**Python programming Lab(23CP301P)**

**Name: Krishika Vansh Semester: V**

**Roll No: 23BCP448 Faculty: Mr. Davinder Singh**

**Division: VI I Batch G13**

**Branch: Computer Engineering**

**A logo of a university

AI-generated content may be incorrect.**

**School of Technology**

**November 2025**

**LAB1:**

**Experiment No: 1**

**Text File Analysis for Product Reviews**

**Objective:** To develop a Python script that reads multiple customer review files from a directory, extracts structured data using regular expressions, computes the average rating per product, identifies the top-rated products, and writes a detailed summary to a text file.

Code:

import os

import re

from collections import defaultdict

directory = "reviews"

product\_ratings = defaultdict(list)

total\_reviews = 0

valid\_reviews = 0

invalid\_reviews = 0

customer\_pattern = re.compile(**r**"[A-Za-z0-9]{6}")

product\_pattern = re.compile(**r**"[A-Za-z0-9]{10}")

date\_pattern = re.compile(**r**"\d{4}-\d{2}-\d{2}")

rating\_pattern = re.compile(**r**"\b[1-5]\b")

review\_text\_pattern = re.compile(**r**"Review:\s\*(.\*)")

for filename in os.listdir(directory):

    if filename.endswith(".txt"):

        filepath = os.path.join(directory, filename)

        try:

            with open(filepath, "r", encoding="utf-8") as file:

                content = file.read()

                reviews = content.strip().split("\n\n")

                for review in reviews:

                    total\_reviews += 1

                    customer\_id = customer\_pattern.search(review)

                    product\_id = product\_pattern.search(review)

                    review\_date = date\_pattern.search(review)

                    rating = rating\_pattern.search(review)

                    review\_text = review\_text\_pattern.search(review)

                    if all([customer\_id, product\_id, review\_date, rating, review\_text]):

                        valid\_reviews += 1

                        product\_ratings[product\_id.group()].append(int(rating.group()))

                    else:

                        invalid\_reviews += 1

        except Exception as e:

            print(**f**"Error reading {filename}: {e}")

avg\_ratings = {

    pid: round(sum(ratings) / len(ratings), 2)

    for pid, ratings in product\_ratings.items()

}

top\_products = sorted(avg\_ratings.items(), key=**lambda** x: x[1], reverse=True)[:3]

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

    summary.write(**f**"Total reviews processed: {total\_reviews}\n")

    summary.write(**f**"Valid reviews: {valid\_reviews}\n")

    summary.write(**f**"Invalid reviews: {invalid\_reviews}\n")

    summary.write("\nTop 3 Products:\n")

    for pid, avg in top\_products:

        summary.write(**f**"{pid}: {avg}\n")

print("Summary saved to 'summary.txt'")

Output:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Summary.txt:

Total reviews processed: 6

Valid reviews: 6

Invalid reviews: 0

Top 3 Products:CustomerID: 3.83

**LAB2:**

**Experiment No: 2**

**CSV File Handling and Basic Operations**

**Objective:** To write a Python program that reads student grades from a CSV file, calculates the average score for each student, and writes the result into a new CSV file. The program will demonstrate effective CSV file manipulation and the use of functions for modular and reusable code.

Code:

import csv

with open("student\_grades.csv", "r") as file:

    reader = csv.reader(file)

    next(reader)  *# Skip header*

    data = list(reader)

averages = {}

for row in data:

    name = row[0]

    scores = list(map(float, row[1:]))

    avg = round(sum(scores) / len(scores), 2)

    averages[name] = avg

with open("student\_average\_grades.csv", "w", newline="") as file:

    writer = csv.writer(file)

    writer.writerow(["Name", "Average"])

    for name, avg in averages.items():

        writer.writerow([name, avg])

print("Done! 'student\_average\_grades.csv' created.")

Output:

A screenshot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

Name,Average

Alice,84.33

Bob,75.0

Charlie,90.67

David,65.0

Eva,88.33

**Lab 3:**

**Experiment No: 3**

**Railway Ticket Reservation System using Python**

**Objective:** To implement a Python program that simulates a basic railway ticket reservation system by loading train and passenger data from CSV files, checking seat availability, booking tickets, updating train data, calculating fare, and generating reports.

**Code:**

import csv

import sys

**def** load\_trains(filename):

    trains = {}

    try:

        with open(filename, mode="r", newline="") as file:

            reader = csv.DictReader(file)

            for row in reader:

                trains[row['Train ID']] = {

                    "TrainName": row['Train Name'],

                    "Source": row['Source Station'],

                    "Destination": row['Destination Station'],

                    "Seats": int(row['Total Seats']),

                    "FarePerSeat": int(row['fareperseat'])

                }

    except FileNotFoundError:

        print(**f**"Error: {filename} not found.")

        sys.exit(1)

    except KeyError as e:

        print(**f**"CSV format error: Missing column {e}")

        sys.exit(1)

    return trains

**def** load\_passengers(filename):

    passengers = []

    try:

        with open(filename, mode="r", newline="") as file:

            reader = csv.DictReader(file)

            for row in reader:

                passengers.append({

                    "Name": row['Passenger Name'],

                    "TrainID": row['Train ID'],

                    "Tickets": int(row['Number of Tickets'])

                })

    except FileNotFoundError:

        print(**f**"Error: {filename} not found.")

        sys.exit(1)

    except KeyError as e:

        print(**f**"CSV format error: Missing column {e}")

        sys.exit(1)

    return passengers

**def** check\_availability(train, requested\_tickets):

    return train["Seats"] >= requested\_tickets

**def** calculate\_fare(tickets, fare\_per\_seat):

    return tickets \* fare\_per\_seat

**def** book\_ticket(trains, passenger, revenue):

    train\_id = passenger["TrainID"]

    if train\_id not in trains:

        print(**f**"Booking Error for {passenger['Name']}: Invalid Train ID {train\_id}")

        return

    train = trains[train\_id]

    requested\_tickets = passenger["Tickets"]

    if check\_availability(train, requested\_tickets):

        fare = calculate\_fare(requested\_tickets, train["FarePerSeat"])

        train["Seats"] -= requested\_tickets

        revenue[train\_id] = revenue.get(train\_id, 0) + fare

        print(**f**"Booking confirmed for {passenger['Name']} on Train {train\_id} | Fare: Rs.{fare}")

    else:

        print(**f**"Booking Error for {passenger['Name']}: Not enough seats available.")

**def** generate\_report1(trains):

    print("\n--- Report 1: Train Details & Seat Availability ---")

    print(**f**"{'Train ID'**:<10**} {'Train Name'**:<20**} {'Source'**:<15**} {'Destination'**:<15**} {'Seats Available'}")

    for tid, details in trains.items():

        print(**f**"{tid**:<10**} {details['TrainName']**:<20**} {details['Source']**:<15**} {details['Destination']**:<15**} {details['Seats']}")

**def** generate\_report2(revenue, trains):

    print("\n--- Report 2: Train Revenue ---")

    print(**f**"{'Train ID'**:<10**} {'Train Name'**:<20**} {'Revenue (Rs.)'}")

    for tid, rev in revenue.items():

        print(**f**"{tid**:<10**} {trains[tid]['TrainName']**:<20**} {rev}")

**def** main():

    trains\_file = "Train.csv"

    passengers\_file = "Passenger.csv"

    trains = load\_trains(trains\_file)

    passengers = load\_passengers(passengers\_file)

    revenue = {}

    for passenger in passengers:

        book\_ticket(trains, passenger, revenue)

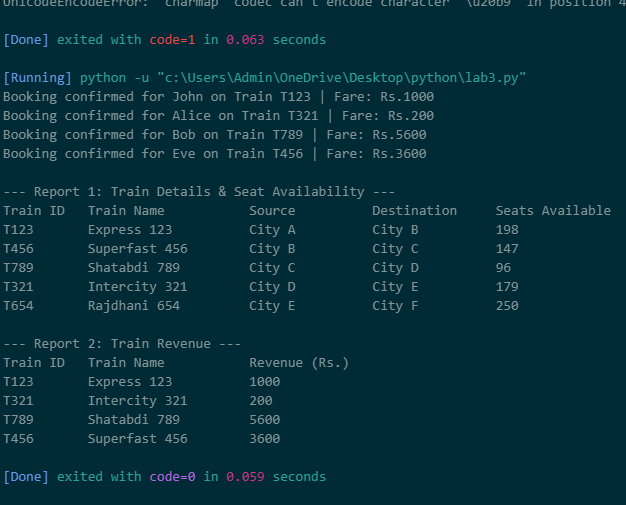
    generate\_report1(trains)

    generate\_report2(revenue, trains)

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

    main()

**Output:**



**LAB4:**

**Experiment No: 4**

**Advanced Operations on csv file using Pandas**

**Code:**

import os

import pandas as pd

*# === Paths ===*

base\_dir = "."

product\_names\_file = os.path.join(base\_dir, "product\_names.csv")

sales\_dir = os.path.join(base\_dir, "sales")

summary\_file = os.path.join(base\_dir, "sales\_summary.csv")

*# === Step 1: Load product names ===*

product\_df = pd.read\_csv(product\_names\_file)

product\_dict = dict(zip(product\_df["ProductID"], product\_df["Product\_Name"]))

*# === Step 2: Gather sales CSV files ===*

sales\_files = [os.path.join(sales\_dir, f) for f in os.listdir(sales\_dir) if f.endswith(".csv")]

*# === Step 3: Process sales data ===*

product\_sales = {}

months\_set = set()

for file in sales\_files:

    df = pd.read\_csv(file)

    df['Month'] = pd.to\_datetime(df['Date']).dt.to\_period('M')

    for \_, row in df.iterrows():

        pid = row["ProductID"]

        qty = int(row["Quantity"])

        product\_sales[pid] = product\_sales.get(pid, 0) + qty

    months\_set.update(df['Month'].unique())

*# === Step 4: Calculate statistics ===*

months\_count = len(months\_set)

summary\_data = []

for pid, total\_qty in product\_sales.items():

    avg\_qty = total\_qty / months\_count if months\_count else 0

    pname = product\_dict.get(pid, "Unknown Product")

    summary\_data.append([pid, pname, total\_qty, round(avg\_qty, 2)])

*# === Step 5: Sort & Save Top 5 to CSV ===*

summary\_df = pd.DataFrame(summary\_data, columns=[

    "Product ID", "Product Name", "Total Quantity Sold", "Average Quantity Sold per Month"

])

summary\_df = summary\_df.sort\_values(by="Total Quantity Sold", ascending=False).head(5)

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

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**

**Experiment No: 5**

**JSON Data Parsing and Manipulation**

**Objective:** To develop a Python program that reads and analyzes structured JSON files containing COVID19 data across multiple countries, computes summary statistics, identifies extreme cases, and generates a consolidated report.

Code:

import os

import json

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)

                try:

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

                        data = json.load(f)

                        covid\_data.append(data)

                except Exception as e:

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

    return covid\_data

def process\_covid\_data(covid\_data):

    summary = {}

    for record in covid\_data:

        country = record["country"]

        confirmed\_total = record["confirmed\_cases"]["total"]

        deaths\_total = record["deaths"]["total"]

        recovered\_total = record["recovered"]["total"]

        if country not in summary:

            summary[country] = {

                "total\_confirmed": 0,

                "total\_deaths": 0,

                "total\_recovered": 0

            }

        summary[country]["total\_confirmed"] += confirmed\_total

        summary[country]["total\_deaths"] += deaths\_total

        summary[country]["total\_recovered"] += recovered\_total

    for country, stats in summary.items():

        stats["total\_active"] = stats["total\_confirmed"] - (

            stats["total\_deaths"] + stats["total\_recovered"]

        )

    return summary

def find\_extremes(summary):

    sorted\_countries = sorted(summary.items(), key=lambda x: x[1]["total\_confirmed"], reverse=True)

    top\_5\_highest = sorted\_countries[:5]

    top\_5\_lowest = sorted\_countries[-5:]

    return top\_5\_highest, top\_5\_lowest

def save\_summary(summary, output\_file="covid19\_summary.json"):

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

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

    print(f"Summary report saved to {output\_file}")

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

    directory = "covid\_data"

    covid\_data = read\_json\_files(directory)

    summary = process\_covid\_data(covid\_data)

    print("\n--- COVID-19 Statistics by Country ---")

    for country, stats in summary.items():

        print(f"{country}: {stats}")

    top\_highest, top\_lowest = find\_extremes(summary)

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

    for country, stats in top\_highest:

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

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

    for country, stats in top\_lowest:

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

    save\_summary(summary)

output:

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Experiment No: 6**

**Error and Exception Handling in Python Programs**

**Objective:** To build a robust Python-based text processing tool that can read input from a user-specified file, process the content, and write results to an output file using custom exception handling mechanisms to ensure fault tolerance and graceful error reporting.

Code:

import os

import shutil

**class** InvalidInputDataError(Exception):

**def** \_\_init\_\_(self, message="Invalid input data."):

        super().\_\_init\_\_(message)

**class** DiskSpaceFullError(Exception):

**def** \_\_init\_\_(self, message="Not enough disk space to save the file."):

        super().\_\_init\_\_(message)

**class** CustomFileNotFoundError(Exception):

**def** \_\_init\_\_(self, message="Input file not found."):

        super().\_\_init\_\_(message)

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

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

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

    with open(file\_path, "r") as f:

        content = f.read()

    if not content.strip():

        raise InvalidInputDataError("Error: Input file is empty or contains invalid data.")

    return content

**def** process\_word\_counts(content):

    words = content.split()

    if not words:

        raise InvalidInputDataError("Error: No valid words found in the file.")

    word\_count = {}

    for word in words:

        word = word.lower().strip(",.!?;:\"'()[]{}")

        if word:

            word\_count[word] = word\_count.get(word, 0) + 1

    return word\_count

**def** process\_char\_counts(content):

    char\_count = {}

    for char in content:

        if char.isalnum():

            char = char.lower()

            char\_count[char] = char\_count.get(char, 0) + 1

    return char\_count

**def** save\_output(output\_file, word\_count, char\_count):

    try:

        total, used, free = shutil.disk\_usage(".")

        if free < 1024:  *# less than 1 KB free*

            raise DiskSpaceFullError()

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

            f.write("=== Word Frequency ===\n")

            for word, count in word\_count.items():

                f.write(**f**"{word}: {count}\n")

            f.write("\n=== Character Frequency ===\n")

            for char, count in char\_count.items():

                f.write(**f**"{char}: {count}\n")

    except OSError as e:

        raise DiskSpaceFullError(**f**"Disk write error: {e}")

**def** main():

    input\_file = "input6.txt"

    output\_file = "output6.txt"

    try:

        content = read\_input\_file(input\_file)

        word\_count = process\_word\_counts(content)

        char\_count = process\_char\_counts(content)

        save\_output(output\_file, word\_count, char\_count)

        print(**f**"Analysis saved to '{output\_file}' successfully.")

    except CustomFileNotFoundError as e:

        print(e)

    except InvalidInputDataError as e:

        print(e)

    except DiskSpaceFullError as e:

        print(e)

    except Exception as e:

        print(**f**"Unexpected Error: {e}")

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

    main()

A screen shot of a computer

AI-generated content may be incorrect.

Output:

A screenshot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

**Experiment No: 7**

**Implementing Logging Mechanism in Python**

**Objective:** To analyze a Python-based project, identify potential points for logging, and implement dummy code with logging at key places such as function calls, exception handling, input/output operations, and loops for debugging and monitoring purposes.

Code:

import logging

import json

import random

import time

import os

logging.basicConfig(

    level=logging.DEBUG,

    format='%(asctime)s | %(levelname)-7s | %(message)s'

)

logger = logging.getLogger("simple")

**def** read\_input(path):

    logger.info("Reading input file: %s", path)

    if not os.path.exists(path):

        logger.info("Input not found — writing a small sample to %s", path)

        sample = {"name": "sample-run", "values": [random.random() for \_ in range(200)], "factor": 1.1}

        with open(path, "w") as f:

            json.dump(sample, f)

    try:

        with open(path) as f:

            data = json.load(f)

        logger.debug("Loaded JSON keys: %s", list(data.keys()))

        return data

    except Exception:

        logger.exception("Failed to read/parse input")

        return None

**def** validate\_input(data):

    logger.info("Validating input")

    if not data:

        logger.warning("No data provided")

        return False

    if "values" not in data or not isinstance(data["values"], list):

        logger.error("Missing or invalid 'values' field")

        return False

    logger.debug("Validation OK (name=%s, n\_values=%d)", data.get("name"), len(data["values"]))

    return True

**def** process\_values(values):

    logger.info("Processing %d values", len(values))

    result = []

    for i, v in enumerate(values):

        if i % 50 == 0 and i > 0:

            logger.debug("Processed %d items", i)

        try:

            num = float(v)

            if num < 0.2:

                logger.debug("Small value at index %d: %f", i, num)

            result.append(num \* 2)

        except Exception:

            logger.warning("Skipping invalid item at index %d: %r", i, v)

    logger.info("Processing finished, kept %d items", len(result))

    return result

**def** fake\_api\_call(payload):

    logger.info("Calling external service with %d items", len(payload.get("items", [])))

    try:

        time.sleep(random.uniform(0.01, 0.05))

        if random.random() < 0.05:

            raise TimeoutError("simulated timeout")

        resp = {"status": "ok", "received": len(payload.get("items", []))}

        logger.info("External service replied: %s", resp)

        return resp

    except Exception:

        logger.exception("External call failed")

        return {"status": "error"}

**def** main(input\_path="input.json"):

    logger.info("Run started")

    data = read\_input(input\_path)

    if not validate\_input(data):

        logger.error("Validation failed, exiting")

        return

    values = process\_values(data["values"])

    if not values:

        logger.error("No valid values after processing, exiting")

        return

    payload = {"items": values[:100], "meta": {"name": data.get("name")}}

    resp = fake\_api\_call(payload)

    if resp.get("status") == "ok":

        logger.info("Pipeline succeeded for %s", data.get("name"))

    else:

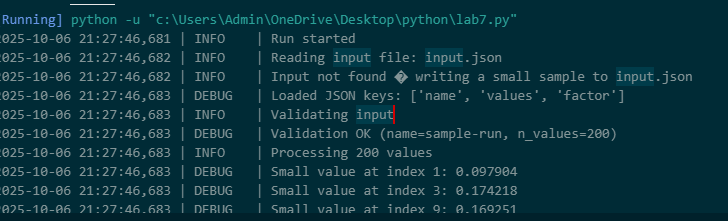
        logger.warning("Pipeline finished with issues (status=%s)", resp.get("status"))

    logger.info("Run finished")

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

    main()

Output:



**Experiment No: 8**

**Image Processing using Python Libraries**

**Objective:** To develop a Python program that performs various image processing operations including loading, displaying, manipulating images, and analyzing image histograms.

Code:

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image, ImageFilter, ImageEnhance

import cv2

def load\_and\_display\_image(file\_path):

    try:

        img = Image.open(file\_path)

        plt.figure(figsize=(8, 6))

        plt.imshow(img)

        plt.title('Original Image')

        plt.axis('off')

        plt.show()

        return img

    except FileNotFoundError:

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

        return None

def manipulate\_image(img):

    if img is None:

        return

    grayscale\_img = img.convert('L')

    blurred\_img = img.filter(ImageFilter.GaussianBlur(radius=5))

    img\_np = np.array(img)

    gray\_np = cv2.cvtColor(img\_np, cv2.COLOR\_RGB2GRAY)

    edges\_np = cv2.Canny(gray\_np, threshold1=100, threshold2=200)

    edge\_img = Image.fromarray(edges\_np)

    enhancer = ImageEnhance.Brightness(img)

    bright\_img = enhancer.enhance(1.8)

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    axs[0, 0].imshow(grayscale\_img, cmap='gray')

    axs[0, 0].set\_title('Grayscale')

    axs[0, 0].axis('off')

    axs[0, 1].imshow(blurred\_img)

    axs[0, 1].set\_title('Gaussian Blur')

    axs[0, 1].axis('off')

    axs[1, 0].imshow(edge\_img, cmap='gray')

    axs[1, 0].set\_title('Edge Detection (Canny)')

    axs[1, 0].axis('off')

    axs[1, 1].imshow(bright\_img)

    axs[1, 1].set\_title('Increased Brightness')

    axs[1, 1].axis('off')

    plt.suptitle('Image Manipulations', fontsize=16)

    plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

    plt.show()

def analyze\_histogram(img):

    if img is None:

        return

    img\_np = np.array(img)

    plt.figure(figsize=(10, 6))

    plt.title('Color Histogram')

    plt.xlabel('Pixel Intensity (0-255)')

    plt.ylabel('Number of Pixels')

    colors = ('r', 'g', 'b')

    for i, color in enumerate(colors):

        # cv2.calcHist([images], [channels], mask, [histSize], [ranges])

        histogram = cv2.calcHist([img\_np], [i], None, [256], [0, 256])

        plt.plot(histogram, color=color)

        plt.xlim([0, 256])

    plt.legend(['Red Channel', 'Green Channel', 'Blue Channel'])

    plt.grid(True, linestyle='--', alpha=0.6)

    plt.show()

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

    image\_file = 'myimage.jpeg'

    original\_image = load\_and\_display\_image(image\_file)

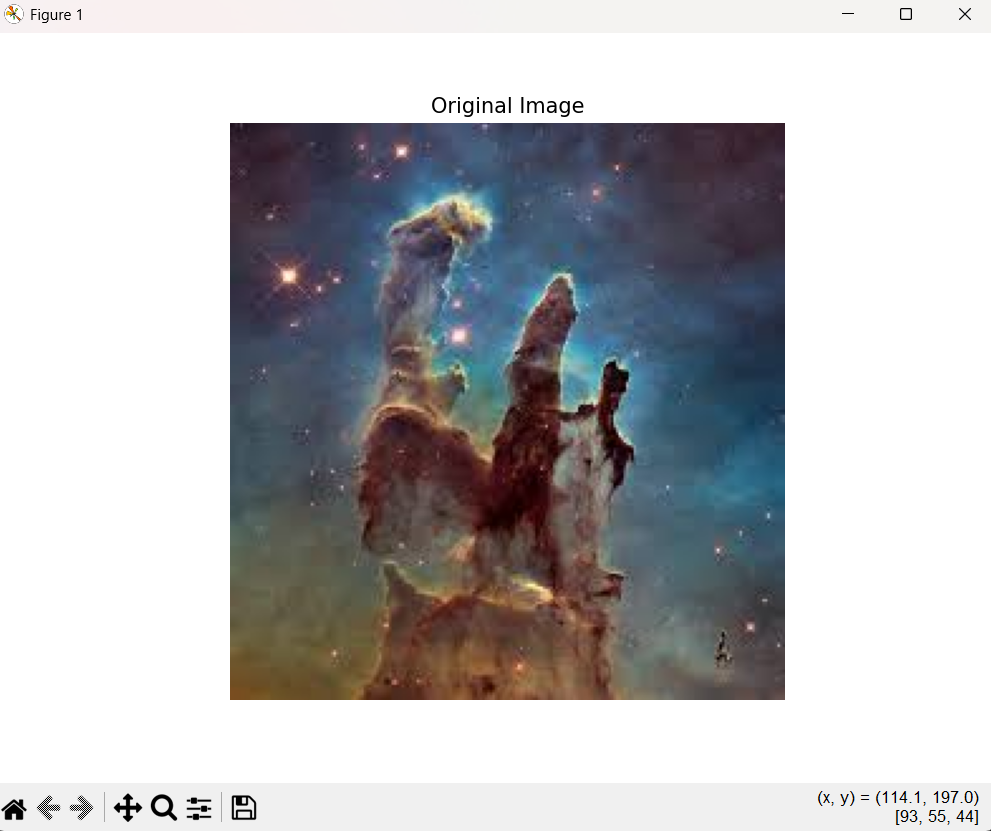
    manipulate\_image(original\_image)

    analyze\_histogram(original\_image)

#install these:

pip install pillow matplotlib numpy opencv-python

Output:



**Experiment No: 9**

**Basic Data Analysis using pandas and matplotlib**

**Objective: To analyze customer transaction data and segment customers based on their shopping behavior using data analysis and clustering techniques in Python.**

Install pip install numpy pandas matplotlib scikit-learn

**Code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# 1. Load dataset

data = pd.read\_csv("customerTransactions.csv")

print("Dataset loaded successfully!")

print(data.head())

# 2. Data Cleaning

print("\nMissing values before cleaning:")

print(data.isnull().sum())

data.dropna(subset=["Customer ID"], inplace=True)

data.drop\_duplicates(inplace=True)

print("\nData cleaned successfully!")

print(f"Total records after cleaning: {len(data)}")

# 3. Descriptive Statistics

print("\nDescriptive Statistics:")

print(data[["Total Amount Spent", "Total Items Purchased"]].describe())

# 4. Clustering Preparation

X = data[["Total Amount Spent", "Total Items Purchased", "Average Purchase Value"]]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 5. Apply K-Means

kmeans = KMeans(n\_clusters=3, random\_state=42)

data["Cluster"] = kmeans.fit\_predict(X\_scaled)

# 6. Visualization

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

plt.scatter(data["Total Amount Spent"], data["Total Items Purchased"],

            c=data["Cluster"], cmap="viridis", s=100, edgecolors='k')

plt.title("Customer Segmentation based on Spending and Purchase Behavior")

plt.xlabel("Total Amount Spent")

plt.ylabel("Total Items Purchased")

plt.colorbar(label="Cluster")

plt.show()

# 7. Segment Summary

print("\nCustomer Segment Summary:")

segment\_summary = data.groupby("Cluster")[["Total Amount Spent", "Total Items Purchased", "Average Purchase Value"]].mean()

print(segment\_summary)

# 8. Assign segment labels

cluster\_labels = {

    0: "Low-Value Customers (Inactive/Occasional Shoppers)",

    1: "Mid-Tier Customers (Moderate Shoppers)",

    2: "High-Value Customers (Frequent & High Spenders)"

}

data["Segment"] = data["Cluster"].map(cluster\_labels)

# 9. Insights & Recommendations

print("\nCustomer Engagement Recommendations:")

print("""

High-Value Customers:

   - Offer loyalty rewards, exclusive previews, or premium deals.

   - Personalized product recommendations.

Mid-Tier Customers:

   - Targeted marketing emails with discounts to boost spending.

   - Encourage subscription or bundle offers.

Low-Value/Inactive Customers:

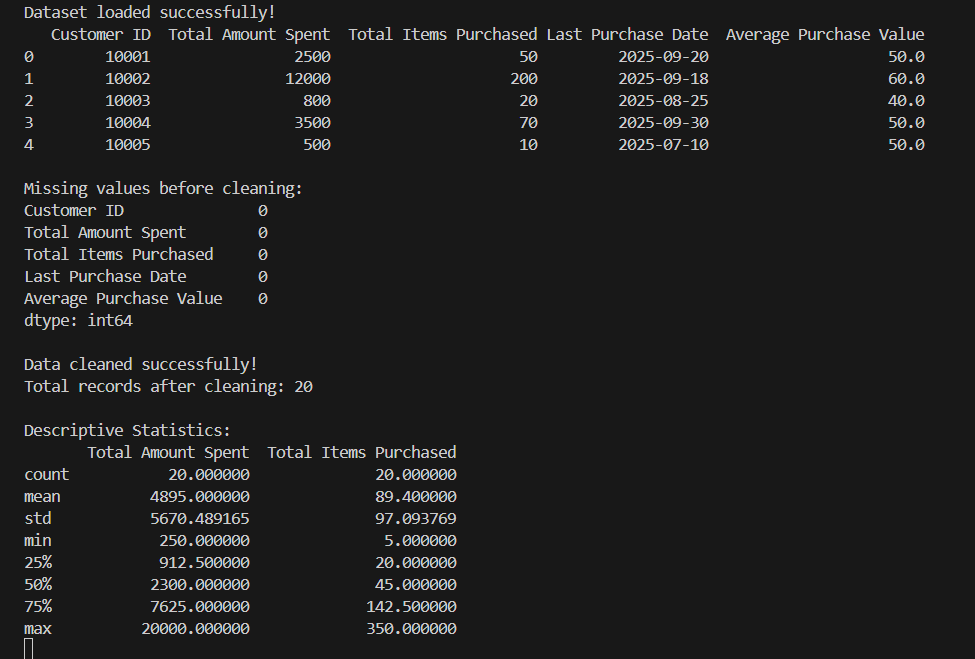
   - Send reactivation offers, reminders, and special limited-time discounts.

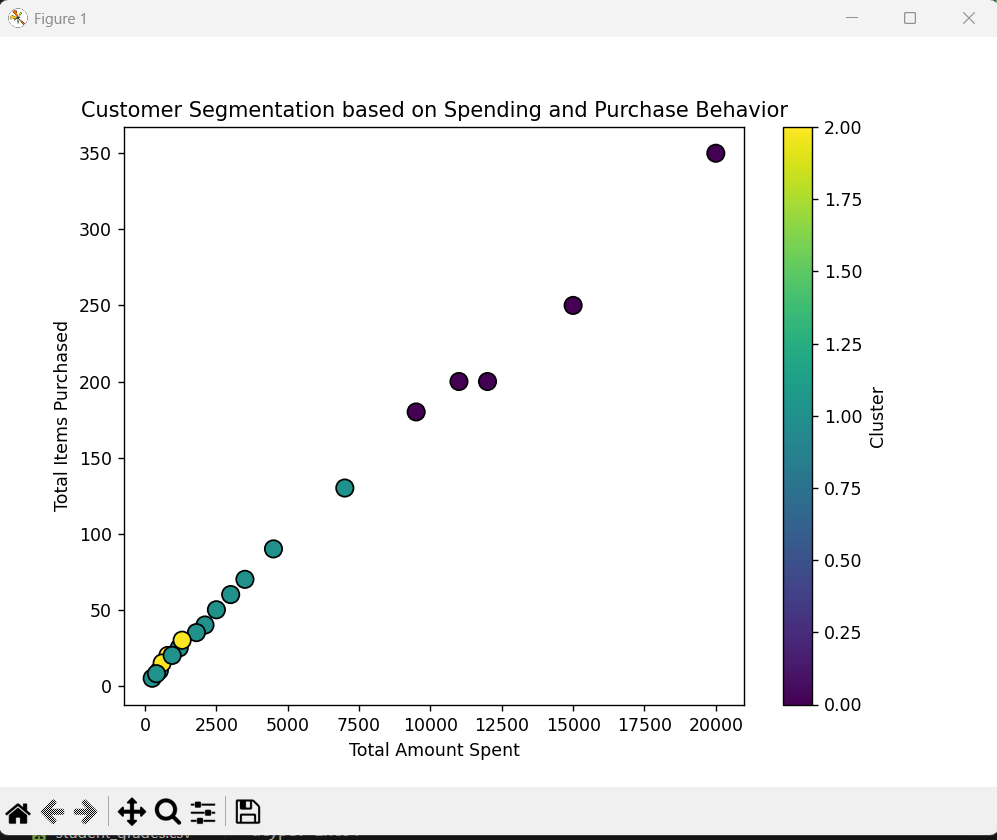
   - Analyze their drop-off reasons (delivery time, pricing, etc.)

""")

print("\nSegmentation complete!")

**Output:**





**Experiment No: 10**

**Reading, Creating, and Modifying PDF Files in Python**

**Objective: To analyze customer transaction data and segment customers based on their shopping behavior using data analysis and clustering techniques in Python.**

pip install pandas numpy matplotlib scikit-learn reportlab PyPDF2

**Code:**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from reportlab.lib.pagesizes import letter

from reportlab.pdfgen import canvas

from PyPDF2 import PdfMerger

from datetime import datetime

import os

print("=== CUSTOMER SEGMENTATION ===")

# Load customer data

customer\_data = pd.read\_csv("customers.csv")

print("Customer Data Loaded Successfully!\n")

print(customer\_data.head())

X = customer\_data[["Total Amount Spent", "Total Items Purchased", "Average Purchase Value"]]

# Apply K-Means clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

customer\_data["Cluster"] = kmeans.fit\_predict(X)

print("\nCustomer Clustering Completed. Cluster Centers:")

print(kmeans.cluster\_centers\_)

# Visualize clusters

plt.figure(figsize=(8, 5))

plt.scatter(customer\_data["Total Amount Spent"], customer\_data["Total Items Purchased"],

            c=customer\_data["Cluster"], cmap='viridis', s=100)

plt.title("Customer Segments Based on Spending and Purchase Frequency")

plt.xlabel("Total Amount Spent")

plt.ylabel("Total Items Purchased")

plt.grid(True)

plt.show()

# Save clustered data

customer\_data.to\_csv("customer\_segments.csv", index=False)

print("\nClustered customer data saved to 'customer\_segments.csv'\n")

print("=== INVOICE GENERATION ===")

# Load order data

orders = pd.read\_csv("orders.csv")

print("Order Data Loaded Successfully!\n")

print(orders.head())

# Create directory for invoices

if not os.path.exists("invoices"):

    os.makedirs("invoices")

# Generate PDF invoices for each order

for \_, order in orders.iterrows():

    order\_id = str(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 = datetime.now().strftime("%Y-%m-%d")

    filename = f"invoices/{order\_id}.pdf"

    c = canvas.Canvas(filename, pagesize=letter)

    c.setFont("Helvetica-Bold", 16)

    c.drawString(200, 750, "INVOICE")

    c.setFont("Helvetica", 12)

    c.drawString(50, 700, f"Invoice Number: {order\_id}")

    c.drawString(50, 680, f"Date of Purchase: {date}")

    c.drawString(50, 660, f"Customer Name: {customer\_name}")

    c.drawString(50, 640, f"Product Name: {product\_name}")

    c.drawString(50, 620, f"Quantity: {quantity}")

    c.drawString(50, 600, f"Unit Price: ₹{unit\_price}")

    c.drawString(50, 580, f"Total Amount: ₹{total\_amount:.2f}")

    c.showPage()

    c.save()

    print(f"Invoice generated for Order ID {order\_id}")

# Merge all invoices into one PDF

merger = PdfMerger()

for file in sorted(os.listdir("invoices")):

    if file.endswith(".pdf"):

        merger.append(os.path.join("invoices", file))

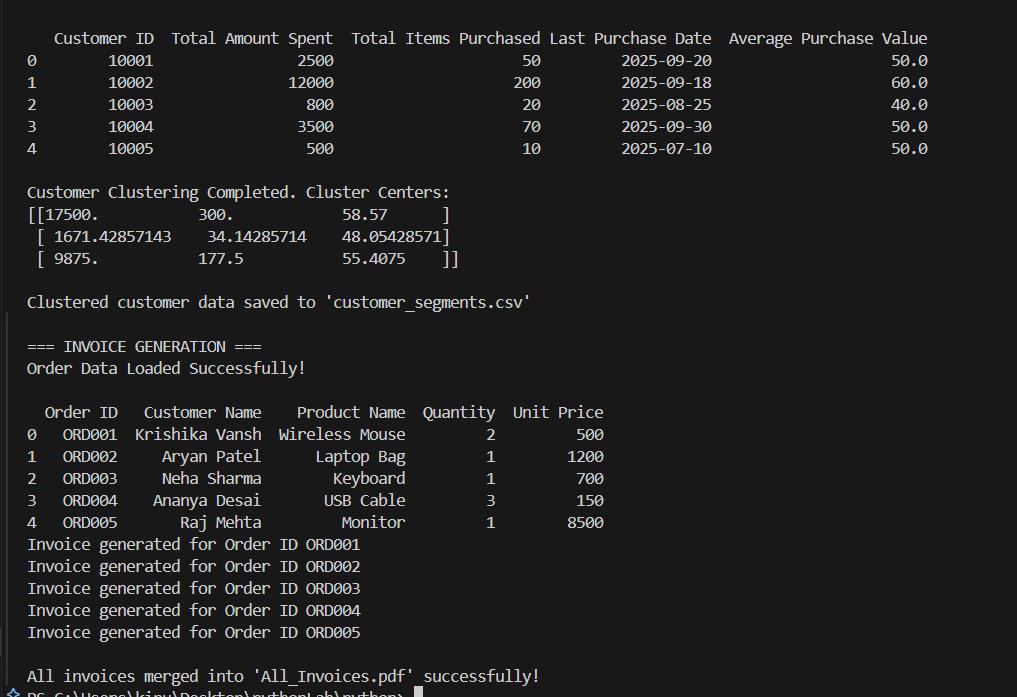
merged\_filename = "All\_Invoices.pdf"

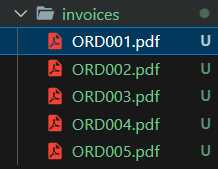
merger.write(merged\_filename)

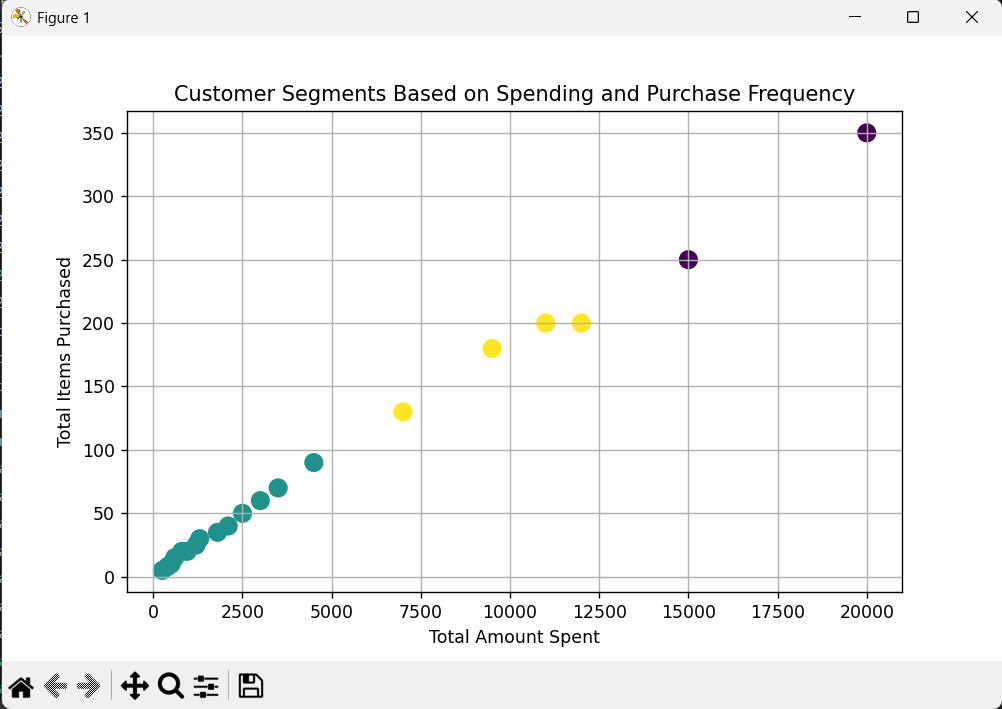
merger.close()

print(f"\nAll invoices merged into '{merged\_filename}' successfully!")

**Output:**

****

****

****