5 Countries with Lowest Confirmed Cases": [
        "Japan: 137725",
        "Vietnam: 114336",
        "Indonesia: 113355",
        "Russia: 60876",
        "Philippines: 14215"
    ]
}
```

Lab Assignment 6: Error and Exception Handling

You are working on a project to build a custom text processing tool that reads input from various sources, processes the text data, and stores the results in an output file. As part of this project, you need to implement a robust exception handling mechanism to handle potential errors that may arise during the text processing.

- The tool needs to perform the following steps:

1. Read the input data from a file specified by the user.
2. Process the text data by performing various operations, such as counting words, calculating character frequencies, and generating word clouds.
3. Store the processed results in an output file.

Your task is to design a Python program that incorporates appropriate exception handling to handle the following situations:

1. File Not Found Error: If the user provides an invalid file path or the input file is not found, your program should raise a custom exception FileNotFoundError with a suitable error message.
2. Invalid Input Data: During text processing, if any unexpected input data is encountered (e.g., non-string values or missing data), your program should raise a custom exception InvalidInputDataError with relevant details.
3. Disk Space Full: If the output file cannot be written due to insufficient disk space, your program should raise a custom exception DiskSpaceFullError

Additionally, the program should have the following features:

- The custom exception classes should inherit from the base Exception class and provide meaningful error messages.
- Proper logging should be implemented to capture details about the exceptions that occur during text processing.
- The program should provide a user-friendly interface, allowing the user to enter the input file path and choose the desired text processing operations.

The processed results should be stored in an output file with a suitable format (e.g., JSON, CSV, or plain text).

Source code:

```
import os
import shutil

# Custom exceptions
class FileNotFoundError(Exception):
    def __init__(self, message="Input file not found. Please check the file path and try again."):
        self.message = message
        super().__init__(self.message)

class InvalidInputDataError(Exception):
    def __init__(self, message="Invalid input data encountered during processing."):
        self.message = message
        super().__init__(self.message)

class DiskSpaceFullError(Exception):
    def __init__(self, message="Insufficient disk space to write the output file."):
        self.message = message
        super().__init__(self.message)
```

```
# Function to read input from a file
def read_input(file_path):
    try:
        if not os.path.exists(file_path):
            raise FileNotFoundError
        with open(file_path, 'r') as file:
            data = file.read()
            if not isinstance(data, str) or len(data.strip()) == 0:
                raise InvalidInputDataError
            return data
    except FileNotFoundError as e:
        print(f"Error: {e}")
        return None
    except InvalidInputDataError as e:
        print(f"Error: {e}")
        return None

# Function to process the text data
def process_text(data):
    try:
        word_count = len(data.split())
        char_frequencies = {char: data.count(char) for char in set(data)}
        return word_count, char_frequencies
    except Exception as e:
        print(f"Error during processing: {e}")
        raise InvalidInputDataError

# Function to check disk space and write the results to an output file
def write_output(output_file, word_count, char_frequencies):
    try:
        # Check disk space using shutil.disk_usage()
        total, used, free = shutil.disk_usage(os.path.dirname(output_file))


        if free < 1024:  # Check if at least 1 KB is free
            raise DiskSpaceFullError

        with open(output_file, 'w') as file:
            file.write(f"Word Count: {word_count}\n")
            file.write("Character Frequencies:\n")
            for char, freq in char_frequencies.items():
                file.write(f"{char}: {freq}\n")
    except DiskSpaceFullError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"Unexpected error during file write: {e}")

# Main program flow
def main():
    input_file = input("Enter the input file path: ")
    output_file = input("Enter the output file path: ")
```

```
    data = read_input(input_file)

    if data:
        try:
            word_count, char_frequencies = process_text(data)
            write_output(output_file, word_count, char_frequencies)
            print("Nmae:Patel Honey Ashishkumar")
            print("Roll number: 22BCP402")
            print("Processing complete and results saved to the output file.")
        except InvalidInputDataError as e:
            print(f"Error: {e}")
        except DiskSpaceFullError as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

In this I have given the input file path , file named check.txt.

```
check.txt
1   Hi i am Honey Patel.i am B.tech Third year student.i am persuing CSE. i like drawing and travelling.
```

It will genrate check_summary.txt:

check_summary.txt

```
Word Count: 22
Character Frequencies:
b: 1
4: 1
p: 1
P: 2
v: 1
m: 4
S: 1
n: 7
E: 1
H: 2
C: 2
i: 11
T: 1
g: 3
k: 1
0: 1
B: 2
h: 2
d: 4
.: 6
y: 3
t: 5
e: 9
a: 8
 : 21
c: 1
l: 6
u: 3
M: 1
s: 3
w: 1
r: 7
2: 3
o: 2
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/
Enter the input file path: C:\Users\lenovo\Desktop\python_lab\check.txt
Enter the output file path: C:\Users\lenovo\Desktop\python_lab\check_summary.txt
Nmae:Patel Honey Ashishkumar
Roll number: 22BCP402
Processing complete and results saved to the output file.
PS C:\Users\lenovo\Desktop\python_lab>
```

Lab Assignment 7 : Pdf reading creation and editing

You are working for a company that sells products online. Your task is to develop a Python program that reads order data from a CSV file, generates individual PDF invoices for each order, and then merges all the PDF invoices into a single PDF file.

1. Load Order Data: The program should read order data from a CSV file named "orders.csv." Each row in the CSV file represents an order with the following information:
    1. Order ID (a unique alphanumeric code)
    2. Customer Name
    3. Product Name
    4. Quantity
    5. Unit Price
2. Calculate Total Amount: For each order, calculate the total amount by multiplying the quantity with the unit price.

Generate PDF Invoices: Create individual PDF invoices for each order. Each invoice should contain the following details:

1. Invoice Number (same as the Order ID)
2. Date of Purchase (current date)
3. Customer Name
4. Product Name
5. Quantity
6. Unit Price

Total Amount

Source code:

```
import csv
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from reportlab.lib.units import mm
from datetime import datetime
from PyPDF2 import PdfMerger
import os

# Step 1: Load Order Data
def load_orders(filename):
    orders = []
    with open(filename, mode='r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            row['Quantity'] = int(row['Quantity'])
            row['Unit Price'] = float(row['Unit Price'])
            row['Total Amount'] = row['Quantity'] * row['Unit Price']
            orders.append(row)
    return orders

# Step 2: Generate PDF Invoices
def generate_pdf_invoice(order, output_directory="invoices_folder"):
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    order_id = order['Order ID']
    customer_name = order['Customer Name']
    product_name = order['Product Name']
```

```
    quantity = order['Quantity']
    unit_price = order['Unit Price']
    total_amount = order['Total Amount']
    purchase_date = datetime.now().strftime("%Y-%m-%d")

    # Create PDF
    pdf_filename = f"{output_directory}/Invoice_{order_id}.pdf"
    c = canvas.Canvas(pdf_filename, pagesize=A4)

    # Add content to PDF
    c.setFont("Helvetica", 12)

    # Invoice Header
    c.drawString(20*mm, 270*mm, f"Invoice Number: {order_id}")
    c.drawString(20*mm, 260*mm, f"Date of Purchase: {purchase_date}")

    # Customer Information
    c.drawString(20*mm, 240*mm, f"Customer Name: {customer_name}")

    # Order Details
    c.drawString(20*mm, 220*mm, f"Product Name: {product_name}")
    c.drawString(20*mm, 210*mm, f"Quantity: {quantity}")
    c.drawString(20*mm, 200*mm, f"Unit Price: ${unit_price:.2f}")
    c.drawString(20*mm, 190*mm, f"Total Amount: ${total_amount:.2f}")

    # Save PDF
    c.showPage()
    c.save()

    return pdf_filename

# Step 3: Merge PDFs into a Single File
def merge_pdfs(pdf_files, output_filename="Merged_Invoices.pdf"):
    merger = PdfMerger()

    for pdf in pdf_files:
        merger.append(pdf)

    # Write to final output PDF file
    merger.write(output_filename)
    merger.close()

# Main function to run the process
def create_invoices(csv_filename):
    orders = load_orders(csv_filename)
    pdf_files = []

    # Generate individual PDFs
    for order in orders:
        pdf_file = generate_pdf_invoice(order)
        pdf_files.append(pdf_file)

    # Merge all PDFs
```

```
    merge_pdfs(pdf_files)

    print(f"All invoices merged into 'Merged_Invoices.pdf'.")
    print("\nName: Patel Honey Ashishkumar")
    print("\nRoll Number: 22BCP402")

# Usage
csv_filename = "orders.csv"
create_invoices(csv_filename)
```

Input file: orders.csv

```
orders.csv
1   Order ID,Customer Name,Product Name,Quantity,Unit Price
2   ORD001,Honey Patel,Laptop,2,799.99
3   ORD002,Saanvi Patel,Smartphone,1,699.50
4   ORD003,Keni Tandel,Headphones,3,199.99
5   ORD004,Aksh Patel,Monitor,1,299.99
6   ORD005,Niya Patel,Keyboard,5,49.99
7
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.e
All invoices merged into 'Merged_Invoices.pdf'.

Name: Patel Honey Ashishkumar

Roll Number: 22BCP402
PS C:\Users\lenovo\Desktop\python_lab>
```

One folder generated in this all the pdf files are created:

covid_data	26-10-2024 21:46	File folder
invoices_folder	13-11-2024 20:54	File folder

Name	Date modified	Type	Size
Invoice_ORD001	13-11-2024 20:58	Microsoft Edge PD...	2 KB
Invoice_ORD002	13-11-2024 20:58	Microsoft Edge PD...	2 KB
Invoice_ORD003	13-11-2024 20:58	Microsoft Edge PD...	2 KB
Invoice_ORD004	13-11-2024 20:58	Microsoft Edge PD...	2 KB
Invoice_ORD005	13-11-2024 20:58	Microsoft Edge PD...	2 KB

All the pdf files:

Invoice Number: ORD001 Date of Purchase: 2024-11-13 Customer Name: Honey Patel Product Name: Laptop Quantity: 2 Unit Price: $799.99 Total Amount: $1599.98	Invoice Number: ORD002 Date of Purchase: 2024-11-13 Customer Name: Saanvi Patel Product Name: Smartphone Quantity: 1 Unit Price: $699.50 Total Amount: $699.50
Invoice Number: ORD003 Date of Purchase: 2024-11-13 Customer Name: Keni Tandel Product Name: Headphones Quantity: 3 Unit Price: $199.99 Total Amount: $599.97	Invoice Number: ORD004 Date of Purchase: 2024-11-13 Customer Name: Aksh Patel Product Name: Monitor Quantity: 1 Unit Price: $299.99 Total Amount: $299.99

Invoice Number: ORD005 Date of Purchase: 2024-11-13 Customer Name: Niya Patel Product Name: Keyboard Quantity: 5 Unit Price: $49.99 Total Amount: $249.95	

Merged file:

Invoice Number: ORD001
Date of Purchase: 2024-11-13

Customer Name: Honey Patel

Product Name: Laptop
Quantity: 2
Unit Price: $799.99
Total Amount: $1599.99

Invoice Number: ORD002
Date of Purchase: 2024-11-13

Customer Name: Sneevi Patel

Product Name: Smartphone
Quantity: 1
Unit Price: $699.50
Total Amount: $699.50

Invoice Number: ORD003
Date of Purchase: 2024-11-13

Customer Name: Keni Tandel

Product Name: Headphones
Quantity: 3
Unit Price: $199.99
Total Amount: $599.97

4 of 5

Invoice Number: ORD004
Date of Purchase: 2024-11-13

Customer Name: Aksh Patel

Product Name: Monitor
Quantity: 1
Unit Price: $299.99
Total Amount: $299.99

Invoice Number: ORD005
Date of Purchase: 2024-11-13

Customer Name: Niya Patel

Product Name: Keyboard
Quantity: 5
Unit Price: $49.99
Total Amount: $249.95

Lab Assignment 8 : User Management Automation

You are developing a command-line task management system for a small team of users.

User Management:

- Implement a user registration system where users can sign up and log in. Store user data in a file, including usernames and hashed passwords.

Source code:

```
import bcrypt
import os

# File to store user data
USER_FILE = 'users.txt'

# Function to register a new user
def register_user():
    username = input("Enter a unique username: ")

    # Check if the username already exists
    if username_exists(username):
        print("Username already exists. Please choose another one.")
        return

    password = input("Enter a password: ").encode('utf-8')

    # Hash the password with bcrypt
    hashed_password = bcrypt.hashpw(password, bcrypt.gensalt())

    # Store the username and hashed password in the file
    with open(USER_FILE, 'a') as file:
        file.write(f"{username},{hashed_password.decode('utf-8')}\n")

    print("Registration successful!")

# Function to check if a username exists in the file
def username_exists(username):
    if not os.path.exists(USER_FILE):
        return False

    with open(USER_FILE, 'r') as file:
        for line in file:
            stored_username, _ = line.strip().split(',')
            if stored_username == username:
                return True
    return False

# Function to log in a user
def login_user():
    username = input("Enter your username: ")
```

```
    password = input("Enter your password: ").encode('utf-8')

    # Check if the username exists and get the stored hash
    stored_hash = get_stored_hash(username)
    if not stored_hash:
        print("Username not found.")
        return

    # Verify the password against the stored hash
    if bcrypt.checkpw(password, stored_hash.encode('utf-8')):
        print("Login successful!")
    else:
        print("Incorrect password.")

# Function to get the stored hash for a username
def get_stored_hash(username):
    if not os.path.exists(USER_FILE):
        return None

    with open(USER_FILE, 'r') as file:
        for line in file:
            stored_username, stored_hash = line.strip().split(',')
            if stored_username == username:
                return stored_hash
    return None

# Main function to drive the program
def main():
    while True:
        print("\nName:Patel Honey Ashishkumar")
        print("\nRoll number:22BCP402")
        print("\nUser Management System")
        print("1. Register")
        print("2. Login")
        print("3. Exit")
        choice = input("Choose an option: ")

        if choice == '1':
            register_user()
        elif choice == '2':
            login_user()
        elif choice == '3':
            print("Exiting the program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

Output:

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.e

Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option: 1
Enter a unique username: Keni tandel
Enter a password: keni183
Registration successful!

Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option:
```

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/pytho

Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option: 2
Enter your username: Keni tandel
Enter your password: keni183
Login successful!

Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option:
```

```
Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option: 2
Enter your username: Niya patel
Enter your password: niya456
Username not found.

Name:Patel Honey Ashishkumar

Roll number:22BCP402

User Management System
1. Register
2. Login
3. Exit
Choose an option: 3
Exiting the program.
PS C:\Users\lenovo\Desktop\python_lab>
```

users.txt file generated:

```
users.txt
honey,$2b$12$ZSVNpTZWHEGBXKqx.3dmS.0DYVR5mWEAtQPOYFHUn.Mh2qb0tCkI.
tamanna,$2b$12$V7/G1kCPUMlbCuVhrZkr9ewoPDnKTDLp2HeS0rbEwHUX4upIHij/S
Keni tandel,$2b$12$h7belFsPw00ef55KysRZ9O6ivA70K5ej4GQKGVzN7bWJyhxwbs8Ly
```

Lab Assignment 9:Image and Audio Data Processing

You are tasked with developing a comprehensive Python program that reads and manipulates both image and audio data. The goal is to create a tool that processes images and audio waveforms, allowing users to perform various operations on both types of data. This exercise aims to test your proficiency in handling different data formats and applying appropriate algorithms for manipulation.

Part 1: Image Data Processing

1. **Image Loading and Display:** Your program should allow users to load an image file and display it. Ensure you use an image processing library like Pillow (PIL) to handle image data.
2. **Image Manipulation:** Implement at least two image manipulation operations, such as:
   - Applying filters (e.g., Gaussian blur, edge detection).
   - Changing image dimensions or cropping.
   - Adjusting brightness, contrast, or saturation.
   - Converting to grayscale or other color spaces.
3. **Histogram Analysis:** Implement a feature that calculates and displays histograms for different color channels of the loaded image. Allow users to analyze and manipulate histogram data.

Source code:

```
from PIL import Image, ImageFilter, ImageOps
import matplotlib.pyplot as plt
import numpy as np

print("NAME: Patel Honey Ashishkumar")
print("ROLL NO.: 22BCP402")

def load_and_display_image(image_path):
    image = Image.open(image_path)
    image.show(title="Main Image")
    return image

def resize_image(image, size=(200, 200)):
    resized_image = image.resize(size)
    resized_image.show(title="Resized Image")
    return resized_image

def convert_to_grayscale(image):
    grayscale_image = ImageOps.grayscale(image)
    grayscale_image.show(title="Grayscale Image")
    return grayscale_image

def apply_gaussian_blur(image, radius=5):
    blurred_image = image.filter(ImageFilter.GaussianBlur(radius))
    blurred_image.show(title="Gaussian Blur")
    return blurred_image

def apply_edge_detection(image):
    edge_image = image.filter(ImageFilter.FIND_EDGES)
    edge_image.show(title="Edge Detection")
    return edge_image

def plot_histogram(image):
```

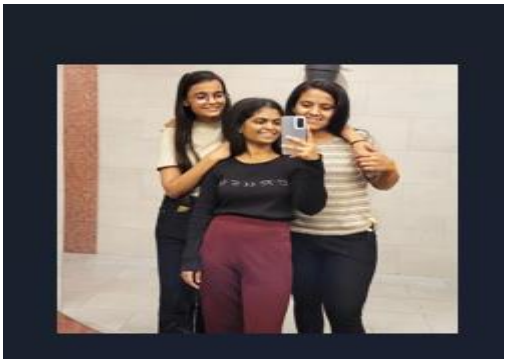
```
    plt.figure(figsize=(10, 5))
    for i, color in enumerate(['red', 'green', 'blue']):
        histogram_data = image.getchannel(i).histogram()
        plt.plot(histogram_data, color=color)
    plt.title("Color Histogram")
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.legend(['Red', 'Green', 'Blue'])
    plt.grid(True)
    plt.show()

def plot_histogram_separate_bars(image):
    channels = image.split()
    colors = ['r', 'g', 'b']
    plt.figure(figsize=(10, 6))

    for i, color in enumerate(colors):
        histogram_data = channels[i].histogram()
        plt.bar(np.arange(256), histogram_data, color=color, alpha=0.5)

    plt.title("Color Histogram (Separate Bars)")
    plt.xlabel("Color Value Ranges")
    plt.ylabel("# of Pixels")
    plt.legend(['r', 'g', 'b'])
    plt.grid(True)
    plt.show()

if __name__ == "__main__":

    image_path = 'C:\\Users\\lenovo\\Desktop\\python_lab\\hanike.jpg'
    image = load_and_display_image(image_path)

    resized_image = resize_image(image)
    grayscale_image = convert_to_grayscale(image)
    gaussian_blur_image = apply_gaussian_blur(image)
    edge_detection_image = apply_edge_detection(image)
    plot_histogram(image)
    plot_histogram_separate_bars(image)
    print("Images and plots generated successfully!")
```

Output:

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe"
NAME: Patel Honey Ashishkumar
ROLL NO.: 22BCP402
```

Main image:

Resizes image:

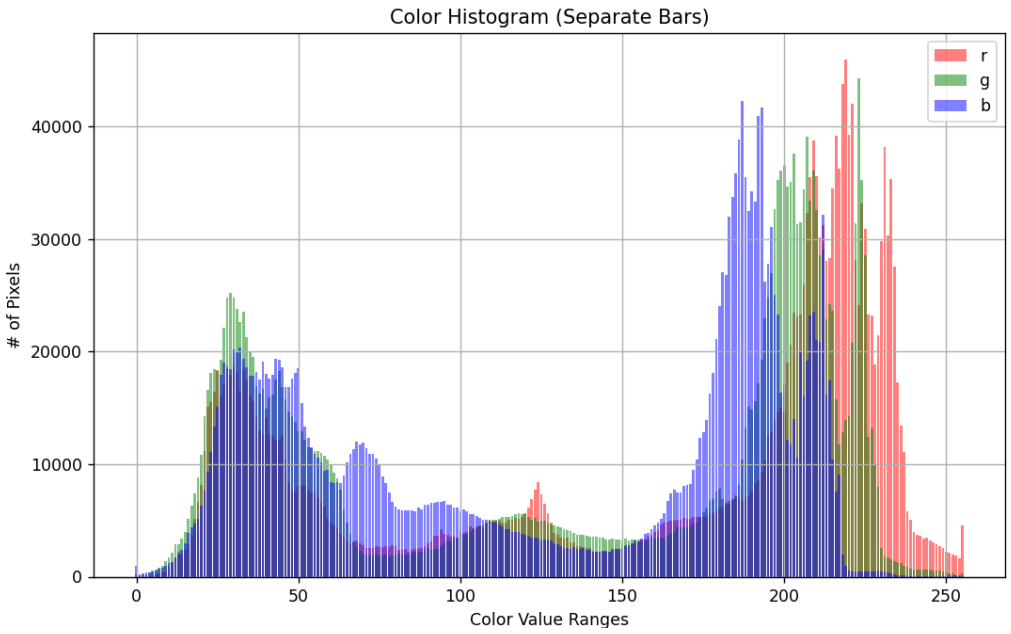


Grayscale:

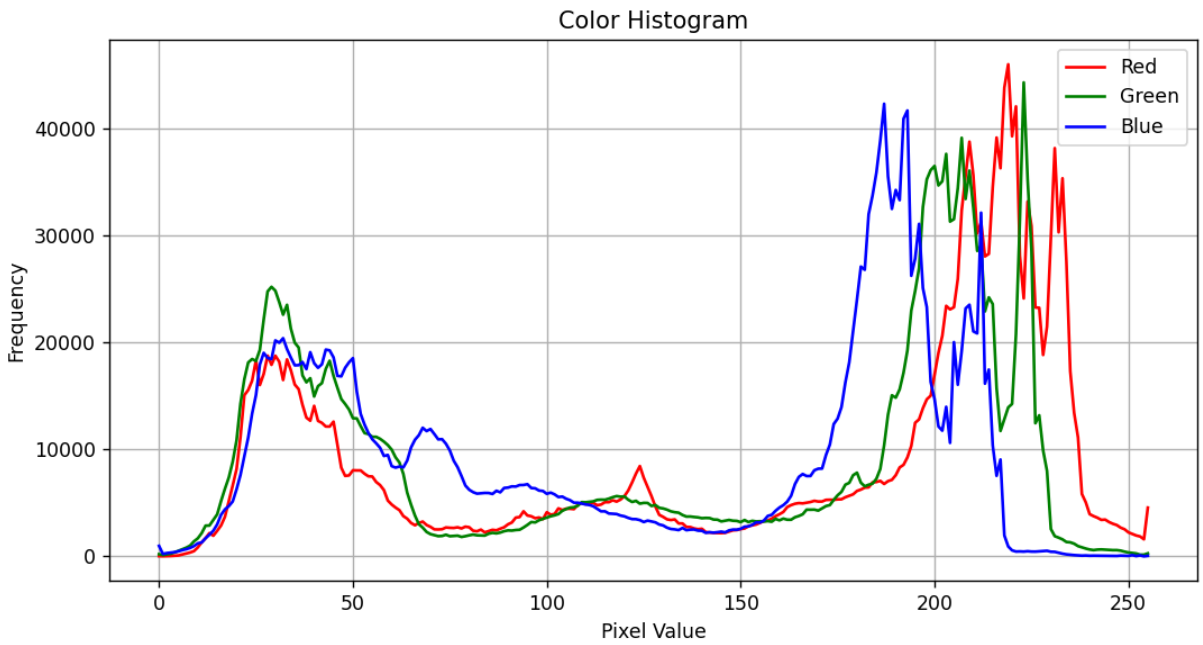
Gaussian blur:

Edge detection:

Graph:



Histogram:


Color Histogram
Red
Green
Blue
Frequency
Pixel Value
0
10000
20000
30000
40000
0
50
100
150
200
250

LAB Assignment 10: Logging engineer

Analyze your project to identify potential places for logging involves understanding the application's structure, components, and potential points of interest where capturing information would be beneficial. Here's a systematic approach to help you identify these places:

Understand the Application's Purpose and Flow: Familiarize yourself with the application's functionality and objectives. Understand the user interactions, data processing steps, and overall flow of the program.

Identify Critical Components and Functions: Identify key components, functions, or methods that play a central role in the application's operation. These might include functions responsible for user input processing, data transformation, database interactions, or external API calls.

Identify Decision Points: Look for decision points in the application where different paths or outcomes are possible. These decision points often involve conditionals (if statements, switches, etc.) that determine the application's behaviour.

Identify External Interactions: Identify any interactions with external services, APIs, databases, or files. These interactions can provide insights into the data exchange between your application and external entities.

Identify Exception Handling: Pay attention to exception handling mechanisms in the application. Whenever an exception is caught, it's often helpful to log information about the exception, its context, and potential reasons for its occurrence.

Identify Loops and Iterations: Examine loops and iterations in your application. These might involve processing multiple items or steps in a repetitive manner. Logging within loops can help track progress and the values being processed.

Identify Inputs and Outputs: Look for points where the application interacts with user inputs, configuration settings, or external data sources. Logging inputs and outputs can help track data transformations and ensure that inputs are correctly processed.

Identify Troubleshooting Points: Consider where troubleshooting or debugging might be necessary in the future. These might be areas prone to errors or complex logic that might require detailed inspection.

Identify User Actions: If the application involves user interactions, consider logging user actions or events that help you understand how users are interacting with the software.

Consider Performance Monitoring: If performance is a concern, consider logging timing information to analyze the execution time of different components and identify potential bottlenecks.

Consult Documentation and Comments: Review any existing documentation, comments, or architectural diagrams that provide insights into the application's structure and behavior.
Brainstorm with Stakeholders: Discuss potential logging points with other developers, stakeholders, or users of the application. They might provide valuable insights into where logging would be most beneficial.
Think Like a Debugger: Put yourself in the shoes of someone who needs to debug the application. Where would you look for information to understand why something went wrong or to verify that everything is working as expected?

Once you've identified potential places for logging, you can strategically insert logging statements at these points. Remember to vary the logging levels (e.g., DEBUG, INFO, WARNING, ERROR, CRITICAL) based on the importance of the information being logged. Regularly reviewing and adjusting your logging strategy as the application evolves is crucial for maintaining effective and relevant logs.

Task- Find the potential places of logging write modules potential places in each module where you need of logging. Create a dummy code for your project. Every member in the project will select minimum two function and implement dummy code of that function with logging implementation.

Shopping Management System

Critical Components and Functions:

1. **load_inventory_data and load_orders_data**:
    - These functions load the inventory and order data from CSV files into pandas DataFrames.
    - They ensure the correct data is loaded for processing orders.
2. **process_orders**:
    - This function processes orders by checking product availability and stock levels.
    - It updates the inventory stock based on the orders, calculates total costs, and generates order confirmations.
3. **generate_reports**:
    - This function generates two key reports: the inventory details and total revenue earned per customer.
    - These reports help track the status of inventory and assess customer transactions.

Identifying Decision Points:

1. **process_orders**:
    - **Decision Points**:
        - **Valid Product ID**: Ensures the product exists in the inventory before processing an order.
        - **Available Stock**: Verifies if there is enough stock available for the order. If not, the order is not processed.
        - **Order Confirmation**: Once an order is validated, it is either confirmed or denied based on available stock.
2. **File Paths in main()**:
    - **Decision Points**:
        - Determines if the file paths for inventory and orders are valid. If a file is missing or corrupt, the program cannot proceed.

External Interactions:

1. **File Reading/Writing**:
    - **load_inventory_data** and **load_orders_data** read data from CSV files.
    - The program writes the updated inventory data back to the file after processing orders.
2. **Logging**:
    - Logs can be added to confirm whether the files were loaded and saved correctly, and if errors such as missing or invalid files occur.

Identifying Exceptional Handling:

1. **File I/O Operations**:
    - Add explicit exception handling for cases like missing files or corrupted data when reading inventory and order CSV files.
    - Example: FileNotFoundError can be raised if the files are not found.
2. **Data Validation**:
    - Within **process_orders**, handle invalid input data (e.g., non-integer ticket counts or invalid product IDs) gracefully by logging errors and skipping problematic records.
3. **Inventory Update Failures**:
    - When updating stock in **process_orders**, check for issues like insufficient permissions to write to the file or any unforeseen issues in the DataFrame manipulation.

Identifying Loops and Iteration:

1. **process_orders**:
    - The function iterates over each row in the orders DataFrame to process orders one by one.
    - Inside the loop, the program checks product availability, updates inventory, and calculates costs.
2. **generate_reports**:
    - The function iterates over the successful orders to generate revenue reports, grouping by customer and product.

Identifying Inputs and Outputs:

1. **Inputs**:
    - File paths for inventory and order data.
    - The input data from the CSV files such as product IDs, quantities, and customer details.
2. **Outputs**:
    - Order confirmations or failure messages (printed to the console).
    - Generated reports (inventory details and revenue per customer).
    - Updated inventory data saved back to the CSV files.
    - Logging the outputs can provide a detailed record of the operations, confirming what actions were performed, and flagging any issues that occurred during processing.

Identifying Troubleshooting Points:

1. **Data Processing and Validation**:
    - **process_orders** could encounter issues if the order data is in an unexpected format or contains invalid product IDs.
    - **Logging** these validation issues will help track and resolve errors like missing products or invalid quantities.
2. **File Saving**:

- After processing orders and generating reports, ensure the updated data is saved back to the files. If the saving fails, log the error and provide details for troubleshooting.

Identifying User Actions:

1. **User Inputs**:
    - The user inputs the file paths for inventory and orders in the **main()** function. Logging these inputs can help verify if the user provided correct paths and can assist with troubleshooting file-related issues.
2. **User Actions on Reports**:
    - When generating reports, users expect detailed outputs. Logging this process can track when reports are generated and if they contain any errors or missing data.

Logging Levels and Points Where It Should Be Applied:

1. **DEBUG**:
    - **Start of each function** (e.g., inside load_inventory_data, process_orders, generate_reports) to track function execution.
    - **Inside the loops in** process_orders to trace each order being processed (e.g., validating product ID, checking stock levels, calculating total costs).
    - **Inside the loops in** generate_reports to track each report row as it is processed.
2. **INFO**:
    - When a file loads successfully in load_inventory_data and load_orders_data.
    - When an order is successfully processed and confirmed, log details like customer name, product ID, and total cost.
    - When reports are successfully generated.
3. **WARNING**:
    - Invalid file paths or missing files in **main()**.
    - Unexpected data formats or type mismatches in **process_orders**, like non-integer quantities or invalid product IDs.
4. **ERROR**:
    - Failed orders due to insufficient stock or invalid product IDs in **process_orders**.
    - Issues with file saving, such as file permission errors or invalid paths.
5. **CRITICAL**:
    - Missing or corrupted data files that prevent the system from functioning.
    - Any other critical failure that would stop the program from completing its execution (e.g., the inability to access a required file or data source).

Source code:

```
import pandas as pd
import logging

# Configure logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

# Load Inventory Data
def load_inventory_data(file_path):
    try:
        logging.debug(f"Attempting to load inventory data from {file_path}.")
        df = pd.read_csv(file_path)
        df.columns = df.columns.str.strip()
        logging.info(f"Inventory data loaded successfully from {file_path}.")
        return df
    except FileNotFoundError:
        logging.critical(f"File not found: {file_path}")
        raise
    except Exception as e:
        logging.error(f"An error occurred while loading inventory data: {e}")
        raise

# Load Orders Data
def load_orders_data(file_path):
    try:
        logging.debug(f"Attempting to load orders data from {file_path}.")
        df = pd.read_csv(file_path)
        df.columns = df.columns.str.strip()
        logging.info(f"Order data loaded successfully from {file_path}.")
        return df
    except FileNotFoundError:
        logging.critical(f"File not found: {file_path}")
        raise
    except Exception as e:
        logging.error(f"An error occurred while loading order data: {e}")
        raise

# Process Orders
def process_orders(inventory_df, orders_df):
    successful_orders = []
```

```
    logging.debug("Starting order processing.")
    for _, row in orders_df.iterrows():
        customer_name = row['Customer Name']
        product_id = row['Product ID']
        quantity_requested = row['Quantity Ordered']

        product = inventory_df[inventory_df['Product ID'] == product_id]

        if not product.empty:
            available_stock = product.iloc[0]['Available Stock']
            price_per_unit = product.iloc[0]['Price per Unit']
            logging.debug(f"Processing order for {customer_name}, Product ID: {product_id}, Quantity:
{quantity_requested}.")
            if available_stock >= quantity_requested:
                # Update stock
                inventory_df.loc[inventory_df['Product ID'] == product_id, 'Available Stock'] -=
quantity_requested

                # Calculate total cost
                total_cost = quantity_requested * price_per_unit
                successful_orders.append({
                    'Customer Name': customer_name,
                    'Product ID': product_id,
                    'Quantity Ordered': quantity_requested,
                    'Total Cost': total_cost
                })

                logging.info(f"Order confirmed for {customer_name}. Total Cost: ${total_cost}.")
            else:
                logging.warning(f"Order not confirmed for {customer_name}. Only {available_stock}
units available.")
        else:
            logging.error(f"Product ID {product_id} not found for {customer_name}.")
    logging.debug("Order processing completed.")

    # Convert successful orders to DataFrame
    successful_orders_df = pd.DataFrame(successful_orders)

    return inventory_df, successful_orders_df
```

```
# Generate Reports
def generate_reports(inventory_df, successful_orders):
    logging.debug("Generating reports.")

    # Report 1: Inventory Details
    inventory_report = inventory_df[['Product ID', 'Product Name', 'Total Stock', 'Available Stock',
'Price per Unit']]

    # Report 2: Total Revenue Earned Per Customer
    revenue_per_customer = successful_orders.groupby(['Customer Name', 'Product ID']).agg({
        'Quantity Ordered': 'sum',
        'Total Cost': 'sum'
    }).reset_index()

    logging.info("Reports generated successfully.")

    return inventory_report, revenue_per_customer

# Main program
def main():
    # Get file paths from the user
    print("\nName:Patel Honey Ashishkumar")
    print("\nRoll Number:22BCP402")
    inventory_file = input("\nEnter the path to the inventory CSV file: ")
    orders_file = input("\nEnter the path to the orders CSV file: ")

    # Load data
    logging.debug(f"Loading data from {inventory_file} and {orders_file}.")
    inventory_df = load_inventory_data(inventory_file)
    orders_df = load_orders_data(orders_file)

    # Process orders
    inventory_df, successful_orders = process_orders(inventory_df, orders_df)

    # Print order confirmations
    print("\nOrder Confirmations:")
    for _, row in successful_orders.iterrows():
        print(f"Order confirmed for {row['Customer Name']}. Total Cost: ${row['Total Cost']:.2f}.")
```

```
    # Generate reports
    inventory_report, revenue_per_customer = generate_reports(inventory_df, successful_orders)

    print("\nREPORT 1: INVENTORY DETAILS:")
    print(inventory_report.to_string(index=False))

    print("\nREPORT 2: TOTAL REVENUE EARNED PER CUSTOMER:")
    print(revenue_per_customer.to_string(index=False))

    # Save updated inventory data
    inventory_df.to_csv(inventory_file, index=False)
    logging.info(f"Updated inventory data saved successfully to {inventory_file}.")

if __name__ == "__main__":
    main()
```

Input file: inventory.csv

inventory.csv

```
Product ID,Product Name,Total Stock,Available Stock,Price per Unit
PRD001,Laptop,100,92,800.0
PRD002,Smartphone,200,146,500.0
PRD003,Headphone,50,20,150.0
PRD004,Printer,30,26,200.0
PRD005,Camera,20,3,600.0
PRD006,Smartwatch,120,92,250.0
PRD007,Keyboard,150,80,50.0
PRD008,Mouse,300,190,30.0
```

Products.csv

products.csv

```
Customer Name,Product ID,Quantity Ordered
Honey,PRD001,2
Niya,PRD002,1
Keni,PRD003,5
ishan,PRD004,1
Shubman,PRD005,3
Aksh,PRD006,2
Dhruv,PRD007,10
Saanvi,PRD008,15
```

Output:

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/Users/lenovo/Des

Name:Patel Honey Ashishkumar

Roll Number:22BCP402

Enter the path to the inventory CSV file: inventory.csv

Enter the path to the orders CSV file: products.csv
2024-11-13 22:34:01,750 - DEBUG - Loading data from inventory.csv and products.csv.
2024-11-13 22:34:01,750 - DEBUG - Attempting to load inventory data from inventory.csv.
2024-11-13 22:34:01,755 - INFO - Inventory data loaded successfully from inventory.csv.
2024-11-13 22:34:01,755 - DEBUG - Attempting to load orders data from products.csv.
2024-11-13 22:34:01,757 - INFO - Order data loaded successfully from products.csv.
2024-11-13 22:34:01,757 - DEBUG - Starting order processing.
2024-11-13 22:34:01,759 - DEBUG - Processing order for Honey, Product ID: PRD001, Quantity: 2.
2024-11-13 22:34:01,760 - INFO - Order confirmed for Honey. Total Cost: $1600.0.
2024-11-13 22:34:01,762 - DEBUG - Processing order for Niya, Product ID: PRD002, Quantity: 1.
2024-11-13 22:34:01,763 - INFO - Order confirmed for Niya. Total Cost: $500.0.
2024-11-13 22:34:01,764 - DEBUG - Processing order for Keni, Product ID: PRD003, Quantity: 5.
2024-11-13 22:34:01,766 - INFO - Order confirmed for Keni. Total Cost: $750.0.
2024-11-13 22:34:01,767 - DEBUG - Processing order for ishan, Product ID: PRD004, Quantity: 1.
2024-11-13 22:34:01,768 - INFO - Order confirmed for ishan. Total Cost: $200.0.
2024-11-13 22:34:01,769 - DEBUG - Processing order for Shubman, Product ID: PRD005, Quantity: 3.
2024-11-13 22:34:01,771 - INFO - Order confirmed for Shubman. Total Cost: $1800.0.
2024-11-13 22:34:01,772 - DEBUG - Processing order for Aksh, Product ID: PRD006, Quantity: 2.
2024-11-13 22:34:01,773 - INFO - Order confirmed for Aksh. Total Cost: $500.0.
2024-11-13 22:34:01,774 - DEBUG - Processing order for Dhruv, Product ID: PRD007, Quantity: 10.
2024-11-13 22:34:01,775 - INFO - Order confirmed for Dhruv. Total Cost: $500.0.
2024-11-13 22:34:01,776 - DEBUG - Processing order for Saanvi, Product ID: PRD008, Quantity: 15.
2024-11-13 22:34:01,777 - INFO - Order confirmed for Saanvi. Total Cost: $450.0.
2024-11-13 22:34:01,777 - DEBUG - Order processing completed.

Order Confirmations:
Order confirmed for Honey. Total Cost: $1600.00.
Order confirmed for Niya. Total Cost: $500.00.
Order confirmed for Keni. Total Cost: $750.00.
Order confirmed for ishan. Total Cost: $200.00.
Order confirmed for Shubman. Total Cost: $1800.00.
Order confirmed for Aksh. Total Cost: $500.00.
Order confirmed for Dhruv. Total Cost: $500.00.
```

```
Order confirmed for Dhruv. Total Cost: $500.00.
Order confirmed for Saanvi. Total Cost: $450.00.
2024-11-13 22:34:01,780 - DEBUG - Generating reports.
2024-11-13 22:34:01,785 - INFO - Reports generated successfully.

REPORT 1: INVENTORY DETAILS:
Product ID Product Name  Total Stock  Available Stock  Price per Unit
    PRD001       Laptop          100               90           800.0
    PRD002   Smartphone          200              145           500.0
    PRD003    Headphone           50               15           150.0
    PRD004      Printer           30               25           200.0
    PRD005       Camera           20                0           600.0
    PRD006   Smartwatch          120               90           250.0
    PRD007     Keyboard          150               70            50.0
    PRD008        Mouse          300              175            30.0

REPORT 2: TOTAL REVENUE EARNED PER CUSTOMER:
Customer Name Product ID  Quantity Ordered  Total Cost
         Aksh     PRD006                 2       500.0
        Dhruv     PRD007                10       500.0
        Honey     PRD001                 2      1600.0
         Keni     PRD003                 5       750.0
         Niya     PRD002                 1       500.0
       Saanvi     PRD008                15       450.0
      Shubman     PRD005                 3      1800.0
        ishan     PRD004                 1       200.0
2024-11-13 22:34:01,796 - INFO - Updated inventory data saved successfully to inventory.csv.
PS C:\Users\lenovo\Desktop\python_lab>
```

LAB Assignment 11 : Data Analysis

You are working for a large e-commerce platform, and your task is to perform customer segmentation based on their shopping behavior. You have access to a dataset containing information about customer transactions. The dataset is in a CSV format and contains the following columns:

1. **CustomerID**: Unique identifier for each customer.
2. **TotalAmountSpent**: The total amount spent by each customer on the platform.
3. **TotalItemsPurchased**: The total number of items purchased by each customer.
4. **LastPurchaseDate**: The date of the customer's most recent purchase.
5. **AveragePurchaseValue**: The average value of each customer's purchases. Using NumPy and

Pandas, your goal is to perform the following tasks:

1. **Data Loading**: Load the dataset into a Pandas DataFrame for analysis.
2. **Data Cleaning**: Check for missing values, duplicates, or any inconsistencies in the data. If found, clean the data appropriately.
3. **Descriptive Statistics**: Calculate basic statistics such as mean, median, and standard deviation of **TotalAmountSpent** and **TotalItemsPurchased**.
4. **Customer Segmentation**: Divide the customers into segments based on their shopping behavior. You can use techniques like K-means clustering or any other method you prefer. For example, you might create segments like "High Spenders," "Frequent Shoppers," and "Inactive Customers."
5. **Visualization**: Create visualizations (e.g., scatter plots, bar charts) to represent the different customer segments you've identified.
6. **Customer Insights**: Provide insights into each customer segment. What distinguishes one segment from another? How can the e-commerce platform tailor its marketing strategies for each segment?
7. **Customer Engagement Recommendations**: Based on your analysis, provide recommendations for the e- commerce platform on how to engage with each customer segment more effectively. For example, should they offer discounts, provide personalized product recommendations, or run targeted marketing campaigns?

This problem requires you to use Pandas for data manipulation, NumPy for numerical operations, and potentially machine learning libraries for customer segmentation. It showcases the power of data analysis and segmentation for making data-driven decisions in e-commerce.

DataSet for the above analysis- https://www.kaggle.com/datasets/puneetbhaya/online-retail/

Dataset contains following information –

Description Quantity InvoiceDate UnitPrice CustomerID Country

Source code:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Step 1: Load the dataset
df = pd.read_csv('C:\\Users\\lenovo\\Desktop\\python_lab\\online Retail.csv', encoding='ISO-8859-1')
print("\nName:Patel Honey Ashishkumar")
print("\nRoll number:22BCP402")
print(df.head())
print("\n")

# Step 2: Data Cleaning
# Drop rows with missing CustomerID
df = df.dropna(subset=['CustomerID'])
print(f"Number of rows after cleaning: {len(df)}")
print("\n")

# Calculate Total Amount Spent
df['TotalAmountSpent'] = df['Quantity'] * df['UnitPrice']

# Group data by CustomerID
customer_df = df.groupby('CustomerID').agg({
    'TotalAmountSpent': 'sum',
    'Quantity': 'sum',
    'InvoiceDate': 'max'
}).reset_index()

# Rename columns
customer_df.rename(columns={'Quantity': 'TotalItemsPurchased', 'InvoiceDate': 'LastPurchaseDate'},
inplace=True)

# Step 3: Data Preparation
customer_df['AveragePurchaseValue'] = customer_df['TotalAmountSpent'] /
customer_df['TotalItemsPurchased']

# Step 4: Descriptive Statistics
print(customer_df[['TotalAmountSpent', 'TotalItemsPurchased']].describe())

# Step 5: Check if there are enough customers for clustering and adjust n_clusters
n_customers = len(customer_df)
print(f"Number of customers after aggregation: {n_customers}")

# If there are fewer than 3 customers, reduce the number of clusters accordingly
n_clusters = min(3, n_customers)  # Set to 3 if there are enough customers, otherwise use all available
customers

if n_customers >= 2:
    print(f"Performing KMeans clustering with {n_clusters} clusters.")
```

```
    # Step 6: Customer Segmentation using K-means
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    customer_df['Segment'] = kmeans.fit_predict(customer_df[['TotalAmountSpent',
'TotalItemsPurchased']])
else:
    print("Not enough data to perform clustering. Setting default segment.")
    # Create a default segment if not enough data for clustering
    customer_df['Segment'] = 0  # All customers are assigned to the same segment

# Step 7: Visualization
plt.scatter(customer_df['TotalAmountSpent'], customer_df['TotalItemsPurchased'],
            c=customer_df['Segment'])
plt.xlabel('Total Amount Spent')
plt.ylabel('Total Items Purchased')
plt.title('Customer Segmentation')
plt.show()

# Step 8: Customer Insights
segment_insights = customer_df.groupby('Segment').mean()
print("\nCustomer Segment Insights:")
print(segment_insights)

# Step 9: Customer Engagement Recommendations

def recommend_engagement(segment):
    if segment == 0:  # Adjust based on your specific cluster analysis
        return "High Spenders: Offer loyalty rewards, exclusive discounts, and premium product
recommendations."
    elif segment == 1:
        return "Frequent Shoppers: Provide personalized product suggestions and regular updates on new
arrivals."
    elif segment == 2:
        return "Inactive Customers: Send re-engagement emails with special offers or discounts to bring
them back."
    else:
        return "General: Maintain regular engagement with newsletters and product updates."

# Apply recommendations to each customer
customer_df['Recommendation'] = customer_df['Segment'].apply(recommend_engagement)

# Display the first few rows to see the recommendations
print("\nCustomer Recommendations:")
print(customer_df[['CustomerID', 'Segment', 'Recommendation']].head())
```

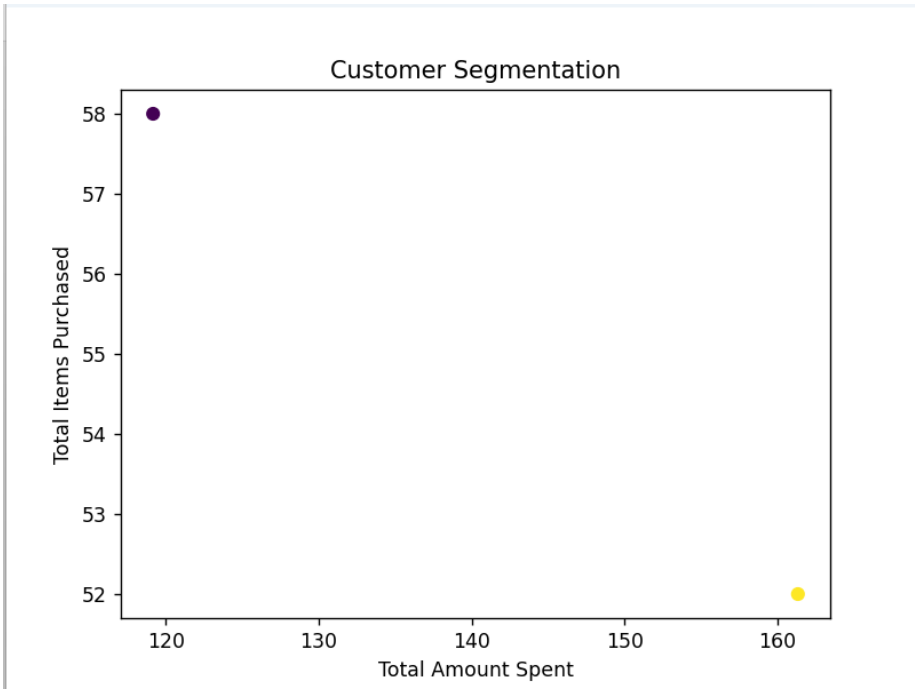
Online Retail.com:

online Retail.csv

```
InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536365,85123A,WHITE HANGING HEART T-LIGHT HOLDER,6,01-12-2010 08:26,2.55,17850,United Kingdom
,71053,WHITE METAL LANTERN,6,01-12-2010 08:26,3.39,17850,United Kingdom
536365,84406B,CREAM CUPID HEARTS COAT HANGER,8,01-12-2010 08:26,2.75,17850,United Kingdom
,84029G,KNITTED UNION FLAG HOT WATER BOTTLE,6,01-12-2010 08:26,3.39,17850,United Kingdom
536365,84029E,RED WOOLLY HOTTIE WHITE HEART.,6,01-12-2010 08:26,3.39,17850,United Kingdom
536365,22752,SET 7 BABUSHKA NESTING BOXES,2,01-12-2010 08:26,7.65,17850,United Kingdom
,21730,GLASS STAR FROSTED T-LIGHT HOLDER,6,01-12-2010 08:26,4.25,17850,United Kingdom
536366,22633,HAND WARMER UNION JACK,6,01-12-2010 08:28,1.85,17850,United Kingdom
536366,22632,HAND WARMER RED POLKA DOT,6,01-12-2010 08:28,1.85,17850,United Kingdom
536367,84879,ASSORTED COLOUR BIRD ORNAMENT,32,01-12-2010 08:34,1.69,13047,United Kingdom
536367,22745,POPPY'S PLAYHOUSE BEDROOM ,6,01-12-2010 08:34,2.1,13047,United Kingdom
,22748,POPPY'S PLAYHOUSE KITCHEN,6,01-12-2010 08:34,2.1,13047,United Kingdom
536367,22749,FELTCRAFT PRINCESS CHARLOTTE DOLL,8,01-12-2010 08:34,3.75,13047,United Kingdom
536367,22310,IVORY KNITTED MUG COSY ,6,01-12-2010 08:34,1.65,13047,United Kingdom
```

Output:

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/Users/lenovo/Desktop/python_lab/dataanalysis.

Name:Patel Honey Ashishkumar

Roll number:22BCP402
   InvoiceNo StockCode                          Description  Quantity       InvoiceDate  UnitPrice  CustomerID         Country
0   536365.0    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6  01-12-2010 08:26       2.55       17850  United Kingdom
1        NaN     71053                  WHITE METAL LANTERN         6  01-12-2010 08:26       3.39       17850  United Kingdom
2   536365.0    84406B       CREAM CUPID HEARTS COAT HANGER         8  01-12-2010 08:26       2.75       17850  United Kingdom
3        NaN    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6  01-12-2010 08:26       3.39       17850  United Kingdom
4   536365.0    84029E       RED WOOLLY HOTTIE WHITE HEART.         6  01-12-2010 08:26       3.39       17850  United Kingdom


Number of rows after cleaning: 14


       TotalAmountSpent  TotalItemsPurchased
count           2.00000             2.000000
mean          140.25000            55.000000
std            29.79748             4.242641
min           119.18000            52.000000
25%           129.71500            53.500000
50%           140.25000            55.000000
75%           150.78500            56.500000
max           161.32000            58.000000
Number of customers after aggregation: 2
Performing KMeans clustering with 2 clusters.
```



Loading the Dataset: It loads the Online.Retail.csv dataset.
Data Cleaning:

- It drops rows where CustomerID is missing to ensure there is no data corruption.
- It prints the number of rows after cleaning.

Calculate Total Amount Spent:

- It calculates the total amount spent by each customer as TotalAmountSpent = Quantity * UnitPrice.

Aggregating by Customer:

- It aggregates the data by CustomerID, calculating the total amount spent, total quantity of items purchased, and the date of the last purchase for each customer.

Data Preparation:

- It creates a new column AveragePurchaseValue as the ratio of total amount spent to the total items purchased.

Cluster Size Check:

- Before running KMeans, it checks if there are enough customers (2 or more) to perform clustering. If there are fewer than 2 customers, it defaults to assigning all customers to a single segment (Segment 0).
- If there are enough customers, it proceeds with KMeans clustering using a dynamic number of clusters based on the number of customers.

KMeans Clustering:

- It uses the KMeans algorithm to cluster the customers into 3 clusters (or fewer if there are less than 3 customers).

Visualization:

- It plots a scatter plot showing TotalAmountSpent vs TotalItemsPurchased, with different colors for different segments.

Customer Insights:

- It calculates the mean values of TotalAmountSpent and TotalItemsPurchased for each segment, providing insights into the behavior of each customer segment.

Customer Engagement Recommendations:
It generates engagement recommendations for each customer based on the cluster they belong to. For example, high spenders get loyalty rewards, frequent shoppers get product suggestions, and inactive customers get re-engagement emails.

Output:

- The program displays the first few rows of customer data, including their CustomerID, assigned segment, and the personalized recommendation.

Lab Assignment 12 : Household Expenses Tracker

You have been tasked with creating a Python program to help manage household expenses. The program should allow family members to input their daily expenses, store them in a CSV file, and provide functionalities for analysis and reporting.

1. Expense Logging: Create a Python program that allows users to input their daily expenses. The program should prompt the user for their name, date of the expense, description, and amount spent. The data should be stored in a CSV file named expenses.csv with columns 'Name', 'Date', 'Description', and 'Amount'.
2. Expense Analysis: Develop a function that reads the expenses.csv file and calculates the total expenses for each family member. Display the total expenses for each member along with the average daily expense for the household.
3. Expense Trends: Implement a feature that generates a line chart using a plotting library (e.g., Matplotlib) to visualize the expense trends over the last month. The x-axis should represent the dates, and the y-axis should show the cumulative expenses for each day.
4. Expense Categorization: Enhance the program to allow users to categorize their expenses. Prompt the user to assign a category (e.g., groceries, utilities, entertainment) to each expense entry. Update the CSV file to include a 'Category' column.
5. Expense Reporting: Create a monthly expense report by reading the data from expenses.csv and generating a report that includes the following:
   - Total expenses for each family member for the month.
   - A breakdown of expenses by category.
   - A comparison of monthly expenses over different months using bar charts.
6. Expense Budgeting: Add an option for users to set a monthly budget for each category. After entering expenses, the program should calculate the remaining budget for each category and provide a warning if the budget is exceeded.
7. Data Backup and Restore: Implement a backup and restore feature that allows users to save a copy of the expenses.csv file to a backup location and restore it if needed. Handle cases where the file might be missing or corrupted.

Source code:

```
import csv
import os
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime

print("Name:Patel Honey Ashishkumar")
print("Roll number:22BCP402")

EXPENSES_FILE = 'expenses.csv'
BACKUP_FILE = 'expenses_backup.csv'

# 1. Expense Logging
def log_expense():
    with open(EXPENSES_FILE, mode='a', newline='') as file:
        writer = csv.writer(file)

        # Gather user input
        name = input("Enter your name: ")
        date = input("Enter the date (YYYY-MM-DD): ")
        description = input("Enter a description of the expense: ")
        amount = float(input("Enter the amount spent: "))
        category = input("Enter the category (e.g., groceries, utilities, entertainment): ")
```

```
        # Write to CSV
        writer.writerow([name, date, description, amount, category])
        print("Expense logged successfully.")

# Initialize CSV with headers if not exists
if not os.path.exists(EXPENSES_FILE):
    with open(EXPENSES_FILE, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['Name', 'Date', 'Description', 'Amount', 'Category'])

# 2. Expense Analysis
def analyze_expenses():
    try:
        # Load the CSV file
        df = pd.read_csv(EXPENSES_FILE)

        # Convert 'Date' column to datetime, handling any invalid dates
        df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
        # Remove rows with invalid dates
        df = df.dropna(subset=['Date'])

        # Debugging: Check the data content after loading
        print("\nData loaded for analysis:")
        print(df.head())

        # Calculate total expenses per family member
        member_totals = df.groupby('Name')['Amount'].sum()
        print("\nTotal expenses per family member:")
        print(member_totals)

        # Group by date and calculate daily expenses
        daily_totals = df.groupby(df['Date'].dt.date)['Amount'].sum()
        print("\nDaily total expenses:")
        print(daily_totals)

        # Calculate average daily expense for the household
        average_daily_expense = daily_totals.mean()
        print(f"\nAverage daily expense for the household: {average_daily_expense:.2f}")

    except Exception as e:
        print(f"Error analyzing expenses: {e}")

# 3. Expense Trends

def plot_expense_trends():
    # Load data
    df = pd.read_csv(EXPENSES_FILE)

    # Ensure 'Date' column is in datetime format
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

    # Drop rows with NaN values in 'Date' or 'Amount'
    df = df.dropna(subset=['Date', 'Amount'])

    # Check if there's data in the filtered DataFrame
    if df.empty:
        print("No data available.")
```

```
        return

    # Sort by date for cumulative calculation
    df = df.sort_values(by='Date')

    # Group by day and calculate daily expenses
    daily_expenses = df.groupby(df['Date'].dt.date)['Amount'].sum()

    # Calculate the cumulative sum of daily expenses
    daily_expenses_cumsum = daily_expenses.cumsum()

    # Plot the cumulative expenses
    plt.figure(figsize=(10, 6))
    plt.plot(pd.to_datetime(daily_expenses.index), daily_expenses_cumsum.values, marker='o')
    plt.xlabel('Date')
    plt.ylabel('Cumulative Expenses')
    plt.title('Expense Trends')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()


# 4. Expense Reporting
def generate_expense_report():
    df = pd.read_csv(EXPENSES_FILE)
    current_month = datetime.now().month
    df['Date'] = pd.to_datetime(df['Date'])
    monthly_df = df[df['Date'].dt.month == current_month]

    # Total expenses for each family member for the current month
    member_totals = monthly_df.groupby('Name')['Amount'].sum()
    print("\nMonthly expenses per family member:")
    print(member_totals)

    # Breakdown of expenses by category
    category_totals = monthly_df.groupby('Category')['Amount'].sum()
    print("\nExpense breakdown by category:")
    print(category_totals)

    # Comparison of monthly expenses
    monthly_totals = df.groupby(df['Date'].dt.to_period('M'))['Amount'].sum()
    monthly_totals.plot(kind='bar', title="Monthly Expenses Comparison")
    plt.xlabel('Month')
    plt.ylabel('Total Expenses')
    plt.tight_layout()
    plt.show()

# 6. Expense Budgeting
def set_budget():
    budgets = {}
    print("Set monthly budget for each category (enter 0 if no budget for a category):")
    while True:
        category = input("Enter category name (or type 'done' to finish): ")
        if category.lower() == 'done':
            break
        amount = float(input(f"Enter budget for {category}: "))
        budgets[category] = amount
    return budgets
```

```
# The function calculates how much has been spent in each category for the current month.
def check_budget(budgets):
    df = pd.read_csv(EXPENSES_FILE)
    current_month = datetime.now().month
    df['Date'] = pd.to_datetime(df['Date'])
    monthly_df = df[df['Date'].dt.month == current_month]

    category_expenses = monthly_df.groupby('Category')['Amount'].sum()
    for category, budget in budgets.items():
        if budget > 0:
            spent = category_expenses.get(category, 0)
            remaining = budget - spent
            if remaining < 0:
                print(f"Warning: Budget exceeded for {category} by {abs(remaining):.2f}")
            else:
                print(f"Remaining budget for {category}: {remaining:.2f}")

# 7. Data Backup and Restore
def backup_data():
    try:
        pd.read_csv(EXPENSES_FILE).to_csv(BACKUP_FILE, index=False)
        print("Data backup completed successfully.")
    except FileNotFoundError:
        print("Expenses file not found. No backup created.")

def restore_data():
    try:
        pd.read_csv(BACKUP_FILE).to_csv(EXPENSES_FILE, index=False)
        print("Data restored from backup successfully.")
    except FileNotFoundError:
        print("Backup file not found. Unable to restore data.")

# Main menu
def main():
    budgets = {}
    while True:
        print("\nHousehold Expenses Tracker Menu:")
        print("1. Log Expense")
        print("2. Analyze Expenses")
        print("3. Plot Expense Trends")
        print("4. Generate Expense Report")
        print("5. Set Budget")
        print("6. Check Budget")
        print("7. Backup Data")
        print("8. Restore Data")
        print("9. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            log_expense()
        elif choice == '2':
            analyze_expenses()
        elif choice == '3':
            plot_expense_trends()
        elif choice == '4':
            generate_expense_report()
```

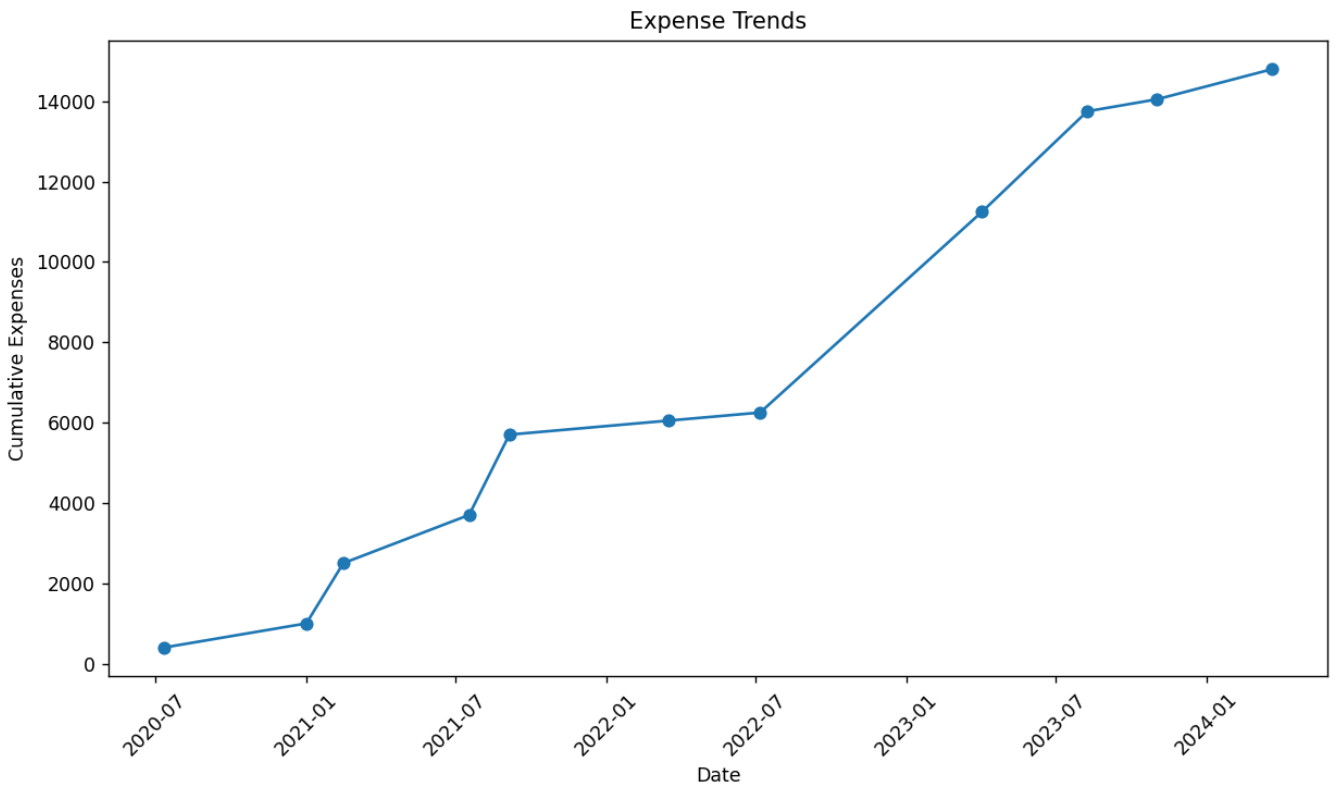
```
            elif choice == '5':
                budgets = set_budget()
            elif choice == '6':
                check_budget(budgets)
            elif choice == '7':
                backup_data()
            elif choice == '8':
                restore_data()
            elif choice == '9':
                print("Exiting program.")
                break
            else:
                print("Invalid choice. Please try again.")

        if __name__ == '__main__':
            main()
```

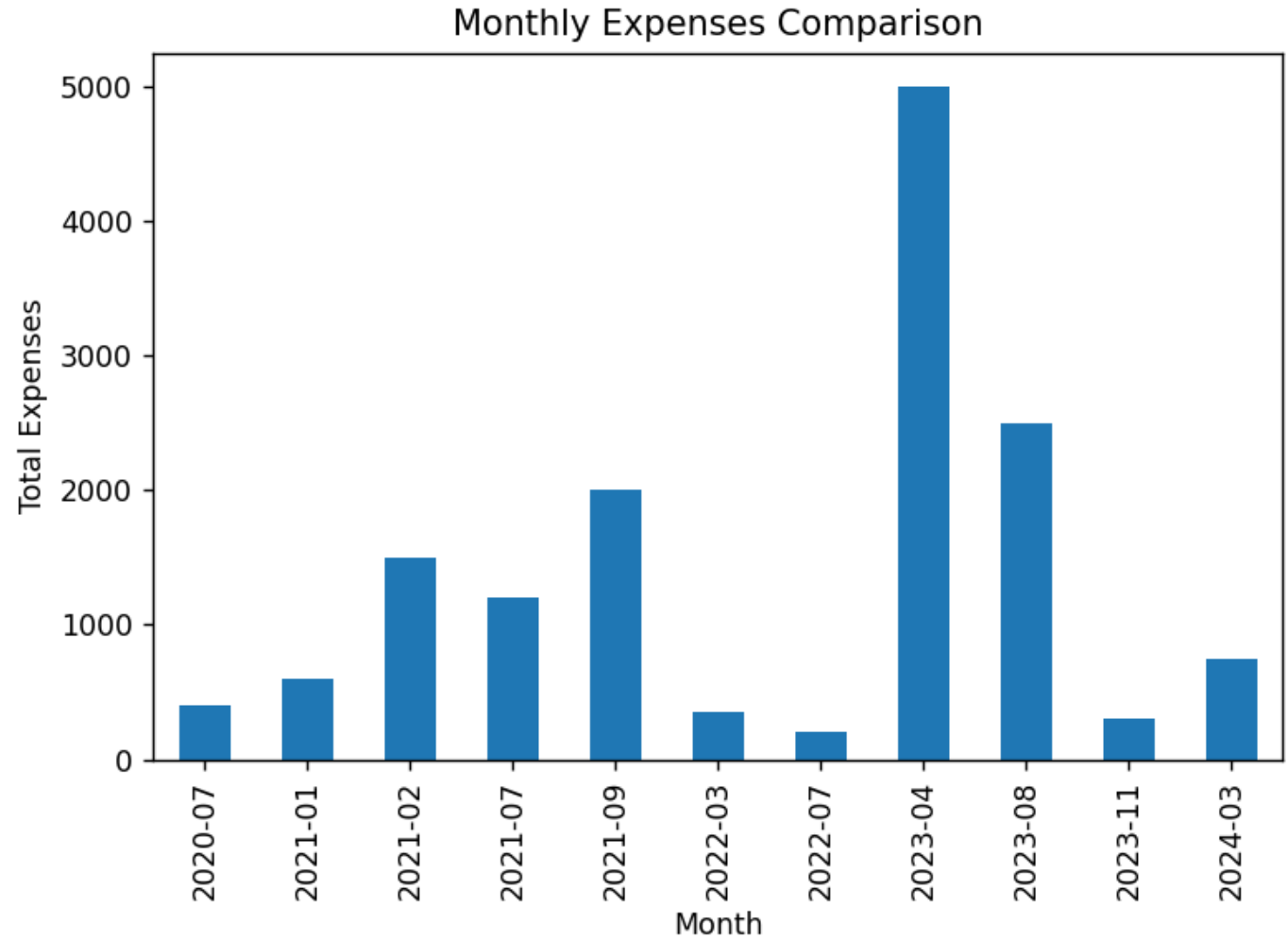
Output:

```
PS C:\Users\lenovo\Desktop\python_lab> & "C:/Program Files/Python313/python.exe" c:/Us
Name:Patel Honey Ashishkumar
Roll number:22BCP402

Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 1
Enter your name: rajvi
Enter the date (YYYY-MM-DD): 2024-03-21
Enter a description of the expense: movietickets
Enter the amount spent: 750
Enter the category (e.g., groceries, utilities, entertainment): entertainment
Expense logged successfully.
```

```
Enter the category (e.g., groceries, utilities, entertainment): entertainment
Expense logged successfully.

Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 1
Enter your name: ashish patel
Enter the date (YYYY-MM-DD): 2021-01-01
Enter a description of the expense: vegetables
Enter the amount spent: 600
Enter the category (e.g., groceries, utilities, entertainment): groceries
Expense logged successfully.
```

```
Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 3
```


Expense Trends
Cumulative Expenses
0
2000
4000
6000
8000
10000
12000
14000
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
Date


Monthly Expenses Comparison
Total Expenses
0
1000
2000
3000
4000
5000
2020-07
2021-01
2021-02
2021-07
2021-09
2022-03
2022-07
2023-04
2023-08
2023-11
2024-03
Month

```
Enter your choice: 5
Set monthly budget for each category (enter 0 if no budget for a category):
Enter category name (or type 'done' to finish): study
Enter budget for study: 2500
Enter category name (or type 'done' to finish): food
Enter budget for food: 5000
Enter category name (or type 'done' to finish): entertainme
nt
Enter budget for entertainment: 3000
Enter category name (or type 'done' to finish): done

Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 6
Remaining budget for study: 2500.00
Remaining budget for food: 5000.00
Remaining budget for entertainment: 3000.00
```

```
Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 7
Data backup completed successfully.

Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 8
Data restored from backup successfully.
```

```
Enter your choice: 8
Data restored from backup successfully.

Household Expenses Tracker Menu:
1. Log Expense
2. Analyze Expenses
3. Plot Expense Trends
4. Generate Expense Report
5. Set Budget
6. Check Budget
7. Backup Data
8. Restore Data
9. Exit
Enter your choice: 9
Exiting program.
PS C:\Users\lenovo\Desktop\python_lab>
```

Expenses.csv

expenses.csv

```
Name,Date,Description,Amount,Category
Honey Patel,2022-07-07,fruits,200.0,groceries
Keni Tandel,2023-08-09,Toys,2500.0,entertainmment
Shubman gill,2021-07-18,bday cake,1200.0,groceries
Ishan kishan,2023-11-02,cooking oil,300.0,groceries
Aksh patel,2021-02-15,electricity bill,1500.0,utilities
Dhruv tandel,2021-09-05,car maintenance,2000.0,utilities
Niya Patel,2023-04-03,Netflix,5000.0,entertainment
Raj Patel,2020-07-12,spices,400.0,groceries
Sara,2022-03-18,phone bill,350.0,utilities
rajvi,2024-03-21,movietickets,750.0,entertainment
ashish patel,2021-01-01,vegetables,600.0,groceries
```

Expenses_backup.csv

expenses_backup.csv

```
Name,Date,Description,Amount,Category
Honey Patel,2022-07-07,fruits,200.0,groceries
Keni Tandel,2023-08-09,Toys,2500.0,entertainmment
Shubman gill,2021-07-18,bday cake,1200.0,groceries
Ishan kishan,2023-11-02,cooking oil,300.0,groceries
Aksh patel,2021-02-15,electricity bill,1500.0,utilities
Dhruv tandel,2021-09-05,car maintenance,2000.0,utilities
Niya Patel,2023-04-03,Netflix,5000.0,entertainment
Raj Patel,2020-07-12,spices,400.0,groceries
Sara,2022-03-18,phone bill,350.0,utilities
rajvi,2024-03-21,movietickets,750.0,entertainment
ashish patel,2021-01-01,vegetables,600.0,groceries
```