

## **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.

- Text Files –

Lab 1 > feedbacks >  review1.txt

```
1 P1234567890 123456 2024-07-25 4 "Great product, very satisfied with the quality!"
2 P0987654321 2024-07-24 5 "Excellent service and fast delivery!"
3 P1112223334 345678 2024-07-23 3 "Product arrived late but good quality overall."
4 P5556667778 456789 2024-07-22 2 "Disappointed with the product, doesn't work as expected."
5 P4443332221 567890 2024-07-21 5 "Absolutely love it! Works perfectly."
```

Lab 1 > feedbacks >  review2.txt

```
1 P1112223334 678901 2024-07-20 4 "Good product, meets expectations."
2 P0987654321 789012 2024-07-19 1 "Terrible experience, item arrived damaged."
3 P5556667778 890123 2024-07-18 3 "Average product, could be better."
4 P1234567890 901234 2024-07-17 5 "Highly recommended, great value for money!"
5 P4443332221 345678 2024-07-16 4 "Arrived sooner than expected, very pleased."
```

Lab 1 > feedbacks >  review3.txt

```
1 P1234567890 112233 2024-07-15 2 "Not satisfied with the product quality, expected better."
2 P0987654321 224466 2024-07-14 5 "Amazing product, exceeded my expectations!"
3 P1112223334 335577 2024-07-13 3 "Decent product for the price, but delivery was slow."
4 P5556667778 448899 2024-07-12 4 "Good value for money, works as described."
5 P4443332221 556677 2024-07-11 1 "Very poor quality, broke after first use."
```

- Code –

```
import os
from collections import defaultdict

class Reviews:
    def __init__(self, productid, cid, date, rating, text):
        self.productid = productid
        self.cid = cid
        self.date = date
        try:
            self.rating = int(rating)
        except ValueError:
            self.rating = 0 # Default value if conversion fails
        self.text = text

def read_review_from_file(filename):
    valid = 0
    invalid = 0
    reviews = []

    with open(filename, 'r') as file:
        for line in file:
            data = line.strip().split(maxsplit=4)
            if len(data) == 5:
                productid, cid, date, rating, text = data
                try:
                    review = Reviews(productid, cid, date, rating, text)
```

```

        if review.rating < 1 or review.rating > 5:
            raise ValueError("Rating out of bounds")
        reviews.append(review)
        valid += 1
    except ValueError:
        invalid += 1
    else:
        invalid += 1

    return reviews, valid, invalid

def calculate_average_ratings(reviews):
    product_rating = defaultdict(list)
    for review in reviews:
        product_rating[review.productid].append(review.rating)

    average_ratings = {productid: sum(ratings) / len(ratings) for productid,
ratings in product_rating.items()}
    return average_ratings

folder_path = "D:/5th Lab/Python/Lab 1/feedbacks"
all_reviews = []
total_valid = 0
total_invalid = 0

for filename in os.listdir(folder_path):
    if filename.endswith('.txt'):
        filepath = os.path.join(folder_path, filename)
        reviews, valid, invalid = read_review_from_file(filepath)
        all_reviews.extend(reviews)
        total_valid += valid
        total_invalid += invalid

average_ratings = calculate_average_ratings(all_reviews)

top_3_products = []
for productid, avg_rating in average_ratings.items():
    top_3_products.append((productid, avg_rating))

n = len(top_3_products)
for i in range(n):
    for j in range(0, n-i-1):
        if top_3_products[j][1] < top_3_products[j+1][1]:
            top_3_products[j], top_3_products[j+1] = top_3_products[j+1],
top_3_products[j]
top_3_products = top_3_products[:3]

with open("summary.txt", "w") as summary_file:

```

```

summary_file.write(f"Total Reviews Processed: {total_valid +
total_invalid}\n")
summary_file.write(f"Total Valid Reviews: {total_valid}\n")
summary_file.write(f"Total Invalid Reviews: {total_invalid}\n")
summary_file.write("Top 3 Products by Average Rating:\n")
for productid, avg_rating in top_3_products:
    summary_file.write(f"Product ID: {productid}, Average Rating:
{avg_rating:.2f}\n")

print(f"Total Valid Reviews: {total_valid}")
print(f"Total Invalid Reviews: {total_invalid}")
print("Average Ratings by Product:")
for productid, avg_rating in top_3_products:
    print(f"Product ID: {productid}, Average Rating: {avg_rating:.2f}")


```

- Output –

```

Total Valid Reviews: 14
Total Invalid Reviews: 1
Average Ratings by Product:
Product ID: P1234567890, Average Rating: 3.67
Product ID: P1112223334, Average Rating: 3.33
Product ID: P4443332221, Average Rating: 3.33

```

 summary.txt

```

1 Total Reviews Processed: 15
2 Total Valid Reviews: 14
3 Total Invalid Reviews: 1
4 Top 3 Products by Average Rating:
5 Product ID: P1234567890, Average Rating: 3.67
6 Product ID: P1112223334, Average Rating: 3.33
7 Product ID: P4443332221, Average Rating: 3.33
8

```

## **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.


- Text Files –

Lab 2 >  trains.csv

```

1 TrainId,TrainName,SourceStation,DestinationStation,TotalSeats,FarePerTicket
2 T001,Train1,SS1,DD1,100,200
3 T002,Train2,SS2,DD2,100,300
4 T003,Train3,SS3,DD3,100,250
5 T004,Train4,SS4,DD4,100,600
6 T005,Train5,SS5,DD5,100,450
7 T006,Train6,SS6,DD6,100,500

```

Lab 2 >  passengers.csv

```

1 PassengerName,TrainId,NumberofTickets
2 Xyz,T002,4
3 Abc,T001,400
4 Mno,T002,4
5 Pqr,T003,4
6 Lkj,T004,4
7 Ajk,T005,100
8 Kah,T005,4
9 Oka,T006,4

```

## Code –

```

import csv
from collections import defaultdict

trains = {}
passenger_bookings = defaultdict(list)
train_revenue = defaultdict(int)

def load_train_data(filename):
    # Load train data from CSV file into the 'trains' dictionary.
    with open(filename, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            train_id = row['TrainId']
            trains[train_id] = {
                'Train Name': row['TrainName'],
                'Source Station': row['SourceStation'],
                'Destination Station': row['DestinationStation'],
                'Total Seats': int(row['TotalSeats']),
                'Available Seats': int(row['TotalSeats']),
                'Fare Per Ticket': float(row['FarePerTicket'])
            }

def load_passenger_data(filename):
    # Load passenger booking requests from a CSV file and attempt to book
    # tickets.
    with open(filename, 'r') as file:

```

```

        reader = csv.DictReader(file)
        for row in reader:
            passenger_name = row['PassengerName']
            train_id = row['TrainId']
            num_tickets = int(row['NumberofTickets'])
            book_ticket(passenger_name, train_id, num_tickets)

def check_seat_availability(train_id, num_tickets):
    # Check if there are enough seats available on the specified train.
    if train_id not in trains:
        raise ValueError("Invalid train ID")
    if trains[train_id]['Available Seats'] >= num_tickets:
        return True
    else:
        return False

def book_ticket(passenger_name, train_id, num_tickets):
    # Book tickets for a passenger if seats are available.
    if check_seat_availability(train_id, num_tickets):
        trains[train_id]['Available Seats'] -= num_tickets
        passenger_bookings[train_id].append((passenger_name, num_tickets))
        fare_per_ticket = trains[train_id]['Fare Per Ticket']
        total_fare = num_tickets * fare_per_ticket
        train_revenue[train_id] += total_fare
        print(f"Booking confirmed for {passenger_name} on train {train_id}.
Total fare: Rupees {total_fare}")
    else:
        print(f"Insufficient seats available for {passenger_name} on train
{train_id}")

def generate_train_report():
    # Generate and print a report of all trains.
    print("Train Report:")
    for train_id, details in trains.items():
        print(f"Train ID: {train_id}, Name: {details['Train Name']}, Source:
{details['Source Station']}, Destination: {details['Destination Station']},
Available Seats: {details['Available Seats']}, Fare Per Ticket: Rupees
{details['Fare Per Ticket']}")

def generate_revenue_report():
    # Generate and print a report of total revenue for each train.
    print("Revenue Report:")
    for train_id, revenue in train_revenue.items():
        print(f"Train ID: {train_id}, Total Revenue: Rupees {revenue}")

def calculate_total_revenue():
    # Calculate and return the total revenue generated.
    total_revenue = sum(train_revenue.values())

```

```
    return total_revenue

load_train_data('D:/5th Lab/Python/Lab 2/trains.csv')
load_passenger_data('D:/5th Lab/Python/Lab 2/passengers.csv')

generate_train_report()
generate_revenue_report()

total_revenue = calculate_total_revenue()
print(f"Total Revenue: Rupees {total_revenue}")
```



- Output –

```
Booking confirmed for Xyz on train T002. Total fare: Rupees 1200.0
Insufficient seats available for Abc on train T001
Booking confirmed for Mno on train T002. Total fare: Rupees 1200.0
Booking confirmed for Pqr on train T003. Total fare: Rupees 1000.0
Booking confirmed for Lkj on train T004. Total fare: Rupees 2400.0
Booking confirmed for Ajk on train T005. Total fare: Rupees 45000.0
Insufficient seats available for Kah on train T005
Booking confirmed for Oka on train T006. Total fare: Rupees 2000.0
```

Train Report:

```
Train ID: T001, Name: Train1, Source: SS1, Destination: DD1, Av Rupees 200.0
Train ID: T002, Name: Train2, Source: SS2, Destination: DD2, Available Seats: 92, Fare Per Ticket:
Rupees 300.0
Train ID: T003, Name: Train3, Source: SS3, Destina DD3, Available Seats: 96, Fare Per Ticket: Rupee
s DD3, Available Seats: 96, Fare Per Ticket: Rupees 250.0
Train ID: T004, Name: Train4, Source: SS4, Destination: DD4, Available Seats: 96, Fare Per Ticket:
Rupees 600.0
  DD3, Available Seats: 96, Fare Per Ticket: Rupees 250.0
Train ID: T004, Name: Train4, Source: SS4, Destination: DD4, Available Seats: 96, Fare Per Ticket:
Rupees 600.0
  DD3, Available Seats: 96, Fare Per Ticket: Rupees 250.0
Train ID: T004, Name: Train4, Source: SS4, Destination: DD4, Available Seats: 96, Fare Per Ticket:
Rupees 600.0
  DD3, Available Seats: 96, Fare Per Ticket: Rupees 250.0
Train ID: T005, Name: Train5, Source: SS5, Destination: DD5, Available Seats: 0, Fare Per Ticket: R
upees 450.0
Train ID: T006, Name: Train6, Source: SS6, Destination: DD6, Available Seats: 96, Fare Per Ticket:
Rupees 500.0
```

Revenue Report:

```
Train ID: T002, Total Revenue: Rupees 2400.0
Train ID: T003, Total Revenue: Rupees 1000.0
Train ID: T004, Total Revenue: Rupees 2400.0
Train ID: T005, Total Revenue: Rupees 45000.0
Train ID: T006, Total Revenue: Rupees 2000.0
Total Revenue: Rupees 52800.0
```

## **Lab Assignment 3: 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, ImageEnhance
import matplotlib.pyplot as plt

# 1. Image Loading and Display
def load_and_display_image(image_path):
    image = Image.open(image_path)
    plt.imshow(image)
    plt.axis('off') # Hide axes
    plt.show()
    return image

# 2. Image Manipulation
```

```

# Applying Gaussian Blur
def apply_gaussian_blur(image, radius=2):
    blurred_image = image.filter(ImageFilter.GaussianBlur(radius))
    return blurred_image

# Applying Edge Detection
def apply_edge_detection(image):
    edge_detected_image = image.filter(ImageFilter.FIND_EDGES)
    return edge_detected_image

# Changing Image Dimensions (Resizing)
def resize_image(image, width, height):
    resized_image = image.resize((width, height))
    return resized_image

# Cropping the Image
def crop_image(image, left, top, right, bottom):
    cropped_image = image.crop((left, top, right, bottom))
    return cropped_image

# Adjusting Brightness
def adjust_brightness(image, factor=1.5):
    enhancer = ImageEnhance.Brightness(image)
    brightened_image = enhancer.enhance(factor)
    return brightened_image

# Adjusting Contrast
def adjust_contrast(image, factor=1.5):
    enhancer = ImageEnhance.Contrast(image)
    contrasted_image = enhancer.enhance(factor)
    return contrasted_image

# Adjusting Saturation
def adjust_saturation(image, factor=1.5):
    enhancer = ImageEnhance.Color(image)
    saturated_image = enhancer.enhance(factor)
    return saturated_image

# Converting to Grayscale
def convert_to_grayscale(image):
    grayscale_image = image.convert("L")
    return grayscale_image

def display_all_images(images, titles):
    plt.figure(figsize=(15, 10))
    for i, (image, title) in enumerate(zip(images, titles)):
        plt.subplot(3, 3, i + 1)

```

```

        plt.imshow(image)
        plt.title(title)
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# 3. Histogram Analysis
def display_histogram(image):
    plt.figure(figsize=(10, 5))
    color_channels = ('r', 'g', 'b')

    for i, color in enumerate(color_channels):
        histogram = image.histogram()[i * 256:(i + 1) * 256]
        plt.plot(histogram, color=color)

    plt.title('Color Histogram')
    plt.xlabel('Pixel value')
    plt.ylabel('Frequency')
    plt.show()

# Load and display the image
image_path = r"D:/5th Lab/Python/Lab 3/image.jpg"
image = load_and_display_image(image_path)

# Apply Gaussian Blur
blurred_image = apply_gaussian_blur(image, radius=3)
plt.imshow(blurred_image)
plt.title('Gaussian Blurred Image')
plt.axis('off')

# Apply Edge Detection
edge_image = apply_edge_detection(image)
plt.imshow(edge_image)
plt.title('Edge Detected Image')
plt.axis('off')

# Resize Image
resized_image = resize_image(image, width=200, height=200)
plt.imshow(resized_image)
plt.title('Resized Image')
plt.axis('off')

# Crop Image
cropped_image = crop_image(image, left=50, top=50, right=250, bottom=250)
plt.imshow(cropped_image)
plt.title('Cropped Image')
plt.axis('off')

```

```
# Adjust Brightness
brightened_image = adjust_brightness(image, factor=1.8)
plt.imshow(brightened_image)
plt.title('Brightened Image')
plt.axis('off')

# Adjust Contrast
contrasted_image = adjust_contrast(image, factor=1.8)
plt.imshow(contrasted_image)
plt.title('Contrasted Image')
plt.axis('off')

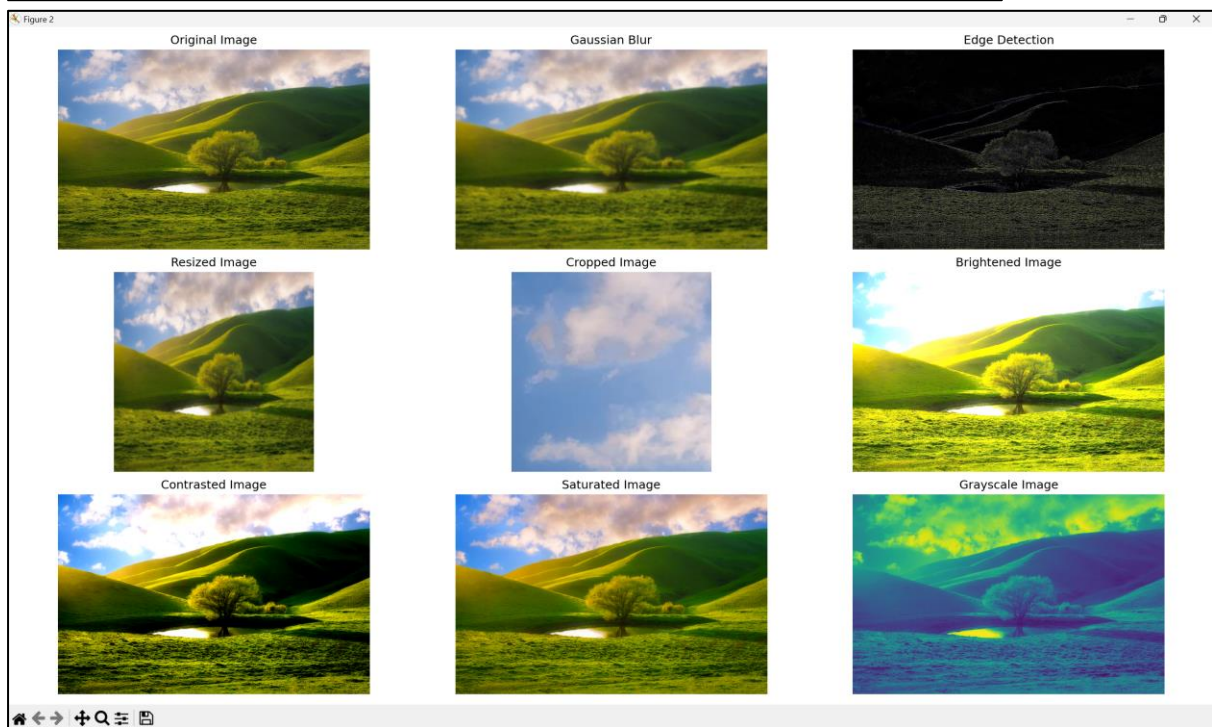
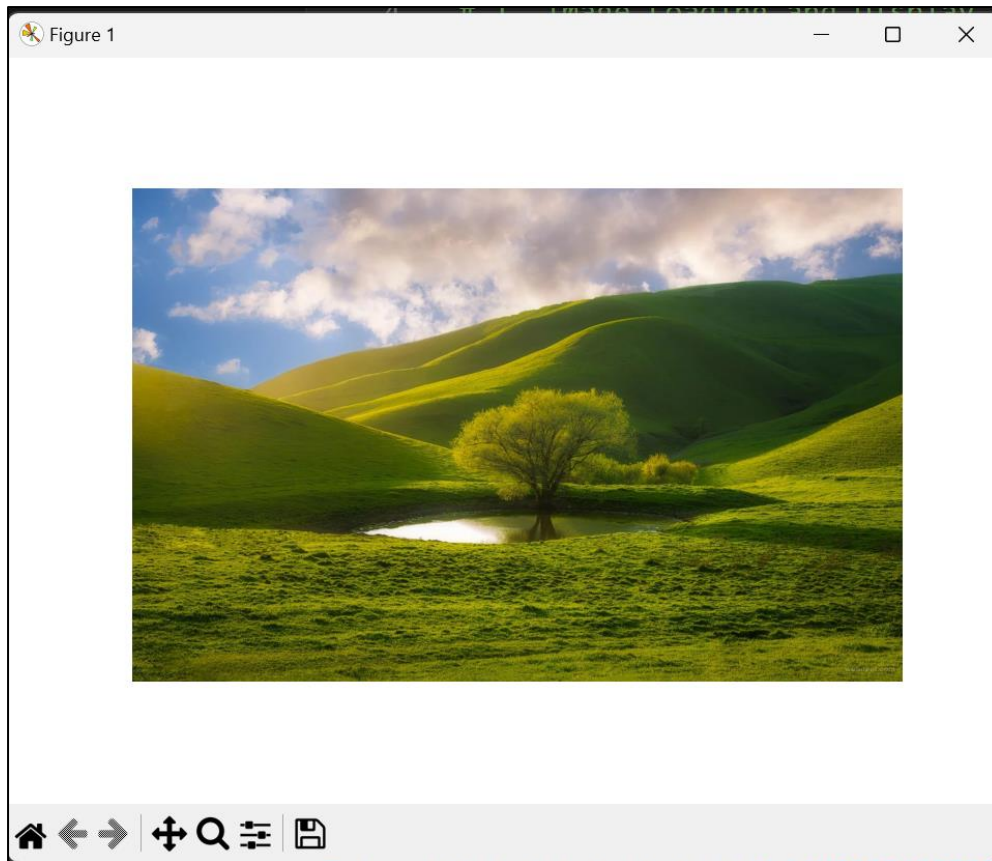
# Adjust Saturation
saturated_image = adjust_saturation(image, factor=1.8)
plt.imshow(saturated_image)
plt.title('Saturated Image')
plt.axis('off')

# Convert to Grayscale
grayscale_image = convert_to_grayscale(image)
plt.imshow(grayscale_image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

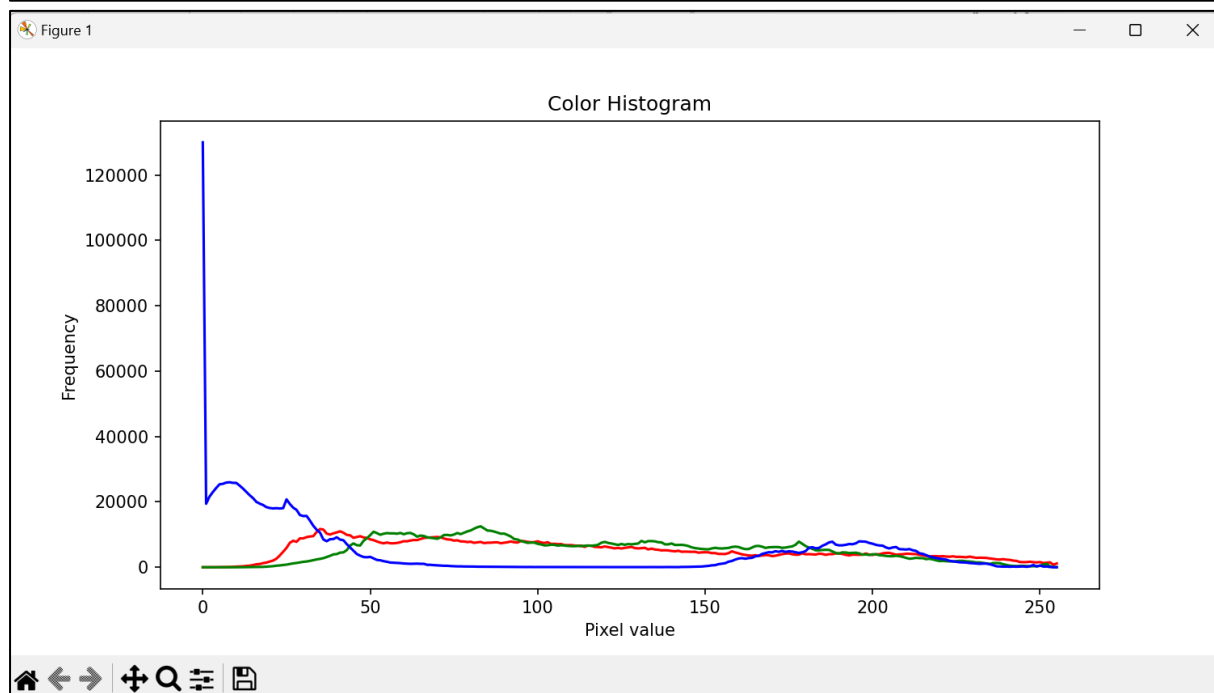
images = [ image, blurred_image, edge_image, resized_image,
cropped_image, brightened_image, contrasted_image, saturated_image,
grayscale_image]
titles = ["Original Image", "Gaussian Blur", "Edge Detection", "Resized
Image", "Cropped Image", "Brightened Image", "Contrasted Image", "Saturated
Image", "Grayscale Image"]
display_all_images(images, titles)

display_histogram(image)
```

- Output –







## **Lab Assignment 4: Data Analysis**

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

1. Customer ID: Unique identifier for each customer.
2. Total Amount Spent: The total amount spent by each customer on the platform.
3. Total Items Purchased: The total number of items purchased by each customer.
4. Last Purchase Date: The date of the customer's most recent purchase.
5. Average Purchase Value: 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 behaviour. 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 ecommerce 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.



## Code –

```

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

# 1. Data Loading
try:
    df = pd.read_csv("D:/5th Lab/Python/Lab 4/Online Retail.csv",
encoding='ISO-8859-1')
except FileNotFoundError:
    print("The file 'Online Retail.csv' was not found. Please check the file
path.")
    raise

# 2. Data Cleaning
df.dropna(subset=['CustomerID'], inplace=True) # Drop rows with missing
CustomerID
df.drop_duplicates(inplace=True) # Remove duplicates

# Add a 'TotalAmountSpent' column (Quantity * UnitPrice)
df['TotalAmountSpent'] = df['Quantity'] * df['UnitPrice']

# 3. Feature Engineering
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], errors='coerce') #
Convert InvoiceDate to datetime format

# Calculate Recency (days since the last purchase)
df['Recency'] = (pd.to_datetime('today') - df['InvoiceDate']).dt.days

# Calculate Total Amount Spent and Total Items Purchased by each customer
customer_data = df.groupby('CustomerID').agg({
    'TotalAmountSpent': 'sum',
    'Quantity': 'sum',
    'Recency': 'min' # Use 'min' to get the most recent purchase
}).reset_index()

customer_data.rename(columns={'Quantity': 'TotalItemsPurchased'},
inplace=True)

# Calculate Average Purchase Value
customer_data['AveragePurchaseValue'] = customer_data['TotalAmountSpent'] /
customer_data['TotalItemsPurchased']

# 4. Handle Missing or Infinite Values
customer_data.replace([np.inf, -np.inf], np.nan, inplace=True)
customer_data.dropna(inplace=True)

```

```

# 5. Descriptive Statistics
print("Descriptive Statistics for TotalAmountSpent:")
print(customer_data['TotalAmountSpent'].describe())
print("\nDescriptive Statistics for TotalItemsPurchased:")
print(customer_data['TotalItemsPurchased'].describe())

# 6. Customer Segmentation using K-means Clustering
X = customer_data[['TotalAmountSpent', 'TotalItemsPurchased',
'AveragePurchaseValue', 'Recency']]

# Apply K-means clustering
kmeans = KMeans(n_clusters=4, random_state=42) # Specify n_init to avoid
warnings
customer_data['Segment'] = kmeans.fit_predict(X)

# 7. Visualization
plt.figure(figsize=(10, 6))
sns.scatterplot(x='TotalAmountSpent', y='TotalItemsPurchased', hue='Segment',
data=customer_data, palette='viridis')
plt.title('Customer Segmentation based on Total Amount Spent and Total Items
Purchased')
plt.show()

# 8. Customer Insights
insights = customer_data.groupby('Segment').mean().reset_index()
print("Customer Segment Insights:")
print(insights)

# 9. Customer Engagement Recommendations
for segment in insights['Segment']:
    print(f"\nSegment {segment} Recommendations:")
    if insights.loc[segment, 'TotalAmountSpent'] >
customer_data['TotalAmountSpent'].mean():
        print("- Targeted promotions and discounts to retain high spenders.")
    if insights.loc[segment, 'Recency'] > customer_data['Recency'].mean():
        print("- Re-engagement campaigns for inactive customers.")
    if insights.loc[segment, 'TotalItemsPurchased'] >
customer_data['TotalItemsPurchased'].mean():
        print("- Loyalty programs to reward frequent shoppers.")

```

- Output –

```
Descriptive Statistics for TotalAmountSpent:
count      3125.000000
mean       2434.927978
std        9657.659144
min        -1192.200000
25%        378.640000
50%        901.200000
75%        2114.330000
max        279489.020000
Name: TotalAmountSpent, dtype: float64

Descriptive Statistics for TotalItemsPurchased:
count      3125.000000
mean       1426.469760
std        5481.896263
min        -303.000000
25%        200.000000
50%        510.000000
75%        1283.000000
max        196719.000000
Name: TotalItemsPurchased, dtype: float64
```

```
Customer Segment Insights:
  Segment  CustomerID  TotalAmountSpent  TotalItemsPurchased  Recency  AveragePurchaseValue
0         0  15282.839935         1715.874614         1008.706818  4795.800325          2.461216
1         1  16374.000000         267963.755000        130420.500000  4691.500000          2.709990
2         2  15136.827586          39397.268621         25749.448276  4688.862069          1.814062
3         3  15288.600000         128969.264000         68665.200000  4661.800000          1.884204
```

## Segment 0 Recommendations:

- Re-engagement campaigns for inactive customers.

## Segment 1 Recommendations:

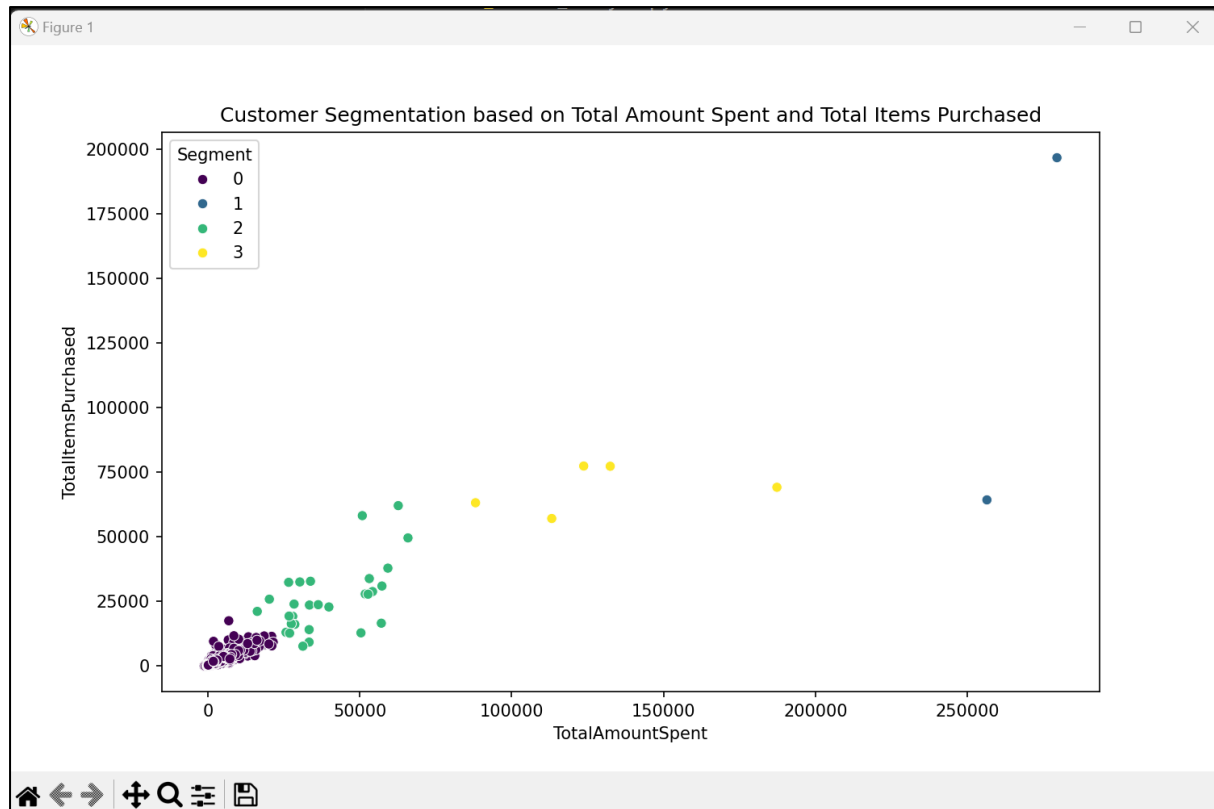
- Targeted promotions and discounts to retain high spenders.
- Loyalty programs to reward frequent shoppers.

## Segment 2 Recommendations:

- Targeted promotions and discounts to retain high spenders.
- Loyalty programs to reward frequent shoppers.

## Segment 3 Recommendations:

- Targeted promotions and discounts to retain high spenders.
- Loyalty programs to reward frequent shoppers.



## **Lab Assignment 5: Data Analysis**

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".

## Data Set –

```
Lab 5 > 📄 student_data.csv
1  NAME,MATHS,SCIENCE,ENGLISH
2  John,85,78,92
3  Alice,89,90,93
4  Bob,76,90,85
5  Eve,88,78,88
6  Charlie,90,85,87
7  Diana,79,82,80
8  Frank,95,92,94
9  Grace,87,83,89
10 Hank,73,75,78
11 Ivy,88,91,87
12 Jack,81,79,84
13 Karen,90,89,91
14 Leo,85,88,85
15 Mona,79,81,82
16 Nina,91,94,95
17 Oscar,77,74,80
18 Paul,83,87,86
19 Quinn,92,89,93
20 Rose,89,88,90
21 Steve,78,80,82
22 Tina,84,86,88
23 Uma,91,92,94
24 Vince,80,77,79
25 Wendy,86,84,83
26 Xander,93,90,94
27 Yara,78,80,81
28 Zane,85,87,88
```

## Code –

```
import csv

def read_student_data(input_file):
    """Reads student data from a CSV file and calculates average scores."""
    student_grade = {}
    with open(input_file, "r") as f:
        reader = csv.DictReader(f)
        for row in reader:
            name = row['NAME']
            math_score = int(row['MATHS'])
            science_score = int(row['SCIENCE'])
            english_score = int(row['ENGLISH'])
            avg_score = calculate_average(math_score, science_score,
english_score)
```

```
        student_grade[name] = avg_score
    return student_grade

def calculate_average(math_score, science_score, english_score):
    """Calculates the average score of three subjects."""
    return round(((math_score + science_score + english_score) / 3), 2)

def write_student_averages(output_file, student_grade):
    """Writes the student names and their average scores to an output file."""
    with open(output_file, "w") as of:
        fieldnames = ['NAME', 'AVERAGE SCORE']
        writer = csv.DictWriter(of, fieldnames=fieldnames)
        writer.writeheader()
        for name, avg_score in student_grade.items():
            writer.writerow({"NAME": name, "AVERAGE SCORE": avg_score})
    print("Average score calculated for all students and stored in file.")

def main():
    input_file = "D:/5th Lab/Python/Lab 5/student_data.csv"
    output_file = "student_data_average.txt"

    # Read student data and calculate average scores
    student_grade = read_student_data(input_file)

    # Write the calculated averages to an output file
    write_student_averages(output_file, student_grade)

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

## Output –

```
PS D:\5th Lab\Python> & "C:/Users/Katha Patel/AppData/Local/Programs/Python/Python312/python.exe"
d:/5th Lab/Python/Lab 5/average.py"
Average score calculated for all students and stored in file.
```

student_data_average.txt	student_data_average.txt
1 NAME,AVERAGE SCORE	27 Leo,86.0
2	28
3 John,85.0	29 Mona,80.67
4	30
5 Alice,90.67	31 Nina,93.33
6	32
7 Bob,83.67	33 Oscar,77.0
8	34
9 Eve,84.67	35 Paul,85.33
10	36
11 Charlie,87.33	37 Quinn,91.33
12	38
13 Diana,80.33	39 Rose,89.0
14	40
15 Frank,93.67	41 Steve,80.0
16	42
17 Grace,86.33	43 Tina,86.0
18	44
19 Hank,75.33	45 Uma,92.33
20	46
21 Ivy,88.67	47 Vince,78.67
22	48
23 Jack,81.33	49 Wendy,84.33
24	50
25 Karen,90.0	51 Xander,92.33



## **Lab Assignment 6: 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)

## Data Set –

```
Lab 6 > files > data1.csv
1 Date,Store ID,Product ID,Quantity sold
2 2024-01-01,S001,P1001,25
3 2024-01-01,S002,P1002,40
4 2024-01-02,S001,P1001,30
5 2024-01-02,S003,P1003,15
6 2024-01-03,S002,P1002,22
7 2024-01-03,S001,P1004,17
8 2024-01-04,S004,P1005,50
9 2024-01-05,S001,P1003,12
10 2024-01-05,S002,P1001,10
11 2024-01-06,S003,P1002,35
12 2024-01-06,S004,P1004,28
13 2024-01-07,S001,P1005,20
14 2024-01-07,S002,P1003,18
15 2024-01-08,S003,P1001,23
16 2024-01-09,S004,P1002,15
17 2024-01-10,S001,P1004,45
18 2024-01-10,S002,P1005,30
19 2024-01-11,S003,P1003,20
20 2024-01-12,S004,P1001,12

Lab 6 > files > data2.csv
1 Date,Store ID,Product ID,Quantity Sold
2 2024-01-13,S001,P1002,10
3 2024-01-14,S002,P1004,28
4 2024-01-15,S003,P1005,16
5 2024-01-16,S004,P1003,22
6 2024-01-17,S001,P1001,35
7 2024-01-18,S002,P1002,40
8 2024-01-19,S003,P1004,18
9 2024-01-20,S004,P1005,27
10 2024-01-21,S001,P1003,29
11 2024-01-22,S002,P1001,15
12 2024-01-23,S003,P1005,33
13 2024-01-24,S004,P1002,31
14 2024-01-25,S001,P1004,50
```

## Code –

```
import os
import pandas as pd

# Directory path where the sales data CSV files are stored
sales_data_dir = 'D:/5th Lab/Python/Lab 6/files'
product_names_file = 'D:/5th Lab/Python/Lab 6/product_names.csv'
output_file = 'sales_summary.csv'

# Read product names mapping file
product_names_df = pd.read_csv(product_names_file)

# Initialize an empty DataFrame to store aggregated sales data
all_sales_data = pd.DataFrame()

# Walk through the directory and read all sales CSV files
for root, dirs, files in os.walk(sales_data_dir):
    for file in files:
        if file.endswith('.csv'):
            file_path = os.path.join(root, file)
            sales_data = pd.read_csv(file_path)

            # Convert the 'Date' column to datetime for further analysis if
            # needed
            if 'Date' in sales_data.columns:
                sales_data['Date'] = pd.to_datetime(sales_data['Date'],
                                                    format='%Y-%m-%d', errors='coerce')

            # Append the current file's sales data to the aggregated data
```

```

        all_sales_data = pd.concat([all_sales_data, sales_data],
ignore_index=True)

# Check if there is data to process
if not all_sales_data.empty:
    # Group by Product ID to calculate total sales across all stores and
months
    if 'Product ID' in all_sales_data.columns and 'Quantity sold' in
all_sales_data.columns:
        total_sales = all_sales_data.groupby('Product ID')['Quantity
sold'].sum().reset_index()

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

        # Calculate the number of months in the data (unique year-months in
'Date')
        if 'Date' in all_sales_data.columns:
            all_sales_data['YearMonth'] =
all_sales_data['Date'].dt.to_period('M')
            months_count = all_sales_data['YearMonth'].nunique()

            # Calculate average sales per month for each product
            total_sales['Average Quantity Sold per Month'] =
total_sales['Quantity sold'] / months_count

            # Sort by total sales and get the top 5 best-selling products
            top_5_sales = total_sales.nlargest(5, 'Quantity sold')

            # Save the summary to a new CSV file
            top_5_sales[['Product ID', 'Product Name', 'Quantity sold',
'Average Quantity Sold per Month']].to_csv(output_file, index=False)

            print(f"Sales summary written to {output_file}")
        else:
            print("Error: 'Date' column is missing or incorrect in the data.")
    else:
        print("Error: Required columns 'Product ID' or 'Quantity sold' are
missing in the data.")
else:
    print("No sales data found.")

```

## Output –

```
PS D:\5th Lab\Python> & "C:/Users/Katha Patel/AppData/Local/Programs/Python/Python312/python.exe" "d:/5th Lab/Python/Lab 6/process.py"
Sales summary written to sales_summary.csv
```

```
x sales_summary.csv
1 Product ID,Product Name,Quantity sold,Average Quantity Sold per Month
2 P1002,Banana,112.0,112.0
3 P1001,Apple,100.0,100.0
4 P1005,Watermelon,100.0,100.0
5 P1004,Strawberry,90.0,90.0
6 P1003,Orange,65.0,65.0
7
```

## **Lab Assignment 7: JSON Handling**

You are working as a data scientist for a healthcare organization, and your team has been tasked with analysing 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:
3. Total confirmed cases.
4. Total deaths.
5. Total recovered cases.
6. Total active cases (total confirmed cases minus total deaths and total recovered).
7. Determine the top 5 countries with the highest number of confirmed cases and the lowest number of confirmed cases.
8. Generate a summary report in JSON format that includes the statistics for all countries and save it to a file named "covid19\_summary.json".

## Data Set –

```
Lab 7 > files > {} country1.json > ...
```

```
1  {
2      "country": "Country A",
3      "date": "2024-09-30",
4      "confirmed_cases": { "total": 1000, "new": 50 },
5      "deaths": { "total": 20, "new": 2 },
6      "recovered": { "total": 800, "new": 30 }
7  }
8
```

```
Lab 7 > files > {} country2.json > ...
```

```
1  {
2      "country": "Country B",
3      "date": "2024-09-30",
4      "confirmed_cases": { "total": 2000, "new": 100 },
5      "deaths": { "total": 40, "new": 5 },
6      "recovered": { "total": 1500, "new": 60 }
7  }
```

## Code –

```
import os
import json

# Function to read all JSON files in a given directory and subdirectories
def read_json_files(directory):
    covid_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 = json.load(f)
                    covid_data.append(data)
    return covid_data

# Function to calculate statistics for each country
def calculate_statistics(covid_data):
    country_stats = {}

    for data in covid_data:
        country = data['country']
```

```

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

            country_stats[country]['total_confirmed'] +=
data['confirmed_cases']['total']
            country_stats[country]['total_deaths'] += data['deaths']['total']
            country_stats[country]['total_recovered'] +=
data['recovered']['total']
            country_stats[country]['total_active'] = (
                country_stats[country]['total_confirmed'] -
                country_stats[country]['total_deaths'] -
                country_stats[country]['total_recovered']
            )

        return country_stats

# Function to determine top 5 countries with highest and lowest confirmed
cases
def top_5_countries_by_confirmed_cases(country_stats):
    sorted_by_confirmed = sorted(country_stats.items(), key=lambda x:
x[1]['total_confirmed'], reverse=True)
    highest_5 = sorted_by_confirmed[:5]
    lowest_5 = sorted_by_confirmed[-5:]

    return highest_5, lowest_5

# Function to generate summary report in JSON format
def generate_summary_report(country_stats, output_file):
    with open(output_file, 'w') as f:
        json.dump(country_stats, f, indent=4)
        print(f"Summary report saved to {output_file}")

def main():
    # Directory containing the JSON files
    directory = 'D:/5th Lab/Python/Lab 7/files'

    # Step 1: Read JSON files
    covid_data = read_json_files(directory)

    # Step 2: Calculate statistics for each country
    country_stats = calculate_statistics(covid_data)

    # Display statistics

```

```

for country, stats in country_stats.items():
    print(f"{country}:")
    print(f"  Total Confirmed Cases: {stats['total_confirmed']}")
    print(f"  Total Deaths: {stats['total_deaths']}")
    print(f"  Total Recovered: {stats['total_recovered']}")
    print(f"  Total Active Cases: {stats['total_active']}")
    print()

# Step 3: Determine top 5 countries with highest and lowest confirmed
cases
highest_5, lowest_5 = top_5_countries_by_confirmed_cases(country_stats)

print("Top 5 countries with highest confirmed cases:")
for country, stats in highest_5:
    print(f"{country}: {stats['total_confirmed']} confirmed cases")

print("\nTop 5 countries with lowest confirmed cases:")
for country, stats in lowest_5:
    print(f"{country}: {stats['total_confirmed']} confirmed cases")

# Step 4: Generate summary report in JSON format
generate_summary_report(country_stats, "covid19_summary.json")

if __name__ == "__main__":
    main()

```

## Output –

```

Country A:
  Total Confirmed Cases: 1000
  Total Deaths: 20
  Total Recovered: 800
  Total Active Cases: 180

Country B:
  Total Confirmed Cases: 2000
  Total Deaths: 40
  Total Recovered: 1500
  Total Active Cases: 460

Top 5 countries with highest confirmed cases:
Country B: 2000 confirmed cases
Country A: 1000 confirmed cases

Top 5 countries with lowest confirmed cases:
Country B: 2000 confirmed cases
Country A: 1000 confirmed cases
Summary report saved to covid19_summary.json

```



```
covid19_summary.json > ...  
1  {  
2    "Country A": {  
3      "total_confirmed": 1000,  
4      "total_deaths": 20,  
5      "total_recovered": 800,  
6      "total_active": 180  
7    },  
8    "Country B": {  
9      "total_confirmed": 2000,  
10     "total_deaths": 40,  
11     "total_recovered": 1500,  
12     "total_active": 460  
13   }  
14 }
```

## **Lab Assignment 8: 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`.

## Data Set –

```
Lab 8 > input.txt
1 Python is an amazing programming language. It is widely used in data science, web development, and automation.
2 Python allows developers to write clear and logical code for small and large-scale projects.
3
```

## Code –

```
import os

# Custom exceptions
class FileNotFoundErrorCustom(Exception):
    pass

class InvalidInputDataError(Exception):
    pass

class DiskSpaceFullError(Exception):
    pass

def read_input_file(file_path):
    """Reads the input file specified by the user."""
    if not os.path.exists(file_path):
        raise FileNotFoundErrorCustom(f"File '{file_path}' not found. Please provide a valid path.")

    try:
        with open(file_path, 'r') as file:
            data = file.read()
    except Exception as e:
        raise InvalidInputDataError(f"An error occurred while reading the file: {e}")

    if not isinstance(data, str) or len(data.strip()) == 0:
        raise InvalidInputDataError("Input data is not valid. Expected non-empty text data.")

    return data

def process_text(data):
    """Processes the text by counting words and calculating character frequencies."""
    if not isinstance(data, str):
        raise InvalidInputDataError("Invalid input data type. Expected a string.")

    # Word count
    words = data.split()
    word_count = len(words)
```

```

# Character frequency
char_freq = {}
for char in data:
    if char.isalpha():
        char_freq[char] = char_freq.get(char, 0) + 1

return word_count, char_freq

def save_results(output_file, word_count, char_freq):
    """Saves the processed results to an output file."""
    try:
        with open(output_file, 'w') as file:
            file.write(f"Word Count: {word_count}\n")
            file.write(f"Character Frequency:\n")
            for char, freq in char_freq.items():
                file.write(f"{char}: {freq}\n")
    except OSError as e:
        if 'No space left on device' in str(e):
            raise DiskSpaceFullError("Failed to write output file due to
insufficient disk space.")
        else:
            raise

def main():
    try:
        input_file = "./Lab 8/input.txt"
        data = read_input_file(input_file)

        # Process the text
        word_count, char_freq = process_text(data)

        # Save results to output file
        output_file = 'output_results.txt'
        save_results(output_file, word_count, char_freq)

        print(f"Processing complete. Results saved to {output_file}.")

    except FileNotFoundErrorCustom as e:
        print(f"Error: {e}")

    except InvalidInputDataError as e:
        print(f"Error: {e}")

    except DiskSpaceFullError as e:
        print(f"Error: {e}")

    except Exception as e:

```

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

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

## Output –

```
PS D:\5th Lab\Python> & "C:/Users/Katha Patel/AppData/Local/Programs/Python/Python312/python.exe" "d:/5th Lab/Python/Lab 8/exception.py"
Processing complete. Results saved to output_results.txt.
```

```
output_results.txt
1  Word Count: 31
2  Character Frequency:
3  P: 2
4  y: 3
5  t: 10
6  h: 2
7  o: 13
8  n: 13
9  i: 10
10 s: 9
11 a: 19
12 m: 6
13 z: 1
14 g: 7
15 p: 4
16 r: 8
17 l: 13
18 u: 3
19 e: 18
20 I: 1
21 w: 4
22 d: 9
23 c: 7
24 b: 1
25 v: 2
26 f: 1
27 j: 1
```

## **Lab Assignment 9: Logging engineer**

Analyse 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.

Code –

```

import logging
import time

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

def get_user_input():
    """Handles user input and logs relevant information."""
    try:
        user_input = input("Enter a number: ")
        logging.info("User input received: %s", user_input)
        if not user_input.isdigit():
            raise ValueError("Invalid input: Not a number")
        return int(user_input)
    except ValueError as e:
        logging.error("Error in user input: %s", e)
        return None

def process_data(data):
    """Processes the input data and logs progress."""
    logging.info("Starting data processing")
    try:
        # Simple transformation: multiply each number by 2
        transformed_data = [x * 2 for x in data]
        logging.debug("Data after transformation: %s", transformed_data)

        if not transformed_data:
            raise Exception("No data to process")

        logging.info("Data processing completed successfully")
        return transformed_data
    except Exception as e:
        logging.error("Error during data processing: %s", e)
        return None

def main():
    # User input
    data = []
    while len(data) < 5:
        user_number = get_user_input()
        if user_number is not None:
            data.append(user_number)

    # Process data
    processed_data = process_data(data)
    if processed_data:

```



```
        logging.info("Final processed data: %s", processed_data)

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

## Output –

```
Enter a number: 34
2024-10-17 16:31:10,434 - INFO - User input received: 34
Enter a number: 23
2024-10-17 16:31:10,434 - INFO - User input received: 34
Enter a number: 23
Enter a number: 23
2024-10-17 16:31:14,393 - INFO - User input received: 23
2024-10-17 16:31:14,393 - INFO - User input received: 23
Enter a number: 56
2024-10-17 16:31:15,176 - INFO - User input received: 56
2024-10-17 16:31:15,176 - INFO - User input received: 56
Enter a number: 23
2024-10-17 16:31:15,974 - INFO - User input received: 23
Enter a number: 798
2024-10-17 16:31:17,044 - INFO - User input received: 798
2024-10-17 16:31:17,044 - INFO - Starting data processing
2024-10-17 16:31:17,045 - DEBUG - Data after transformation: [68, 46, 112, 46, 1596]
2024-10-17 16:31:17,046 - INFO - Data processing completed successfully
2024-10-17 16:31:17,046 - INFO - Final processed data: [68, 46, 112, 46, 1596]
```

## **Lab Assignment 10: Pdf reading creation and editing**

Analyse 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
7. Total Amount

## Code –

```

import pandas as pd
from fpdf import FPDF
from PyPDF2 import PdfMerger
from datetime import datetime
import os

# Step 1: Load order data from the CSV file
def load_order_data(csv_file):
    try:
        # Reading the CSV file using pandas
        return pd.read_csv(csv_file)
    except FileNotFoundError:
        print(f"Error: The file '{csv_file}' was not found.")
        return None

# Step 2: Create PDF invoice for each order
def generate_invoice(order, output_dir):
    # Extract order details
    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 = quantity * unit_price
    date_of_purchase = datetime.now().strftime("%Y-%m-%d")

    # Initialize FPDF and create PDF document
    pdf = FPDF()
    pdf.add_page()

    # Set title and add text to the PDF
    pdf.set_font("Arial", size=12)

    # Invoice Header
    pdf.cell(200, 10, txt=f"Invoice: {order_id}", ln=True, align="C")
    pdf.cell(200, 10, txt=f>Date: {date_of_purchase}", ln=True, align="C")
    pdf.ln(10)

    # Customer Details
    pdf.cell(200, 10, txt=f"Customer Name: {customer_name}", ln=True,
align="L")
    pdf.cell(200, 10, txt=f"Product: {product_name}", ln=True, align="L")

    # Order Details
    pdf.cell(200, 10, txt=f"Quantity: {quantity}", ln=True, align="L")
    pdf.cell(200, 10, txt=f"Unit Price: ${unit_price}", ln=True, align="L")

```

```

    pdf.cell(200, 10, txt=f"Total Amount: ${total_amount:.2f}", ln=True,
align="L")

    # Save the PDF file
    output_file = os.path.join(output_dir, f"invoice_{order_id}.pdf")
    pdf.output(output_file)
    return output_file

# Step 3: Merge all invoices into a single PDF
def merge_invoices(pdf_files, output_file):
    merger = PdfMerger()
    for pdf in pdf_files:
        merger.append(pdf)

    # Write the merged PDF to a file
    merger.write(output_file)
    merger.close()

def main():
    # Path to the CSV file containing order data
    csv_file = "D:/5th Lab/Python/Lab 10/orders.csv"

    # Directory where invoices will be generated
    output_dir = "invoices"
    os.makedirs(output_dir, exist_ok=True)

    # Load the order data from the CSV
    orders = load_order_data(csv_file)

    if orders is not None:
        pdf_files = []

        # Iterate through each order and generate invoices
        for index, order in orders.iterrows():
            pdf_file = generate_invoice(order, output_dir)
            pdf_files.append(pdf_file)

        # Merge the generated invoices into a single PDF file
        merged_output_file = "merged_invoices.pdf"
        merge_invoices(pdf_files, merged_output_file)

        print(f"Merged invoices saved to '{merged_output_file}'.")

if __name__ == "__main__":
    main()

```

Output –

```
Merged invoices saved to 'merged_invoices.pdf'.
```

Invoice: A001

Date: 2024-10-17

Customer Name: John Doe

Product: Laptop

Quantity: 1

Unit Price: \$1200.0

Total Amount: \$1200.00

Invoice: A002

Date: 2024-10-17

Customer Name: Jane Smith

Product: Smartphone

Quantity: 2

Unit Price: \$800.0

Total Amount: \$1600.00

Invoice: A003

Date: 2024-10-17

Customer Name: Alice Johnson

Product: Headphones

Quantity: 1

Unit Price: \$150.0

Total Amount: \$150.00

Invoice: A004

Date: 2024-10-17

Customer Name: Bob Brown

Product: Monitor

Quantity: 3

Unit Price: \$300.0

Total Amount: \$900.00

Invoice: A005

Date: 2024-10-17

Customer Name: Charlie Davis

Product: Keyboard

Quantity: 4

Unit Price: \$100.0

Total Amount: \$400.00

