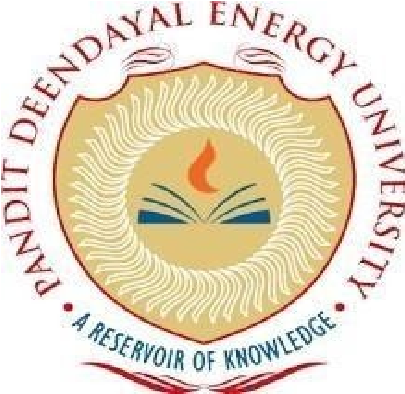
2024

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: Saanvi Nilesh Dave

Enrolment No.: 22BCP241 Semester: 5

Division: 4

Group: G7

Instructor: Dr. Rajendra Choudhary

## Table of Contents

[LAB Assignment 1: Product Review from text file 3](#_Toc3246)

[LAB Assignment 2: Railway Ticket Reservation System 6](#_Toc3247)

Lab Assignment 3: Image and Audio Data Processing.............................................................................................11 LAB Assignment 4: Logging engineer.......................................................................................................................17

LAB Assignment 5 : Data Analysis...........................................................................................................................29 Lab Assignment 6 : CSV Manipulation.....................................................................................................................43 Lab Assignment 7 :CSV In depth...............................................................................................................................45 Lab Assignment 8 : JSON Handling..........................................................................................................................49 Lab Assignment 9 : Pdf reading creation and editing................................................................................................53 Lab Assignment 10: Error and Exception Handling...................................................................................................57 Lab Assignment 11 : User Management Automation.................................................................................................62 Lab Assignment 12 : Household Expenses Tracker...................................................................................................67

# 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 analysing 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.

**Logs.txt** p1 c1 2023-07-03 5 <good> p2 c2 2023-07-04 4 <better> p3 c1 2023-07-07 3 <best> p1 c3 2023-07-04 2 <worst> p2 c4 2023-07-07 1 <awesome>

### logs2.txt

p4 c1 2023-07-03 5 <good> p2 c2 2023-07-04 3 <better> p4 c1 2023-07-07 3 <best> p1 c3 2023-07-04 5 <worst> p1 c4 2023-07-07 4 <awesome>

**summary.txt** p3 3.0 p2 2.6666666666666665 p1 4.0 p4 4.0 Valid: 10

Invalid: 0

Total: 10

CODE:

def is\_valid\_review(review):

if len(review) != 5:

return False

customer\_id, product\_id, review\_date, review\_rating, review\_text = review

if len(customer\_id) != 6 and not customer\_id.isalnum():

return False

if len(product\_id) != 10 and not product\_id.isalnum():

return False

if len(review\_date) != 10 or review\_date[4] != '-' or review\_date[7] != '-':

return False

year, month, day = review\_date.split('-')

if not (year.isdigit() and month.isdigit() and day.isdigit()):

return False

if not review\_rating.isdigit() and not (0 <= int(review\_rating) <= 5):

return False

if not review\_text:

return False

return True

def open\_file(file\_path):

with open(file\_path, "r") as fp:

lines = []

for line in fp:

line = line.strip()

if line:

parts = line.split()

if len(parts) >= 5:

review\_text = ' '.join(parts[4:])

if review\_text.startswith('<') and review\_text.endswith('>'):

review\_text = review\_text[1:-1]

parts = parts[:4] + [review\_text]

lines.append(parts)

else:

lines.append(['Invalid'] \* 5)

return lines

def process\_reviews(file\_paths):

valid\_reviews = []

invalid\_reviews = []

valid\_reviews\_count = 0

invalid\_reviews\_count = 0

for file\_path in file\_paths:

lines = open\_file(file\_path)

for line in lines:

if 'Invalid' in line or len(line) != 5:

invalid\_reviews.append(line)

invalid\_reviews\_count += 1

else:

if is\_valid\_review(line):

valid\_reviews.append(line)

valid\_reviews\_count += 1

else:

invalid\_reviews.append(line)

invalid\_reviews\_count += 1

total\_reviews\_count = valid\_reviews\_count + invalid\_reviews\_count

return valid\_reviews, invalid\_reviews, valid\_reviews\_count, invalid\_reviews\_count, total\_reviews\_count

def print\_reviews(valid\_reviews, invalid\_reviews):

print("Valid Reviews:")

for review in valid\_reviews:

if len(review) == 5:

print("Customer ID: {}, Product ID: {}, Review Date: {}, Review Rating: {}, Review Text: {}".format(\*review))

else:

print("Invalid review format (Valid Reviews):", review)

def calculate\_average\_ratings(reviews):

product\_ratings = {}

for review in reviews:

product\_id = review[1]

review\_rating = int(review[3])

if product\_id not in product\_ratings:

product\_ratings[product\_id] = []

product\_ratings[product\_id].append(review\_rating)

average\_ratings = {}

for product\_id, ratings in product\_ratings.items():

total = sum(ratings)

average\_ratings[product\_id] = total / len(ratings)

return average\_ratings

def create\_summary(file\_name, total\_reviews, valid\_reviews\_count, invalid\_reviews\_count, top\_3\_products):

with open(file\_name, "w") as summary\_file:

summary\_file.write("Total number of reviews processed: {}\n".format(total\_reviews))

summary\_file.write("Total number of valid reviews: {}\n".format(valid\_reviews\_count))

summary\_file.write("Total number of invalid reviews: {}\n".format(invalid\_reviews\_count))

summary\_file.write("Top 3 products with highest average ratings:\n")

for product\_id, avg\_rating in top\_3\_products:

summary\_file.write("Product ID: {}, Average Rating: {:.2f}\n".format(product\_id, avg\_rating))

num\_files = int(input("Enter the number of files: "))

file\_paths = [input("Enter the path for file {}: ".format(i + 1)) for i in range(num\_files)]

valid\_reviews, invalid\_reviews, valid\_reviews\_count, invalid\_reviews\_count, total\_reviews\_count = process\_reviews(file\_paths)

print("\n")

print\_reviews(valid\_reviews, invalid\_reviews)

average\_ratings = calculate\_average\_ratings(valid\_reviews)

print("\n")

print(average\_ratings)

top\_3\_products = sorted(average\_ratings.items(), key=lambda item: item[1], reverse=True)[:3]

create\_summary("summary.txt", total\_reviews\_count, valid\_reviews\_count, invalid\_reviews\_count, top\_3\_products)

print("\nTotal number of reviews processed: {}".format(total\_reviews\_count))

print("\n")

print("Total number of valid reviews: {}".format(valid\_reviews\_count))

print("\n")

print("Total number of invalid reviews: {}".format(invalid\_reviews\_count))

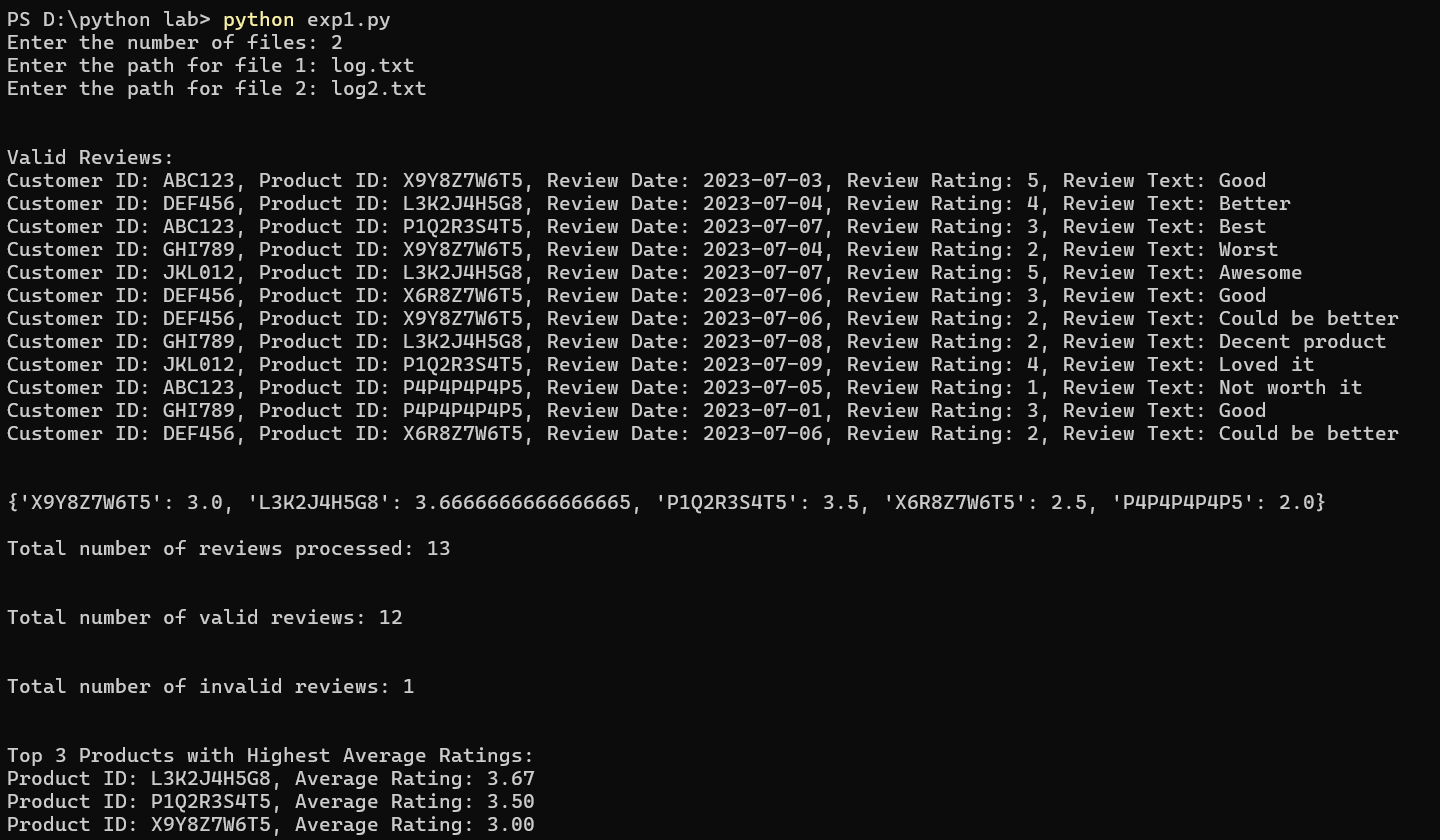
print("\n")

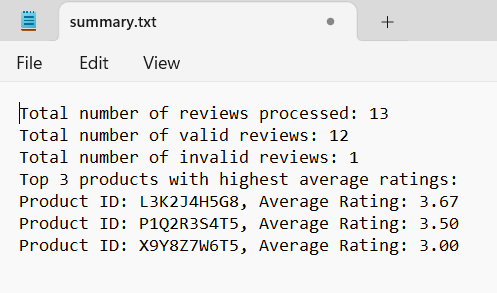
print("Top 3 Products with Highest Average Ratings:")

for product\_id, avg\_rating in top\_3\_products:

print("Product ID: {}, Average Rating: {:.2f}".format(product\_id, avg\_rating))

**OUTPUT:**

****

****

# 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.

**Sample data-**

## **Trains.csv**

Train ID,Train Name,Source Station,Destination Station,Total Seats, Available Seats, Total fare

T123,Express 123,City A,City B,200,198,500

T456,Superfast 456,City B,City C,150,147,1200

T789,Shatabdi 789,City C,City D,100,96,1400

T321,Intercity 321,City D,City E,180,179,200

T654,Rajdhani 654,City E,City F,250,250,1500

## **Passengers.csv**

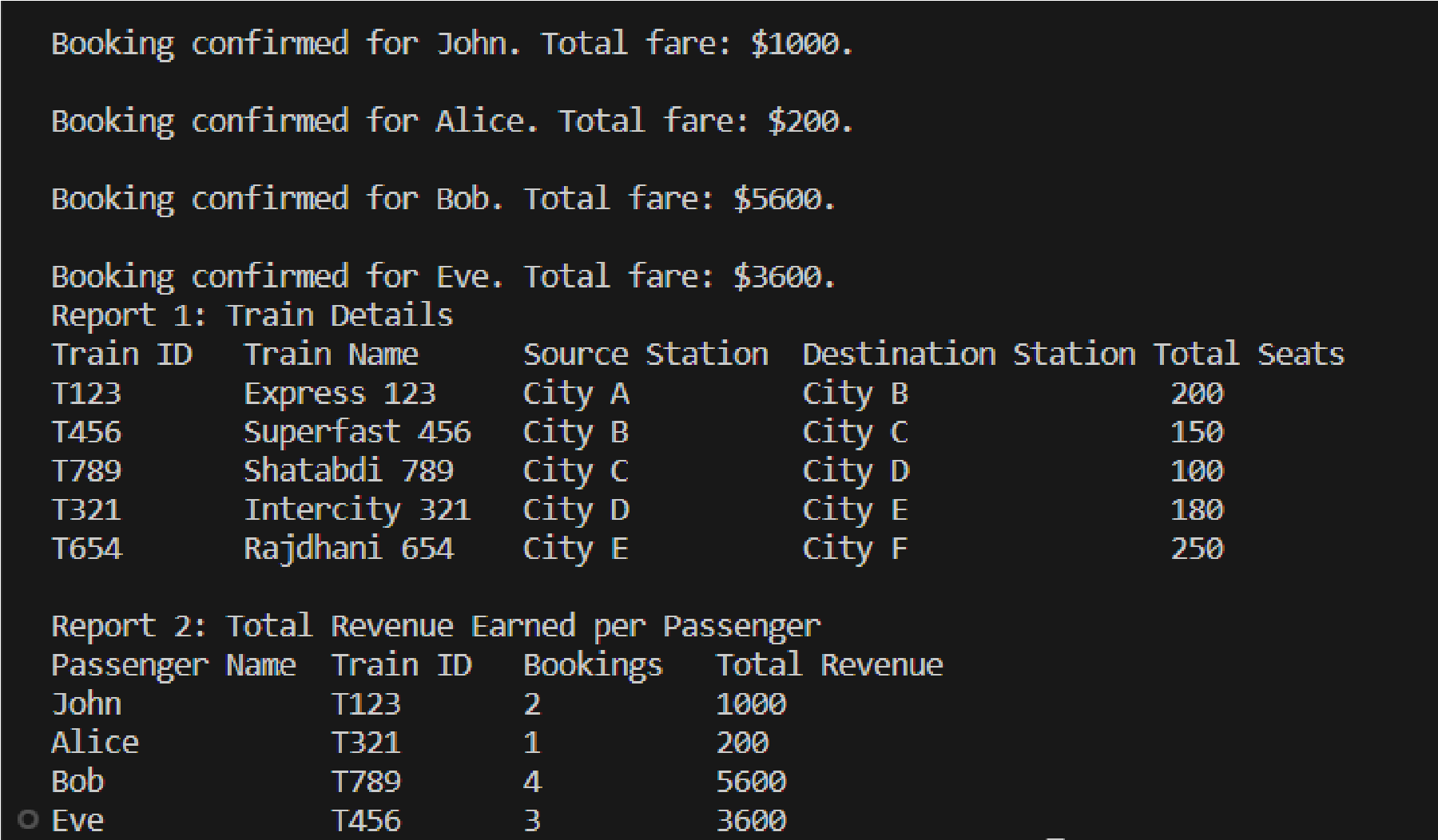
Passenger Name,Train ID,Number of Tickets

John,T123,2

Alice,T321,1

Bob,T789,4

Eve,T456,3

**Output:**

**CODE:**

import pandas as pd

# Load Train Data

def load\_train\_data(file\_path):

df = pd.read\_csv(file\_path)

df.columns= df.columns.str.strip()

return df

# Load Passenger Data

def load\_passenger\_data(file\_path):

df = pd.read\_csv(file\_path)

df.columns = df.columns.str.strip()

return df

# Process Bookings and Update Availability

def process\_bookings(trains\_df, passengers\_df):

booking\_failures = []

confirmed\_bookings = []

for \_, row in passengers\_df.iterrows():

passenger\_name = row['Passenger Name']

train\_id = row['Train ID']

num\_tickets = row['Number of Tickets']

if len(str(train\_id)) != 4:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Invalid train number {train\_id}.")

continue

train = trains\_df[trains\_df['Train ID'] == train\_id]

if not train.empty:

available\_seats = train.iloc[0]['Available Seats']

if not isinstance(num\_tickets, int) or any(char.isdigit() for char in passenger\_name): booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Invalid input: number of tickets should be an integer and passenger name should not contain numbers.")

continue

if available\_seats >= num\_tickets:

# Update available seats in trains\_df

trains\_df.loc[trains\_df['Train ID'] == train\_id, 'Available Seats'] -= num\_tickets

# Add to confirmed bookings

fare = num\_tickets \* train.iloc[0]['Total fare']

confirmed\_bookings.append({

'Passenger Name': passenger\_name,

'Train ID': train\_id,

'Number of Tickets': num\_tickets,

'Fare': fare

})

else:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Available seats are only {available\_seats}.")

else:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Train ID {train\_id} not found.")

# Convert confirmed bookings list to DataFrame

confirmed\_bookings\_df = pd.DataFrame(confirmed\_bookings)

return trains\_df, confirmed\_bookings\_df, booking\_failures

# Generate Reports

def generate\_reports(trains\_df, confirmed\_bookings):

# Report 1: Train Details

train\_report = trains\_df[['Train ID', 'Train Name', 'Source Station', 'Destination Station','Total Seats', 'Available Seats']]

# Report 2: Total Revenue Earned Per Passenger

revenue\_per\_passenger = confirmed\_bookings.groupby(['Passenger Name', 'Train ID']).agg({

'Number of Tickets': 'sum',

'Fare': 'sum'

}).reset\_index()

revenue\_per\_passenger.rename(columns={'Number of Tickets': 'Bookings', 'Fare': 'Total Revenue'}, inplace=True)

return train\_report, revenue\_per\_passenger

# Main program

def main():

# Get file paths from the user

trains\_file = input("Enter the path to the trains CSV file: ")

passengers\_file = input("Enter the path to the passengers CSV file: ")

# Load data

trains\_df = load\_train\_data(trains\_file)

passengers\_df = load\_passenger\_data(passengers\_file)

# Process bookings

trains\_df, confirmed\_bookings, booking\_failures = process\_bookings(trains\_df, passengers\_df)

# Print booking results

print("Booking Confirmations:")

for \_, row in confirmed\_bookings.iterrows():

print(f"Booking confirmed for {row['Passenger Name']}. Total Fare: ${row['Fare']:.2f}.")

# Print failed bookings

print("\nBooking Failures:")

for failure in booking\_failures:

print(failure)

# Generate reports

train\_report, revenue\_per\_passenger = generate\_reports(trains\_df, confirmed\_bookings)

print("\nREPORT 1: TRAIN DETAILS:")

print(train\_report.to\_string(index=False))

print("\nREPORT 2: TOTAL REVENUE EARNED PER PASSENGER:")

print(revenue\_per\_passenger.to\_string(index=False))

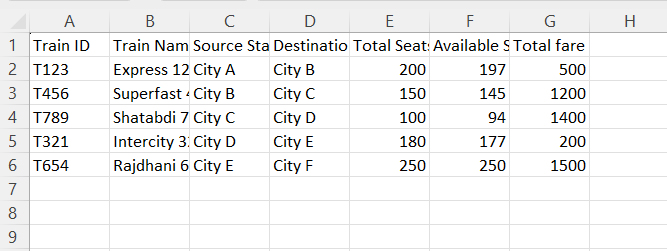
trains\_df.to\_csv(trains\_file, index=False)

if \_\_name\_\_ == "\_\_main\_\_":

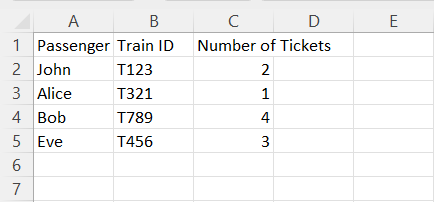
main()

**Data:**

**Trains.csv:**

****

**Passengers.csv**

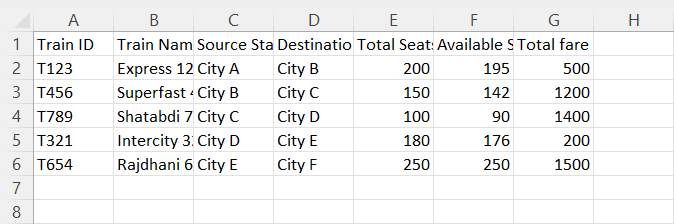
****

**OUTPUT:**

****

**After running:**

**Trains.csv:**

****

# 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

###### Read the data from "student\_grades.csv" using CSV file handling in Python.

###### For each student, calculate their average score across all subjects (Maths, Science, and English).

###### Create average functions to calculate the average for each student.

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

###### Write the data from the dictionary into a new CSV file named "student\_average\_grades.csv" with two columns: "Name" and "Average."

**CODE:**

import pandas as pd

input\_file = input("Enter the CSV file containing student's marks: ")

output\_file="student\_average\_grades.csv"

df = pd.read\_csv(input\_file)

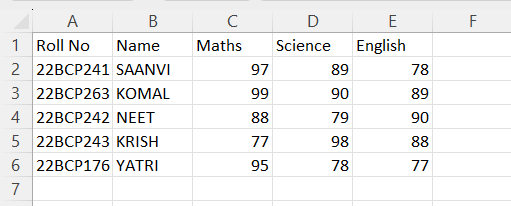
df['Average'] = df[["Maths", "Science", "English"]].mean(axis=1)

average\_df = df[["Roll No","Name", "Average"]]

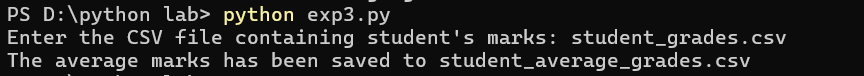
average\_df.to\_csv("student\_average\_grades.csv", index=False)

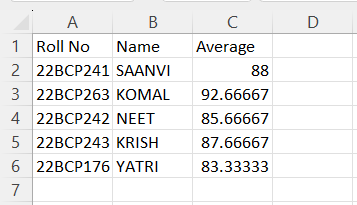
print(f"The average marks has been saved to {output\_file}")

**DATA:**

****

**OUTPUT:**

****



# 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 analysing 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)

**CODE:**

import os

import pandas as pd

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)

def calculate\_sales\_summary(sales\_data, product\_names):

try:

total\_sales = sales\_data.groupby('Product ID')['Quantity sold'].sum().reset\_index()

summary = pd.merge(total\_sales, product\_names, on='Product ID', how='left')

if 'Date' in sales\_data.columns:

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

total\_months = sales\_data['Date'].dt.to\_period('M').nunique()

summary['Average Quantity Sold per Month'] = summary['Quantity sold'] / total\_months

else:

summary['Average Quantity Sold per Month'] = summary['Quantity sold']

return summary

except Exception as e:

print(f"Error in calculating sales summary: {e}")

return pd.DataFrame()

def get\_top\_selling\_products(sales\_summary):

return sales\_summary.sort\_values(by='Quantity sold', ascending=False).head(5)

def main():

try:

directory = input("Enter the path to the directory containing the sales data: ")

product\_names = pd.read\_csv('product\_names.csv')

sales\_data = read\_sales\_data(directory)

sales\_summary = calculate\_sales\_summary(sales\_data, product\_names)

top\_selling\_products = get\_top\_selling\_products(sales\_summary)

sales\_summary.to\_csv('sales\_summary.csv', index=False)

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'.")

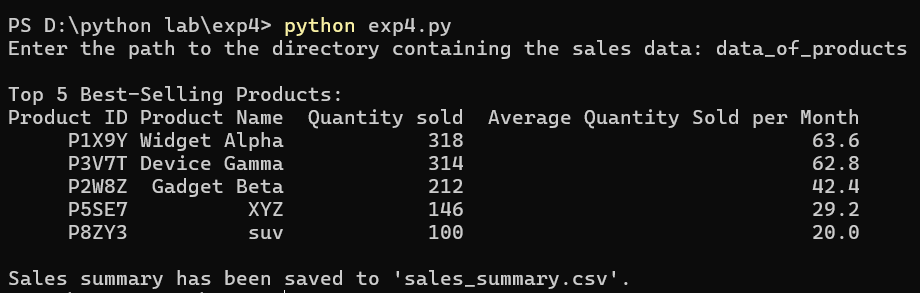
except Exception as e:

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

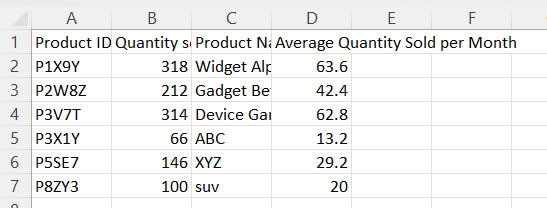
if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:**

****

sales\_summary.csv



# 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:

###### Read COVID-19 data from all JSON files in a given directory and its subdirectories.

###### Calculate and display the following statistics for each country:

###### Total confirmed cases.

###### Total deaths.

###### Total recovered cases.

###### Total active cases (total confirmed cases minus total deaths and total recovered).

###### Determine the top 5 countries with the highest number of confirmed cases and the lowest number of confirmed cases.

###### Generate a summary report in JSON format that includes the statistics for all countries and save it to a file named "covid19\_summary.json".

**CODE:**

import os

import json

from collections import defaultdict

def read\_json\_files(directory):

data = []

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

for file in files:

if file.endswith(".json"):

file\_path = os.path.join(root, file)

with open(file\_path, "r") as f:

data.append(json.load(f))

return data

def calculate\_statistics(data):

country\_stats = defaultdict(lambda: {

"total\_confirmed\_cases": 0,

"total\_deaths": 0,

"total\_recovered": 0,

"total\_active\_cases": 0

})

for entry in data:

country = entry["country"]

confirmed\_cases = entry["confirmed\_cases"]["total"]

deaths = entry["deaths"]["total"]

recovered = entry["recovered"]["total"]

country\_stats[country]["total\_confirmed\_cases"] += confirmed\_cases

country\_stats[country]["total\_deaths"] += deaths

country\_stats[country]["total\_recovered"] += recovered

country\_stats[country]["total\_active\_cases"] = (

country\_stats[country]["total\_confirmed\_cases"]

- country\_stats[country]["total\_deaths"]

- country\_stats[country]["total\_recovered"]

)

return country\_stats

def find\_top\_countries(country\_stats, n=5):

sorted\_by\_confirmed = sorted(country\_stats.items(),

key=lambda x: x[1]["total\_confirmed\_cases"],

reverse=True)

top\_countries = sorted\_by\_confirmed[:n]

bottom\_countries = sorted\_by\_confirmed[-n:]

return top\_countries, bottom\_countries

def generate\_summary\_report(country\_stats, top\_countries, bottom\_countries, output\_file="covid19\_summary.json"):

report = {

"country\_statistics": country\_stats,

"top\_5\_highest\_confirmed\_cases": top\_countries,

"top\_5\_lowest\_confirmed\_cases": bottom\_countries

}

with open(output\_file, "w") as f:

json.dump(report, f, indent=4)

def main(directory):

data = read\_json\_files(directory)

country\_stats = calculate\_statistics(data)

top\_countries, bottom\_countries = find\_top\_countries(country\_stats)

generate\_summary\_report(country\_stats, top\_countries, bottom\_countries)

print("Top 5 countries with the highest confirmed cases:")

for country, stats in top\_countries:

print(f"{country}: {stats['total\_confirmed\_cases']} confirmed cases")

print("\nTop 5 countries with the lowest confirmed cases:")

bottom\_countries\_sorted = sorted(bottom\_countries, key=lambda x: x[1]["total\_confirmed\_cases"])

for country, stats in bottom\_countries\_sorted:

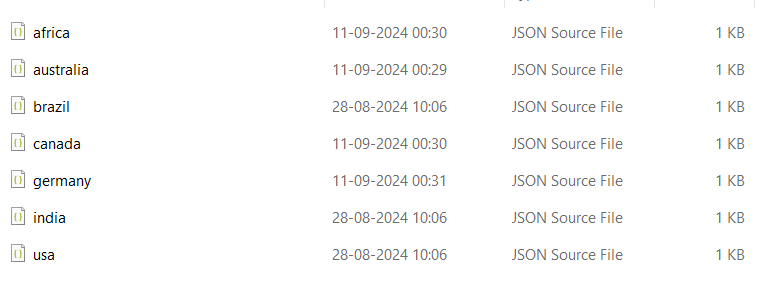
print(f"{country}: {stats['total\_confirmed\_cases']} confirmed cases")

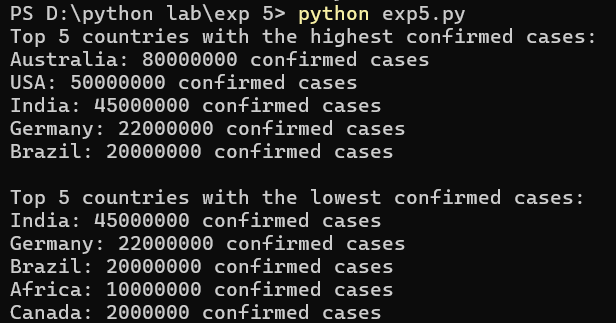
directory = r'D:\python lab\exp 5\CovidData’

main(directory)

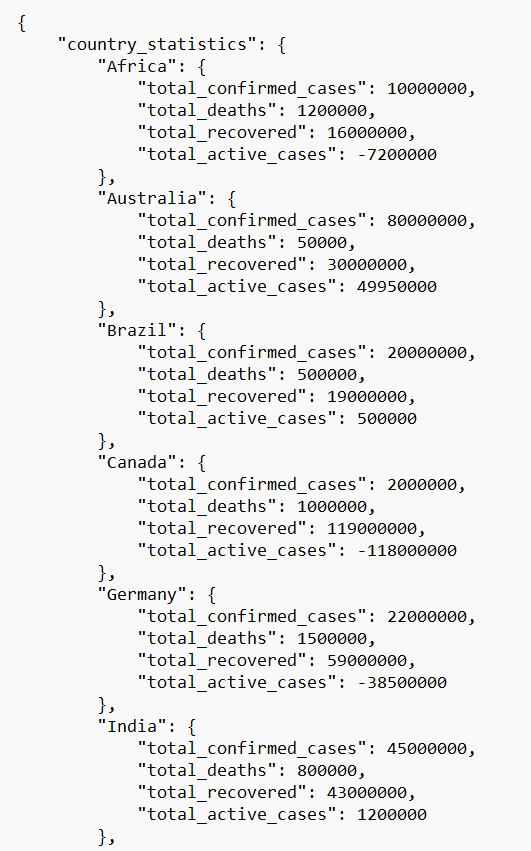
**OUTPUT:**

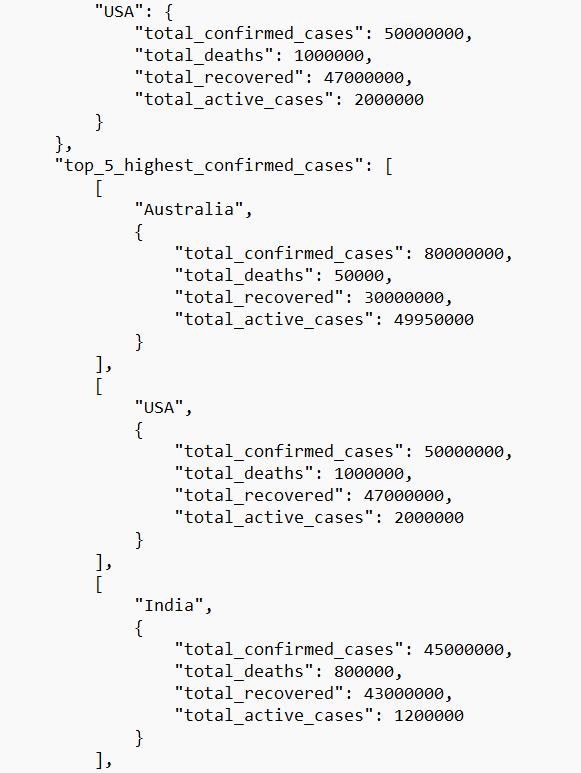
**Data:**

****

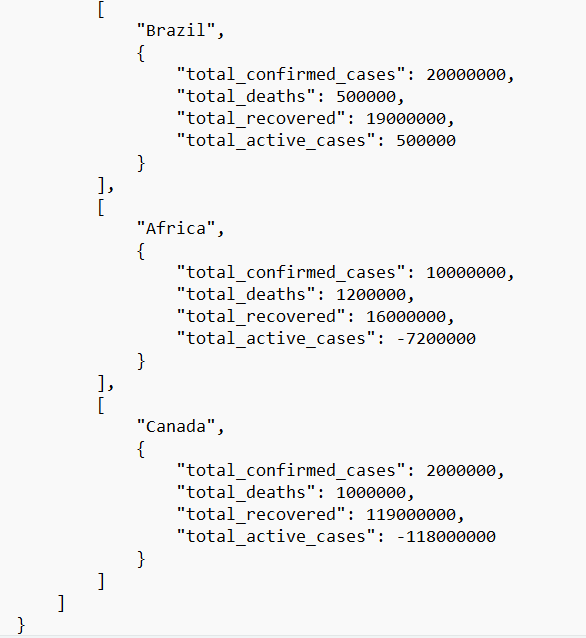
****

Covid19\_summary.json:









# Lab Assignment 6: Error and Exceptional 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).

**CODE:**

import os

from collections import Counter

from wordcloud import WordCloud

import matplotlib.pyplot as plt

# Function to read input data from file

def read\_input\_file(file\_path):

if not os.path.exists(file\_path):

print(f"Error: File '{file\_path}' not found.")

return None

with open(file\_path, "r") as file:

return file.read()

# Function to process text data (word count, character frequency)

def process\_text\_data(text):

word\_count = len(text.split())

char\_frequency = Counter(text)

return {

"word\_count": word\_count,

"char\_frequency": char\_frequency,

"text": text

}

# Function to save results in output file (text format)

def save\_results\_txt(results, output\_file):

with open(output\_file, "w") as file:

file.write(f"number of words: {results['word\_count']}\n")

file.write("character frequencies:\n")

for char, freq in results["char\_frequency"].items():

file.write(f"{char}: {freq}\n")

print(f"Results saved to '{output\_file}'.")

# Function to generate word cloud

def generate\_word\_cloud(text):

wordcloud = WordCloud(width=800, height=400, background\_color="black", colormap="viridis").generate(text)

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis("off")

plt.show()

# Main function

def main():

file\_path = input("Enter the path to the input text file: ")

output\_file = input("Enter the desired output file name (e.g., output.txt): ")

# Read input data

text\_data = read\_input\_file(file\_path)

if text\_data is None:

return

# Process text data

results = process\_text\_data(text\_data)

# Save the results in a text file

save\_results\_txt(results, output\_file)

# Generate and display the word cloud

generate\_word\_cloud(results["text"])

if \_\_name\_\_ == "\_\_main\_\_":

main()

**OUTPUT:**

Exp6.txt:

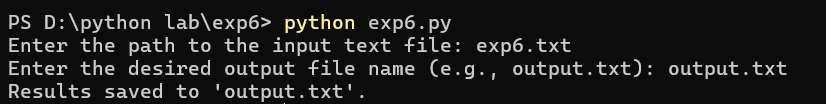
INPUT OUTPUT LAYER MODEL

CODE MACHINE LEARNING PYTHON

MODULES LIBRARY DJANGO

FRAMEWORKS

Hello how are you....?



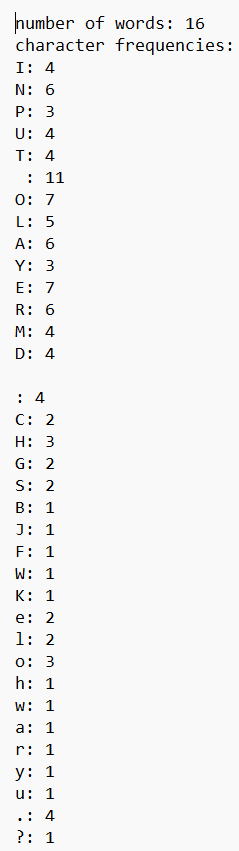
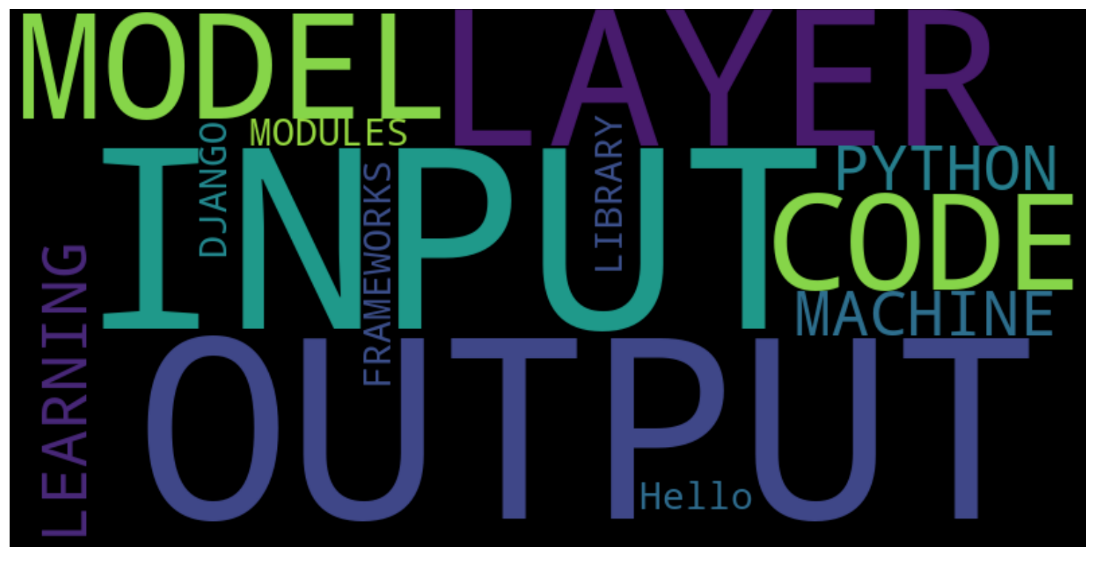


Figure:



# Lab Assignment 7: PDF reading creation and editing

You have to develop a Python program and you are working for a company that sells products online. Your task 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
  7. Total Amount

**CODE:**

import csv

from fpdf import FPDF

from PyPDF2 import PdfMerger

from datetime import date

def create\_invoice(order):

# Prepare invoice data

order\_id = order['Order ID']

customer\_name = order['Customer Name']

product\_name = order['Product Name']

quantity = int(order['Quantity'])

unit\_price = float(order['Unit Price'])

total\_amount = quantity \* unit\_price

pdf\_file = f"invoice\_{order\_id}.pdf"

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", "B", 16)

pdf.cell(200, 10, f"Invoice Number: {order\_id}", ln=True, align="C")

pdf.ln(10)

pdf.set\_font("Arial", "B", 12)

pdf.set\_fill\_color(169, 169, 169) # Gray background for headers

pdf.cell(90, 10, "Field", 1, 0, "C", fill=True)

pdf.cell(100, 10, "Value", 1, 1, "C", fill=True)

pdf.set\_font("Arial", "", 12)

pdf.set\_fill\_color(245, 245, 245) # Light fill for rows

pdf.cell(90, 10, "Date of Purchase:", 1, 0, "L", fill=True)

pdf.cell(100, 10, str(date.today()), 1, 1, "L")

pdf.cell(90, 10, "Customer Name:", 1, 0, "L", fill=True)

pdf.cell(100, 10, customer\_name, 1, 1, "L")

pdf.cell(90, 10, "Product Name:", 1, 0, "L", fill=True)

pdf.cell(100, 10, product\_name, 1, 1, "L")

pdf.cell(90, 10, "Quantity:", 1, 0, "L", fill=True)

pdf.cell(100, 10, str(quantity), 1, 1, "L")

pdf.cell(90, 10, "Unit Price:", 1, 0, "L", fill=True)

pdf.cell(100, 10, f"${unit\_price:.2f}", 1, 1, "L")

pdf.cell(90, 10, "Total Amount:", 1, 0, "L", fill=True)

pdf.cell(100, 10, f"${total\_amount:.2f}", 1, 1, "L")

pdf.output(pdf\_file)

def combine\_invoices(order\_ids):

merger = PdfMerger()

for order\_id in order\_ids:

merger.append(f"invoice\_{order\_id}.pdf")

merger.write("merged\_invoices.pdf")

merger.close()

def process\_csv(csv\_file):

order\_ids = []

with open(csv\_file, newline='') as file:

reader = csv.DictReader(file)

for order in reader:

create\_invoice(order)

order\_ids.append(order['Order ID'])

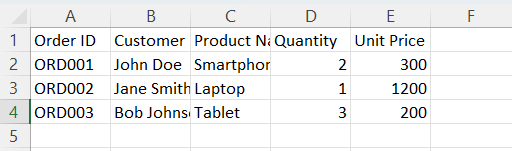
combine\_invoices(order\_ids)

csv\_file = input("Enter the CSV file name (e.g., orders.csv): ")

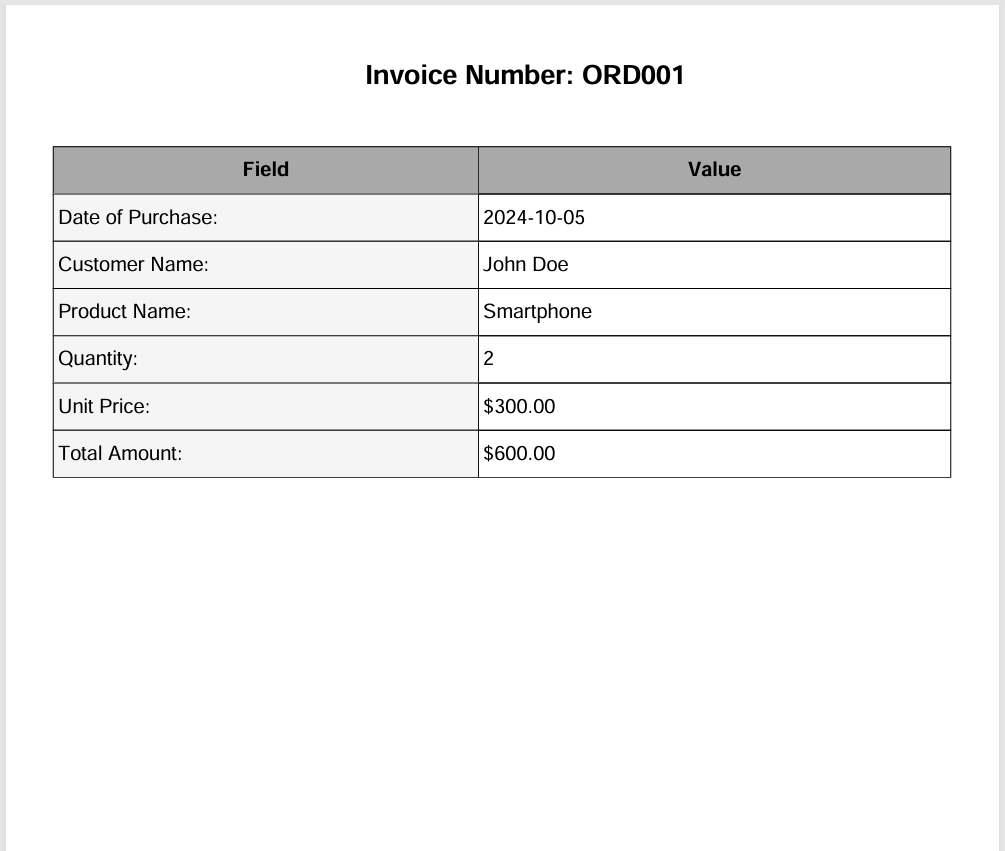
process\_csv(csv\_file)

**OUTPUT:**

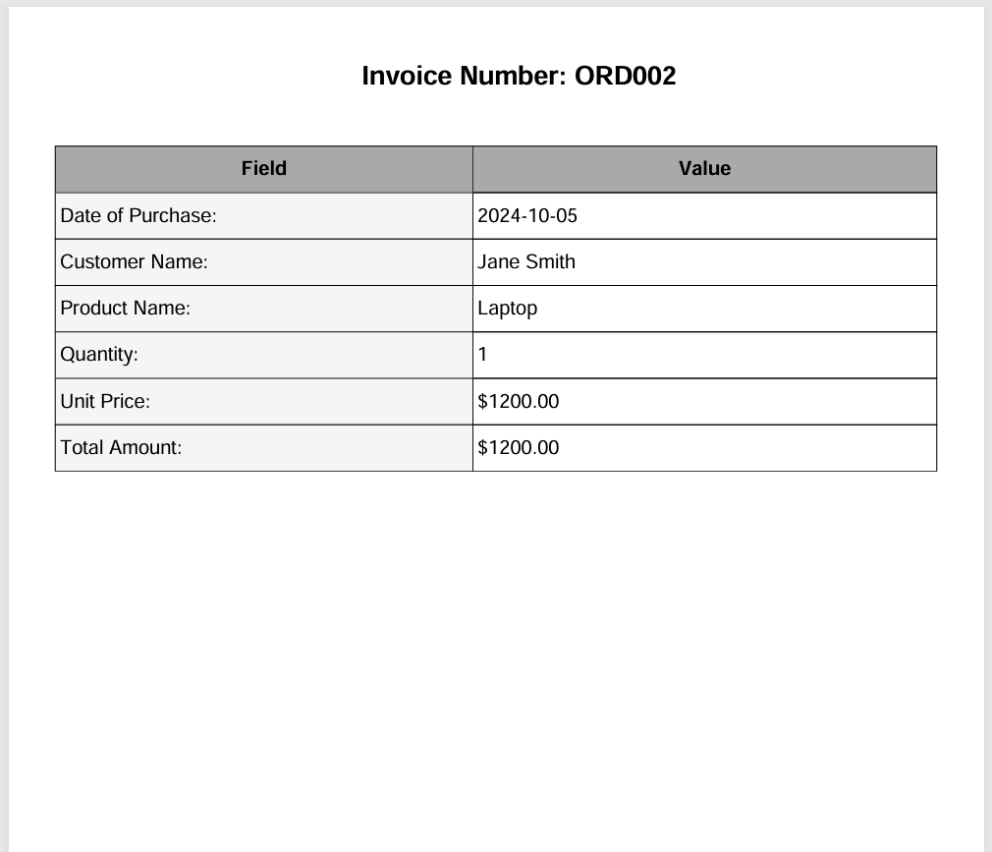
data.csv



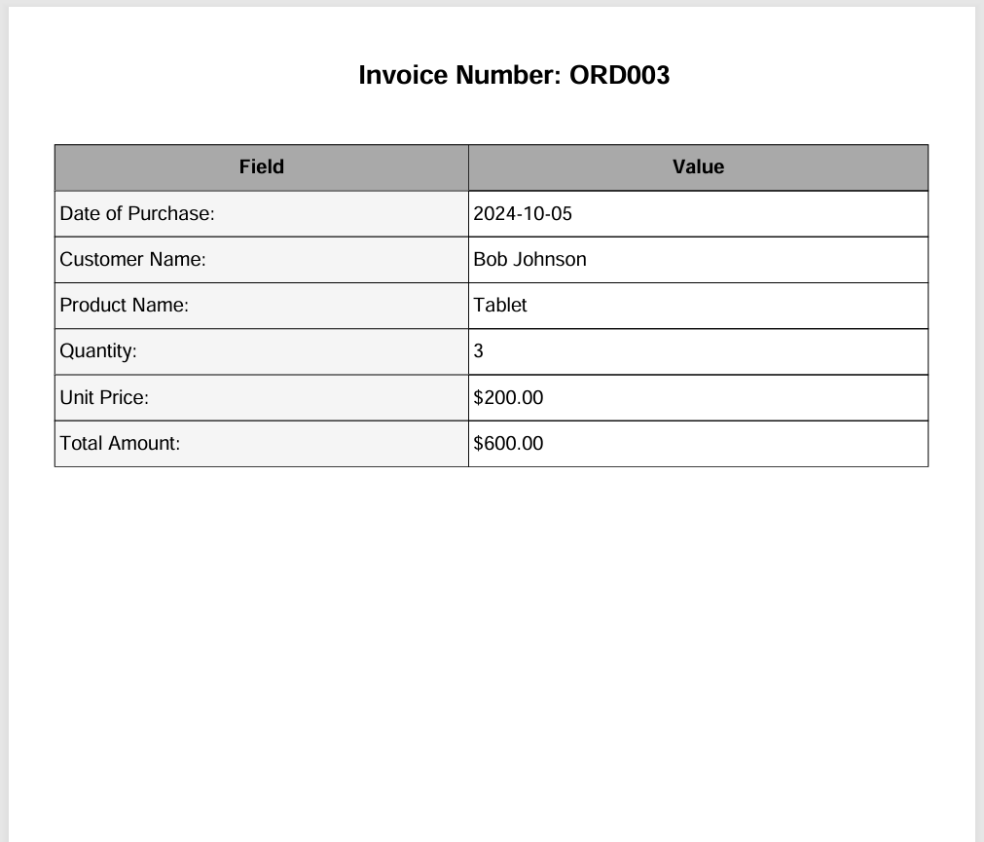
Invoice 1:



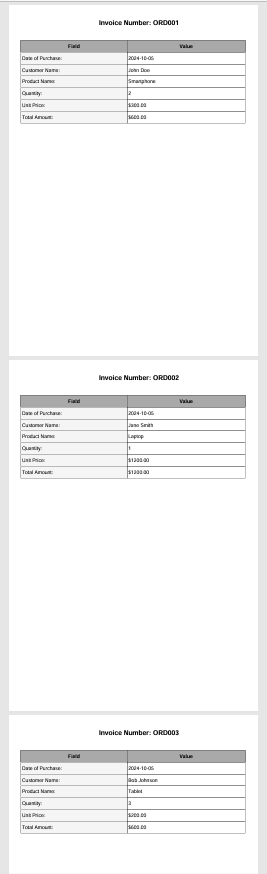
Invoice 2:



Invoice 3:



Merged Invoices:



# 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.

**CODE:**

import hashlib

import os

import logging

from datetime import datetime

logging.basicConfig(filename='usermanagement.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

def hash\_password(password):

return hashlib.sha256(password.encode()).hexdigest()

def read\_user\_data(file\_path='user\_Data.txt'):

if not os.path.exists(file\_path):

return {}

user\_data = {}

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

for line in f.readlines():

username, hashed\_password = line.strip().split(':')

user\_data[username] = hashed\_password

return user\_data

def write\_user\_data(username, hashed\_password, file\_path='user\_Data.txt'):

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

f.write(f'{username}:{hashed\_password}\n')

def sign\_up():

username = input("Enter username: ")

password = input("Enter password: ")

user\_data = read\_user\_data()

if username in user\_data:

print("Username already exists.")

logging.warning(f"Attempted registration with existing username: {username}")

return

hashed\_password = hash\_password(password)

write\_user\_data(username, hashed\_password)

print("Sign up successful.")

logging.info(f"{username} registered.")

def log\_in():

username = input("Enter username: ")

password = input("Enter password: ")

user\_data = read\_user\_data()

if username not in user\_data:

print("Username not found.")

logging.warning(f"Unsuccessful login attempt: {username}")

return

hashed\_password = hash\_password(password)

if user\_data[username] == hashed\_password:

print("Login successful.")

logging.info(f"{username} successfully logged in.")

else:

print("Incorrect password.")

logging.warning(f"Unsuccessful login attempt: {username}")

def main():

while True:

print("\nUSER MANAGEMENT SYSTEM")

print("Menu:")

print("1. Sign Up")

print("2. Log in")

print("3. Exit")

choice = input("Enter choice: ")

if choice == '1':

sign\_up()

elif choice == '2':

log\_in()

elif choice == '3':

print("Exiting...")

break

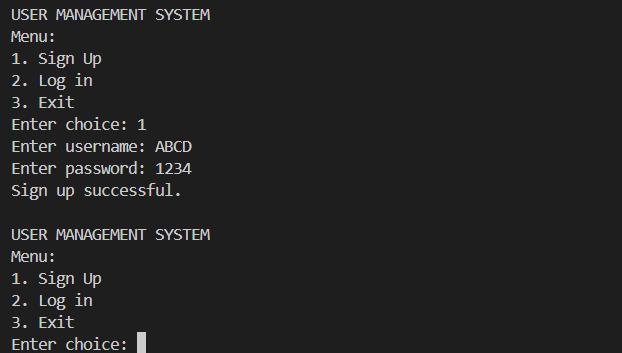
else:

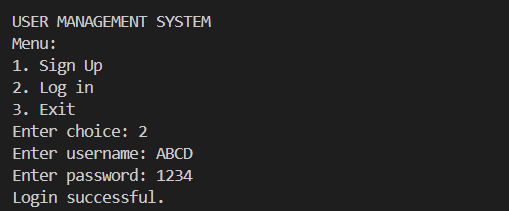
print("Invalid choice. Try again.")

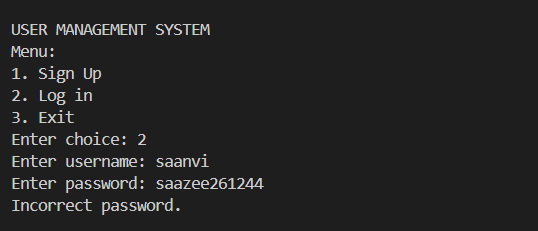
if \_name\_ == "\_main\_":

main()

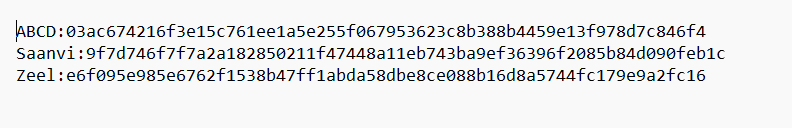
**OUTPUT:**

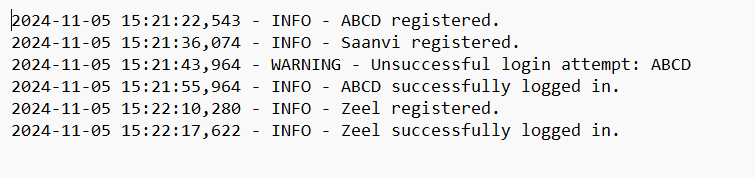
****

****









# 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.

**CODE:**

from PIL import Image, ImageFilter, ImageOps

import matplotlib.pyplot as plt

import numpy as np

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 = 'python\_image4.jpeg'

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)

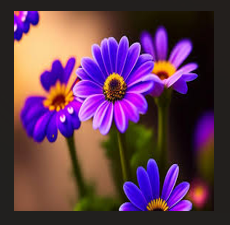
plot\_histogram\_separate\_bars(image)

OUTPUT:

1. Main Image:



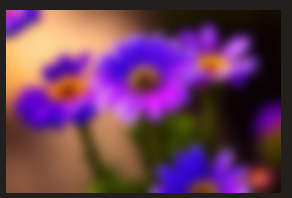
2. Resized Image:



3. GrayScale:



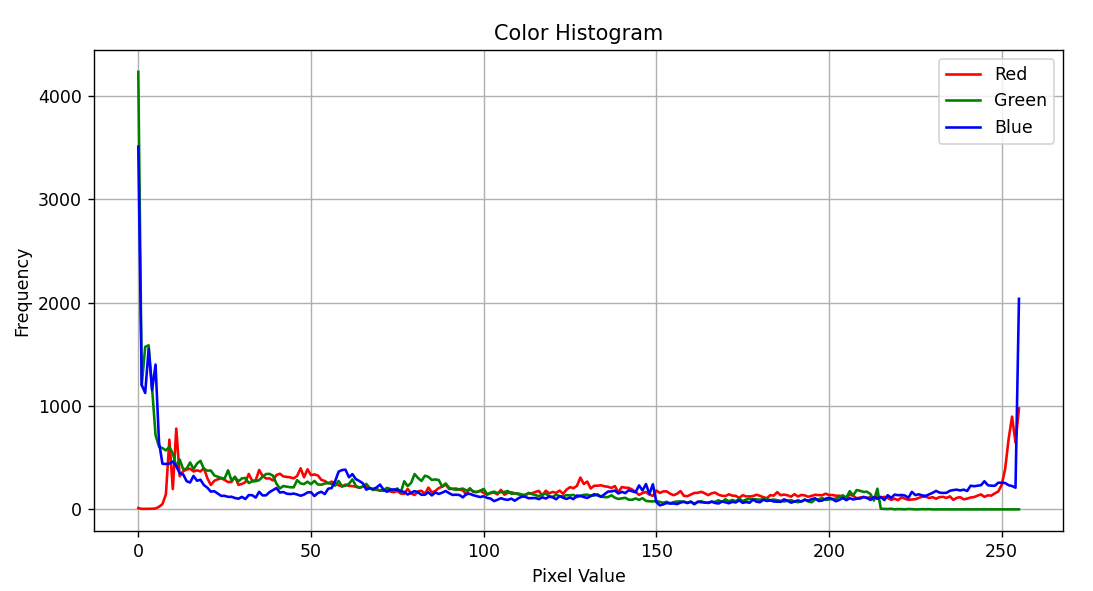
4.Guassian Blur:



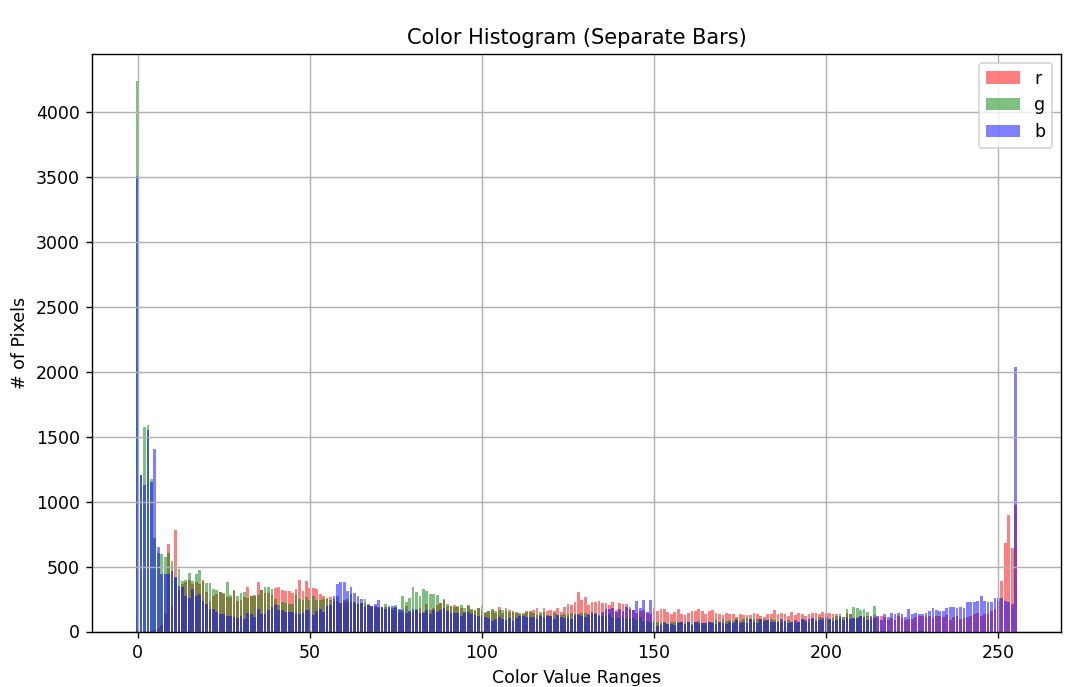
5.Edge Detection:



6. Graph:



7. Histogram:



**Part 2: Audio Data Processing**

1. **Audio Loading and Visualization:** Enable users to load an audio file and visualize its waveform using a suitable library like Matplotlib.
2. **Audio Manipulation:** Implement at least two audio manipulation operations, such as:
   * Changing the playback speed or pitch.
   * Applying audio effects (e.g., echo, reverb).
   * Clipping or trimming audio segments.
   * Adding silence or noise to the audio.
3. **Spectral Analysis:** Implement a feature that calculates and displays the spectrogram of the loaded audio. Allow users to analyze and manipulate spectral data.

# 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.

The code on which analysis is to be performed:

import pandas as pd

# Load Train Data

def load\_train\_data(file\_path):

df = pd.read\_csv(file\_path)

df.columns= df.columns.str.strip()

return df

# Load Passenger Data

def load\_passenger\_data(file\_path):

df = pd.read\_csv(file\_path)

df.columns = df.columns.str.strip()

return df

# Process Bookings and Update Availability

def process\_bookings(trains\_df, passengers\_df):

booking\_failures = []

confirmed\_bookings = []

for \_, row in passengers\_df.iterrows():

passenger\_name = row['Passenger Name']

train\_id = row['Train ID']

num\_tickets = row['Number of Tickets']

if len(str(train\_id)) != 4:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Invalid train number {train\_id}.")

continue

train = trains\_df[trains\_df['Train ID'] == train\_id]

if not train.empty:

available\_seats = train.iloc[0]['Available Seats']

if not isinstance(num\_tickets, int) or any(char.isdigit() for char in passenger\_name): booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Invalid input: number of tickets should be an integer and passenger name should not contain numbers.")

continue

if available\_seats >= num\_tickets:

# Update available seats in trains\_df

trains\_df.loc[trains\_df['Train ID'] == train\_id, 'Available Seats'] -= num\_tickets

# Add to confirmed bookings

fare = num\_tickets \* train.iloc[0]['Total fare']

confirmed\_bookings.append({

'Passenger Name': passenger\_name,

'Train ID': train\_id,

'Number of Tickets': num\_tickets,

'Fare': fare

})

else:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Available seats are only {available\_seats}.")

else:

booking\_failures.append(f"Booking not confirmed for {passenger\_name}. Train ID {train\_id} not found.")

# Convert confirmed bookings list to DataFrame

confirmed\_bookings\_df = pd.DataFrame(confirmed\_bookings)

return trains\_df, confirmed\_bookings\_df, booking\_failures

# Generate Reports

def generate\_reports(trains\_df, confirmed\_bookings):

# Report 1: Train Details

train\_report = trains\_df[['Train ID', 'Train Name', 'Source Station', 'Destination Station','Total Seats', 'Available Seats']]

# Report 2: Total Revenue Earned Per Passenger

revenue\_per\_passenger = confirmed\_bookings.groupby(['Passenger Name', 'Train ID']).agg({

'Number of Tickets': 'sum',

'Fare': 'sum'

}).reset\_index()

revenue\_per\_passenger.rename(columns={'Number of Tickets': 'Bookings', 'Fare': 'Total Revenue'}, inplace=True)

return train\_report, revenue\_per\_passenger

# Main program

def main():

# Get file paths from the user

trains\_file = input("Enter the path to the trains CSV file: ")

passengers\_file = input("Enter the path to the passengers CSV file: ")

# Load data

trains\_df = load\_train\_data(trains\_file)

passengers\_df = load\_passenger\_data(passengers\_file)

# Process bookings

trains\_df, confirmed\_bookings, booking\_failures = process\_bookings(trains\_df, passengers\_df)

# Print booking results

print("Booking Confirmations:")

for \_, row in confirmed\_bookings.iterrows():

print(f"Booking confirmed for {row['Passenger Name']}. Total Fare: ${row['Fare']:.2f}.")

# Print failed bookings

print("\nBooking Failures:")

for failure in booking\_failures:

print(failure)

# Generate reports

train\_report, revenue\_per\_passenger = generate\_reports(trains\_df, confirmed\_bookings)

print("\nREPORT 1: TRAIN DETAILS:")

print(train\_report.to\_string(index=False))

print("\nREPORT 2: TOTAL REVENUE EARNED PER PASSENGER:")

print(revenue\_per\_passenger.to\_string(index=False))

trains\_df.to\_csv(trains\_file, index=False)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Analysis on the code:

The application is a train reservation system that handles tasks like loading data, processing bookings, and generating reports. User interaction is limited to providing file paths for data and viewing the booking confirmations or failures and reports.

The critical components and functions are:

* load\_train\_data and load\_passenger\_data: Load CSV files containing train and passenger data.
* process\_bookings: Processes booking requests, checks seat availability, and updates records.
* generate\_reports: Generates reports on train details and revenue per passenger.

Identifying Decision points:

**Conditional statements** within process\_bookings decide booking success or failure based on criteria like:

* Valid train ID format
* Availability of seats
* Input data validity (numeric ticket counts and string-only names)

**File loading paths** in main() are another decision point, as they can determine if files are found and loaded properly.

External Interactions are:

* **File Reading/Writing**: Both load\_train\_data and load\_passenger\_data read data from CSV files. The main() function rewrites the updated data back to the file.
* Potential logging at these points helps identify if files are loaded or saved correctly, along with any errors (like missing files).

Identifying Exceptional handling:

Currently, there is no explicit exception handling. However, logging can still be applied where exceptions are likely:

* In file I/O operations (e.g., loading CSV files in load\_train\_data, load\_passenger\_data).
* Within process\_bookings, particularly when parsing data types or updating records.

Identifying loops and iteration:

* **process\_bookings iterates** over each passenger record:
* Logging can track each booking attempt, confirm whether it succeeds or fails, and detail reasons for failures.

In the report generation stage, we can log a summary of the generated reports.

Identifying inputs and outputs:

* **Inputs**: File paths for train and passenger data in main() are key inputs. Logging them can help validate user-provided paths and assist if an incorrect path is entered.
* **Outputs**: The booking confirmation and failure messages, as well as the generated reports. Logging these outputs provides a record of actions completed by the application.

Identifying troubleshooting points:

* **Data Processing and Validation in process\_bookings**: This function is prone to errors if data is not as expected. Logging here helps in debugging issues like invalid ticket counts or train IDs.
* **File Saving**: After generating the reports, the data is saved back to the CSV files. Logging a success message (or an error if saving fails) would be useful.

Identifying User actions:

**User Inputs** in main(): When users provide file paths, logging these actions can help track any issues related to incorrect file locations.

Logging levels and the points where it should be applied:

1. **DEBUG**:
   * Begin of each function to track function execution.
   * Inside process\_bookings for each booking attempt, to trace seat availability, ticket count, etc.
   * Inside loops in generate\_reports to track report rows as they are processed.
2. **INFO**:
   * When a file loads successfully in load\_train\_data and load\_passenger\_data.
   * Each confirmed booking, detailing the passenger name, train ID, and total fare.
3. **WARNING**:
   * Invalid file paths or missing files in main().
   * Any unexpected format or type issues in process\_bookings, such as non-integer ticket counts.
4. **ERROR**:
   * Failed bookings with details on why (e.g., insufficient seats, invalid train ID).
   * Issues with file saving, including the file path and error details.
5. **CRITICAL**:
   * Any system-critical issue that prevents bookings from processing, such as missing or corrupted data files.

This strategy helps create a comprehensive, clear logging structure for effective monitoring and debugging in your train reservation system.