ADVANCE Python IA Assignment

Smit Sutariya

Roll no. 21BCP142

Div:3 G:5

Faculty Name:

COMPUTER ENGINEERING

School of Technology,

Pandit Deendayal

EnergyUniversity

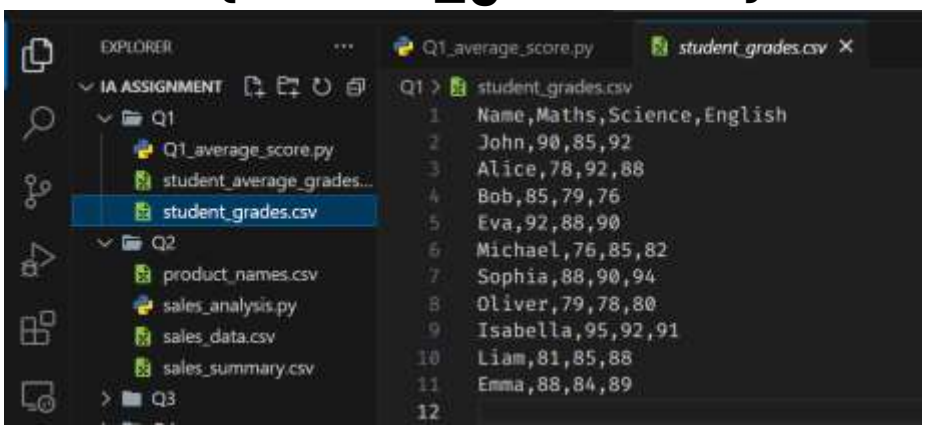
January – 2023

Q1) 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."

CSV FILE {student_grades.csv}:



CODE:

```
import csv

def read_student_grades(filename):
    student_grades = {}
    with open(filename, 'r', newline='') as file:
        reader = csv.DictReader(file)
        for row in reader:
            name = row['Name']
            math = float(row['Maths'])
            science = float(row['Science'])
            english = float(row['English'])
```

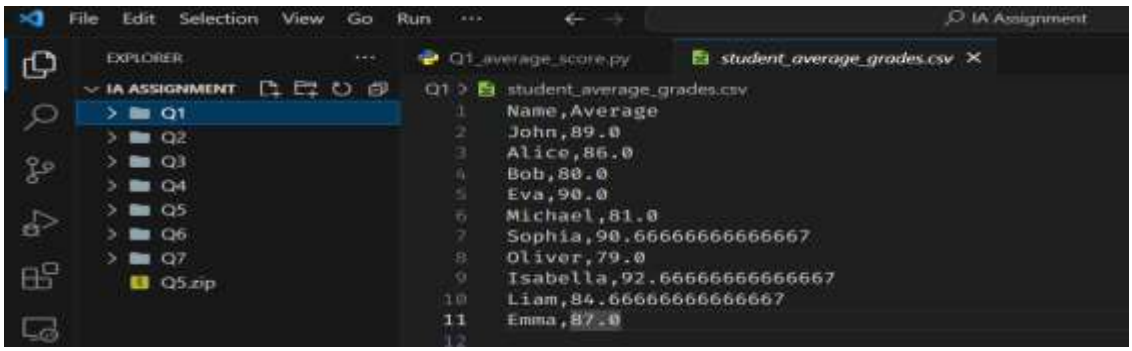
```
            student_grades[name] = [math, science, english]
    return student_grades

def calculate_average(student_grades):
    student_average = {}
    for name, grades in student_grades.items():
        average = sum(grades) / len(grades)
        student_average[name] = average
    return student_average

def write_student_average_to_csv(filename, student_average):
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['Name', 'Average'])
        for name, average in student_average.items():
            writer.writerow([name, average])

if __name__ == "__main__":
    student_grades_data = read_student_grades("D:\\Sem-5\\adv. python\\Vipul's
py\\IA ASSIGNMENT\\All Codes\\Q1\\student_grades.csv")
    student_average_data = calculate_average(student_grades_data)
    write_student_average_to_csv("student_average_grades.csv",
student_average_data)
    print("File written successfully")
```

OUTPUT:

```
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes>
File written successfully
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes>
```

OUTPUT CSV FILE

{student_average_grades.csv}:



Q2) 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.

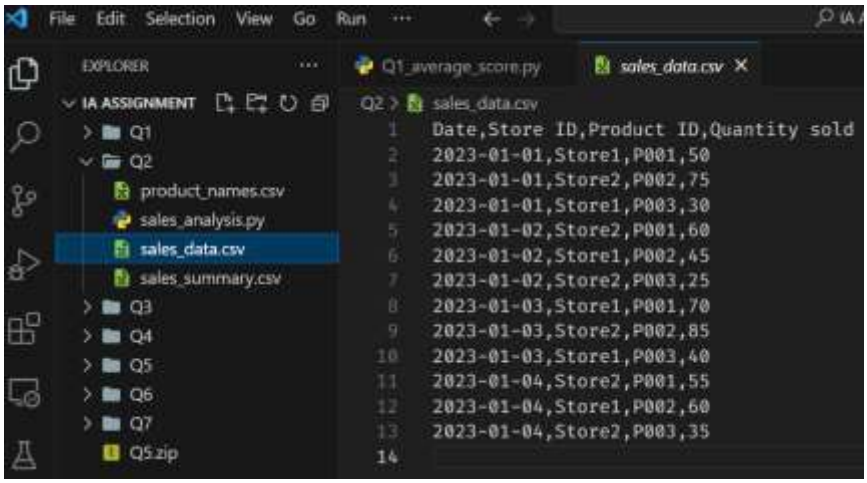
Your task is to write a Python program that performs the following operations:

• 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.

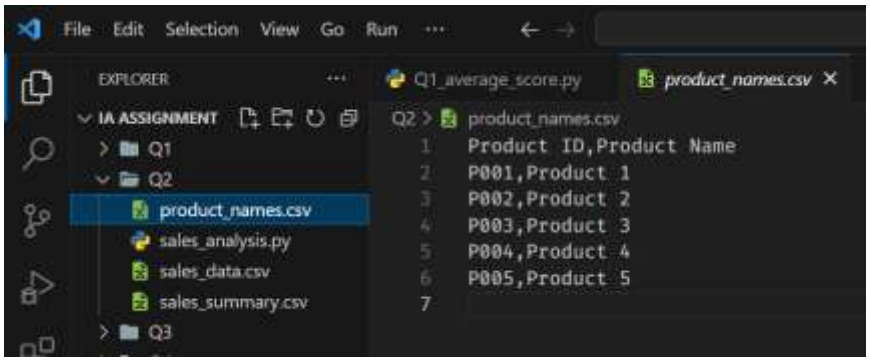
Create a new CSV file named "sales_summary.csv" and write the following information into it:

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

CSV FILE {sales_data.csv}



CSV FILE {product_names.csv}



CODE:

```
import os
import csv
from collections import defaultdict

def read_product_names():
    product_names = {}
    with open(r"D:\\Sem-5\\adv. python\\Vipul's py\\IA ASSIGNMENT\\All
Codes\\Q2\\product_names.csv", mode='r', newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            product_names[row['Product ID']] = row['Product Name']
    return product_names

def process_sales_data(directory):
    total_quantity_sold = defaultdict(int)
```

```
    total_months = defaultdict(int)

    for root, _, files in os.walk(directory):
        for file in files:
            if file.endswith('.csv'):
                with open(os.path.join(root, file), mode='r', newline='') as
csvfile:
                    reader = csv.DictReader(csvfile)
                    for row in reader:
                        product_id = row['Product ID']
                        quantity_sold = int(row['Quantity sold'])
                        total_quantity_sold[product_id] += quantity_sold
                        total_months[product_id] += 1

    return total_quantity_sold, total_months

def get_top_5_products(total_quantity_sold):
    sorted_products = sorted(total_quantity_sold.items(), key=lambda x: x[1],
reverse=True)
    return sorted_products[:5]

def main():
    data_directory = r"D:\\Sem-5\\adv. python\\Vipul's py\\IA ASSIGNMENT\\All
Codes\\Q2\\sales_data.csv"

    product_names = read_product_names()

    total_quantity_sold, total_months = process_sales_data(data_directory)

    top_5_products = get_top_5_products(total_quantity_sold)

    with open('sales_summary.csv', mode='w', newline='') as csvfile:
        fieldnames = ['Product ID', 'Product Name', 'Total Quantity Sold',
'Average Quantity Sold per Month']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for product_id, total_sold in top_5_products:
            product_name = product_names.get(product_id, 'Unknown')
            average_quantity_sold = total_sold / total_months[product_id]
            writer.writerow({
                'Product ID': product_id,
                'Product Name': product_name,
                'Total Quantity Sold': total_sold,
                'Average Quantity Sold per Month': average_quantity_sold
            })
if __name__ == "__main__":
    main()
```

OUTPUT:

```
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT
File written successfully
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT
```

Q3) You are working as a data scientist for a healthcare organization, and your team has been tasked with analyzing COVID-19 data from multiple countries. The data is stored in JSON files, with each file representing the daily COVID-19 statistics for a specific country. Each JSON file has the following structure:

{ "country": "Country Name",
"date": "YYYY-MM-DD",
"confirmed_cases": { "total": 1000, "new": 50 },
"deaths": { "total": 20, "new": 2 },
"recovered": { "total": 800, "new": 30 }
}

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

1. Read COVID-19 data from all JSON files in a given directory and its subdirectories.
2. Calculate and display the following statistics for each country:
   - Total confirmed cases.
   - Total deaths.
   - Total recovered cases.

4. Total active cases (total confirmed cases minus total deaths and total recovered).
3. Determine the top 5 countries with the highest number of confirmed cases and the lowest number of confirmed cases.
5. Generate a summary report in JSON format that includes the statistics for all countries and save it to a file named "covid19_summary.json".

jSON FILES:

Q3 > {} brazil.json > {} recovered > # total

```
{
    "country": "Brazil",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 23000000,
      "new": 18000
    },
    "deaths": {
      "total": 600000,
      "new": 700
    },
    "recovered": {
      "total": 22000000,
      "new": 12000
    }
}
```

Q3 > {} france.json > ...

```
{
    "country": "France",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 7000000,
      "new": 10000
    },
    "deaths": {
      "total": 120000,
      "new": 200
    },
    "recovered": {
      "total": 6900000,
      "new": 8000
    }
}
```

Q3 > {} india.json > country

```
{
    "country": "india",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 1000,
      "new": 50
    },
    "deaths": {
      "total": 20,
      "new": 2
    },
    "recovered": {
      "total": 800,
      "new": 30
    }
}
```

Q3 > {} pakistan.json > country

```
{
    "country": "pakistan",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 2000,
      "new": 75
    },
    "deaths": {
      "total": 40,
      "new": 4
    },
    "recovered": {
      "total": 1500,
      "new": 60
    }
}
```

Q3 > {} russia.json > ...

```
{
    "country": "Russia",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 8000000,
      "new": 12000
    },
    "deaths": {
      "total": 150000,
      "new": 300
    },
    "recovered": {
      "total": 7800000,
      "new": 15000
    }
}
```

Q3 > {} us.json > ...

```
{
    "country": "United States",
    "date": "2023-09-01",
    "confirmed_cases": {
      "total": 40000000,
      "new": 25000
    },
    "deaths": {
      "total": 700000,
      "new": 1000
    },
    "recovered": {
      "total": 35000000,
      "new": 15000
    }
}
```

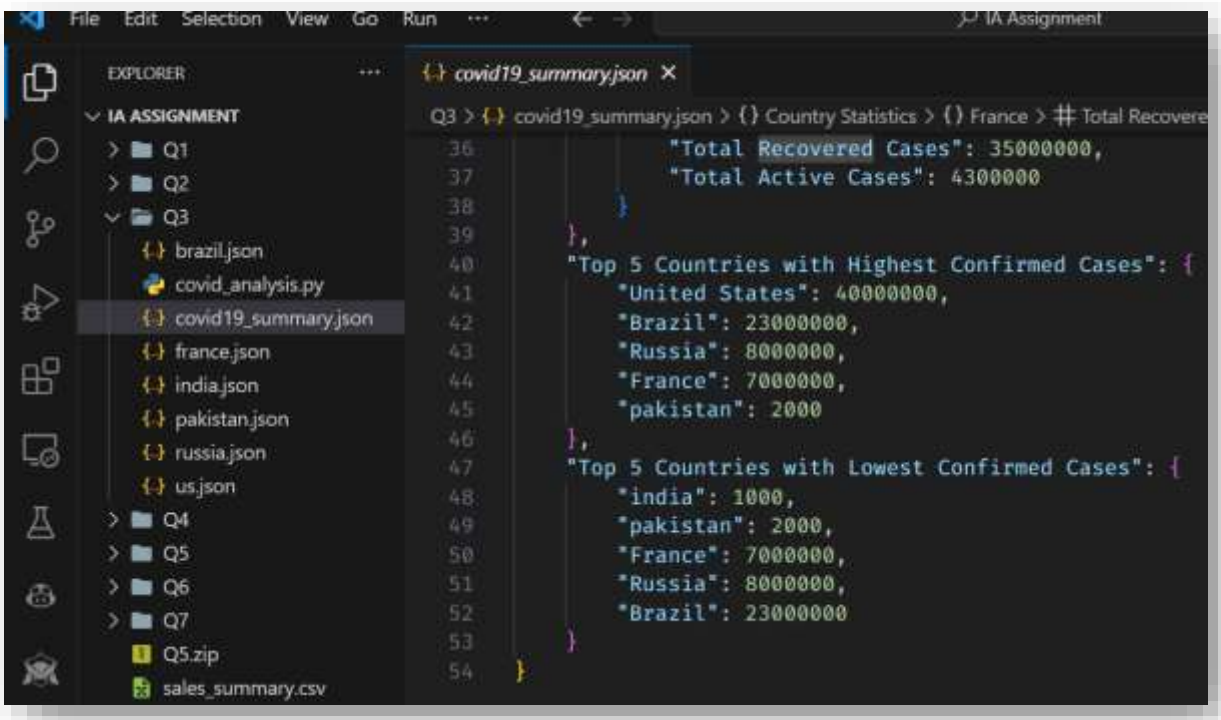
CODE:

```
import os
import json

# Function to read COVID-19 data from JSON files
def read_covid19_data(directory):
    covid19_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 json_file:
                    data = json.load(json_file)
                    covid19_data.append(data)

    return covid19_data

# Function to calculate and display statistics for each country
def calculate_statistics(data):
```

```
    country_statistics = {}

    for record in data:
        country_name = record['country']
        confirmed_cases = record['confirmed_cases']['total']
        deaths = record['deaths']['total']
        recovered = record['recovered']['total']
        active_cases = confirmed_cases - deaths - recovered

        country_statistics[country_name] = {
            'Total Confirmed Cases': confirmed_cases,
            'Total Deaths': deaths,
            'Total Recovered Cases': recovered,
            'Total Active Cases': active_cases
        }

    return country_statistics

# Function to determine the top 5 countries with the highest and lowest
confirmed cases
def find_top_countries(country_statistics):
    sorted_countries = sorted(country_statistics.items(), key=lambda x:
x[1]['Total Confirmed Cases'], reverse=True)
    top_5_highest = sorted_countries[:5]
    top_5_lowest = sorted_countries[-5:][::-1]
    return top_5_highest, top_5_lowest

# Function to generate and save the summary report
def generate_summary_report(country_statistics, top_5_highest, top_5_lowest):
    summary_report = {
        'Country Statistics': country_statistics,
        'Top 5 Countries with Highest Confirmed Cases': {country: stats['Total
Confirmed Cases'] for country, stats in top_5_highest},
        'Top 5 Countries with Lowest Confirmed Cases': {country: stats['Total
Confirmed Cases'] for country, stats in top_5_lowest}
    }

    with open('covid19_summary.json', 'w') as json_file:
        json.dump(summary_report, json_file, indent=4)

def main():
    # Directory where COVID-19 data is stored
    data_directory = "D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All
Codes\Q3"

    # Read COVID-19 data from JSON files
    covid19_data = read_covid19_data(data_directory)
```

```python
    # Calculate statistics for each country
    country_statistics = calculate_statistics(covid19_data)

    # Determine the top 5 countries
    top_5_highest, top_5_lowest = find_top_countries(country_statistics)

    # Generate and save the summary report
    generate_summary_report(country_statistics, top_5_highest, top_5_lowest)

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

OUTPUT:



Q4) 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:

- Order ID (a unique alphanumeric code)
- Customer Name
- Product Name
- Quantity
- 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:

- Invoice Number (same as the Order ID)
- Date of Purchase (current date)
- Customer Name
- Product Name
- Quantity
- Unit Price
- Total Amount

CSV FILE {orders.csv}

orders.csv

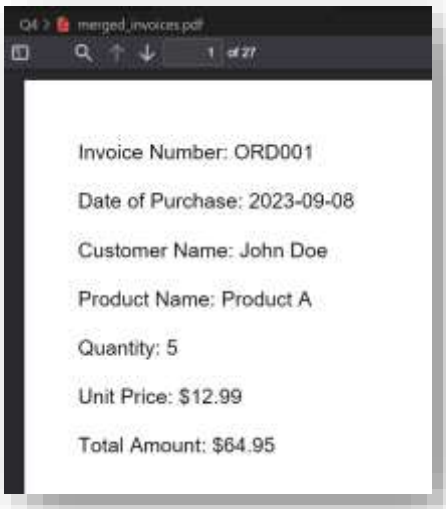
Q4 > orders.csv

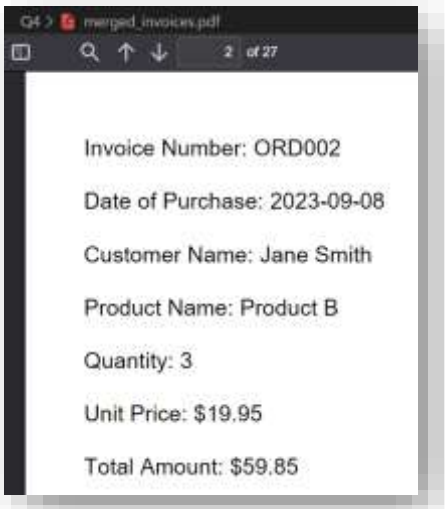
```
Order ID,Customer Name,Product Name,Quantity,Unit Price
ORD001,John Doe,Product A,5,12.99
ORD002,Jane Smith,Product B,3,19.95
ORD003,Michael Johnson,Product C,2,9.99
ORD004,Emily Davis,Product A,7,12.99
ORD005,William Wilson,Product B,4,19.95
ORD006,Susan Brown,Product C,6,9.99
ORD007,David Lee,Product A,2,12.99
ORD008,Linda Wilson,Product B,1,19.95
ORD009,Robert Johnson,Product C,3,9.99
ORD010,Sarah Taylor,Product A,8,12.99
ORD011,James Smith,Product B,5,19.95
ORD012,Karen White,Product C,4,9.99
ORD013,Thomas Davis,Product A,3,12.99
ORD014,Mary Johnson,Product B,2,19.95
ORD015,Christopher Adams,Product C,1,9.99
ORD016,Patricia Harris,Product A,4,12.99
ORD017,Michael Brown,Product B,7,19.95
ORD018,Laura Clark,Product C,5,9.99
```

CODE:

```
import pandas as pd
from fpdf import FPDF
import os
from PyPDF2 import PdfFileMerger
from datetime import date

def create_pdf_invoice(order):
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)

    pdf.cell(0, 10, f"Invoice Number: {order['Order ID']}", ln=True)
    pdf.cell(0, 10, f"Date of Purchase: {date.today()}", ln=True)
    pdf.cell(0, 10, f"Customer Name: {order['Customer Name']}", ln=True)
    pdf.cell(0, 10, f"Product Name: {order['Product Name']}", ln=True)
    pdf.cell(0, 10, f"Quantity: {order['Quantity']}", ln=True)
    pdf.cell(0, 10, f"Unit Price: ${order['Unit Price']:.2f}", ln=True)

    total_amount = order['Quantity'] * order['Unit Price']
    pdf.cell(0, 10, f"Total Amount: ${total_amount:.2f}", ln=True)

    pdf_file_name = f"invoice_{order['Order ID']}.pdf"
    pdf.output(pdf_file_name)
    return pdf_file_name

order_data = pd.read_csv('orders.csv')

pdf_invoice_files = []
for _, order in order_data.iterrows():
    pdf_invoice_file = create_pdf_invoice(order)
    pdf_invoice_files.append(pdf_invoice_file)

pdf_merger = PdfFileMerger()
for pdf_file in pdf_invoice_files:
    pdf_merger.append(pdf_file)

pdf_merger.write('merged_invoices.pdf')
pdf_merger.close()

for pdf_file in pdf_invoice_files:
    os.remove(pdf_file)

print("PDF invoices generated and merged successfully.")
```

OUTPUT:





___and 25 more___

Q5) 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:
- Read the input data from a file specified by the user.
- Process the text data by performing various operations, such as counting words, calculating character frequencies, and generating word clouds.
- 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:

- 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.

- 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.
- Disk Space Full: If the output file cannot be written due to insufficient disk space, your program should raise a custom exception DiskSpaceFullError.

CODE:

```
class FileNotFoundError(Exception):
    def __init__(self, file_path):
        self.file_path = file_path
        super().__init__(f"File not found: {file_path}")

class InvalidInputDataError(Exception):
    def __init__(self, message):
        super().__init__(f"Invalid input data: {message}")

class DiskSpaceFullError(Exception):
    def __init__(self, message):
        super().__init__(f"Disk space is full: {message}")

def read_input_data(file_path):
    try:
        with open(file_path, 'r') as file:
            # Read and process the input data
            # Add your text processing logic here
            pass
    except FileNotFoundError as e:
        raise e  # Re-raise the custom FileNotFoundError
    except Exception as e:
        raise InvalidInputDataError(str(e))  # Raise
InvalidInputDataError for unexpected input data

def store_processed_results(output_file_path, processed_data):
    try:
        with open(output_file_path, 'w') as output_file:
            # Write the processed data to the output file
            # Add your code to store the processed results here
            pass
    except IOError as e:
```

```
        raise DiskSpaceFullError(str(e))  # Raise DiskSpaceFullError
for disk space full

def main():
    input_file_path = input("Enter the input file path: ")
    output_file_path = input("Enter the output file path: ")

    try:
        input_data = read_input_data(input_file_path)
        processed_data = process_text_data(input_data)
        store_processed_results(output_file_path, processed_data)
    except FileNotFoundError as e:
        print(f"Error: {e}")
    except InvalidInputDataError as e:
        print(f"Invalid Input Data: {e}")
    except DiskSpaceFullError as e:
        print(f"Disk Space Full: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

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

OUTPUT:

```
PROBLEMS 17   OUTPUT   DEBUG CONSOLE   TERMINAL   COMMENTS

PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes> python -u "d:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\
Enter the input file path: D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes\Q5\error_handling.py
Enter the output file path: D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes\Q5
An unexpected error occurred: name 'process_text_data' is not defined
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes>
```

Q6) 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 bcrypt
import json

# File path to store user data
USER_DATA_FILE = 'user_data.json'

# Function to load user data from the file
def load_user_data():
    try:
        with open(USER_DATA_FILE, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

# Function to save user data to the file
def save_user_data(users):
    with open(USER_DATA_FILE, 'w') as file:
        json.dump(users, file)

# Function to register a new user
def register_user(username, password):
    users = load_user_data()
    if username in users:
        print("Username already exists. Please choose a different
one.")
    else:
        hashed_password = bcrypt.hashpw(password.encode('utf-8'),
bcrypt.gensalt())
```

```
        users[username] = hashed_password.decode('utf-8')
        save_user_data(users)
        print("Registration successful. You can now log in.")

# Function to log in a user
def login_user(username, password):
    users = load_user_data()
    if username not in users:
        print("User not found. Please register first.")
        return False

    hashed_password = users[username].encode('utf-8')
    if bcrypt.checkpw(password.encode('utf-8'), hashed_password):
        print("Login successful.")
        return True
    else:
        print("Incorrect password. Please try again.")
        return False
def main():
    while True:
        print("\nWelcome to the Task Manager!")
        print("1. Register")
        print("2. Log in")
        print("3. Exit")

        choice = input("Select an option: ")

        if choice == '1':
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            register_user(username, password)
        elif choice == '2':
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            if login_user(username, password):
                # Implement task management features here for logged-in
users
                pass
        elif choice == '3':
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

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

OUTPUT:

```
PS D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All Codes>

Welcome to the Task Manager!
1. Register
2. Log in
3. Exit
Select an option: 1
Enter your username: Smit
Enter your password: s1238
Registration successful. You can now log in.

Welcome to the Task Manager!
1. Register
2. Log in
3. Exit
Select an option: 2
Enter your username: Smit
Enter your password: smit123
Incorrect password. Please try again.

Welcome to the Task Manager!
1. Register
2. Log in
3. Exit
Select an option: 2
Enter your username: smix
Enter your password: 156
User not found. Please register first.

Welcome to the Task Manager!
1. Register
2. Log in
3. Exit
Select an option:
```

{} user_data.json > ...


```
1   {"Smit": "$2b$12$kwsPKLRJSpXKzyPwqMnIcuGalY9EROJXzp5.PGWcTFSi1SIYqie1a"}
```

Q7) Task: Household Expenses Tracker

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

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

7. Data Backup and Restore: Implement a backup and restore feature that allows users to save a copy of the expenses.csv file to a backup location and restore it if needed. Handle cases where the file might be missing or corrupted.

CSV FILE {expenses.csv}

expenses.csv

Q7 > expenses.csv

```csv
Name,Date,Description,Amount,Category
John,2023-09-01,Groceries,75.50,Food
Alice,2023-09-02,Utilities,120.00,Utilities
Bob,2023-09-03,Dinner with friends,45.00,Entertainment
John,2023-09-04,Gasoline,50.00,Transportation
Alice,2023-09-05,Movie night,30.00,Entertainment
Bob,2023-09-06,Shopping,100.00,Clothing
John,2023-09-07,Utilities,110.00,Utilities
Alice,2023-09-08,Groceries,85.00,Food
Bob,2023-09-09,Transportation,60.00,Transportation
John,2023-09-10,Groceries,70.00,Food
Alice,2023-09-11,Utilities,115.00,Utilities
Bob,2023-09-12,Entertainment,40.00,Entertainment
John,2023-09-13,Gasoline,45.00,Transportation
Alice,2023-09-14,Groceries,80.00,Food
```

CODE:

```python
import csv
from datetime import date
##STEP 1
def log_expense():
    name = input("Enter your name: ")
    date_str = input("Enter the date (YYYY-MM-DD): ")
    description = input("Enter the description: ")
    amount = float(input("Enter the amount spent: "))
    category = input("Enter the category: ")

    with open('C:\Aditya\College Code\VIPUL SIR\IA
Assignment\Q7\expenses.csv', mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([name, date_str, description, amount, category])

##STEP 2
import csv

def calculate_total_expenses():
    total_expenses = {}
```

```
    with open("D:\\Sem-5\\adv. python\\Vipul's py\\IA ASSIGNMENT\\All
Codes\\Q7\\expenses.csv", mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the header row
        for row in reader:
            name, _, _, amount, _ = row
            if name not in total_expenses:
                total_expenses[name] = 0
            total_expenses[name] += float(amount)

    for name, expenses in total_expenses.items():
        print(f"{name}: Total Expenses: ${expenses:.2f}")

calculate_total_expenses()

##STEP 3
import csv
import matplotlib.pyplot as plt
from collections import defaultdict

def generate_expense_trends_chart():
    date_expenses = defaultdict(float)

    with open("D:\\Sem-5\\adv. python\\Vipul's py\\IA ASSIGNMENT\\All
Codes\\Q7\\expenses.csv", mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the header row
        for row in reader:
            _, date_str, _, amount, _ = row
            date_expenses[date_str] += float(amount)

    dates = list(date_expenses.keys())
    expenses = [date_expenses[date] for date in dates]

    plt.plot(dates, expenses)
    plt.xlabel('Date')
    plt.ylabel('Cumulative Expenses')
    plt.title('Expense Trends over the Last Month')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

generate_expense_trends_chart()

##STEP 4
import csv
from datetime import date
```

```python
def log_expense():
    name = input("Enter your name: ")
    date_str = input("Enter the date (YYYY-MM-DD): ")
    description = input("Enter the description: ")
    amount = float(input("Enter the amount spent: "))
    category = input("Enter the category: ")

    with open('expenses.csv', mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([name, date_str, description, amount, category])

#STEP 5
import csv

def generate_expense_report():
    # Read the CSV file and calculate the monthly expenses report.
    expenses_by_name = {}
    expenses_by_category = {}

    with open("D:\\Sem-5\\adv. python\\Vipul's py\\IA ASSIGNMENT\\All
Codes\\Q7\\expenses.csv", mode='r') as file:
        reader = csv.reader(file)
        next(reader)  # Skip the header row

        for row in reader:
            name, _, _, amount, category = row
            amount = float(amount)

            # Total expenses by family member
            if name not in expenses_by_name:
                expenses_by_name[name] = 0
            expenses_by_name[name] += amount

            # Total expenses by category
            if category not in expenses_by_category:
                expenses_by_category[category] = 0
            expenses_by_category[category] += amount

    # Display the report
    print("Monthly Expense Report:")
    for name, total in expenses_by_name.items():
        print(f"{name}: Total Expenses: ${total:.2f}")

    print("\nExpense Breakdown by Category:")
    for category, total in expenses_by_category.items():
        print(f"{category}: Total Expenses: ${total:.2f}")

generate_expense_report()
```

```python
##STEP 6
import csv
from collections import defaultdict
def set_budget():
    category = input("Enter the category for budgeting: ")
    budget = float(input("Enter the monthly budget for this category: "))

    # Store the budget in a separate CSV file or data structure for tracking
def check_budget():
    # Calculate remaining budgets for categories and provide warnings if
exceeded
    pass

##STEP 7
import shutil

def backup_expenses():
    # Create a backup copy of the expenses.csv file in a backup folder.
    backup_folder = "D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All
Codes\Q7\BACKKK"
    try:
        shutil.copy("D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All
Codes\Q7", backup_folder)
        print("Expense data backed up successfully.")
    except FileNotFoundError:
        print("Expense data file not found. Backup failed.")

def restore_expenses():
    # Restore the expenses.csv file from the backup folder.
    backup_folder = "D:\Sem-5\adv. python\Vipul's py\IA ASSIGNMENT\All
Codes\Q7\BACKKK"
    try:
        shutil.copy(f'{backup_folder}/expenses.csv', 'expenses.csv')
        print("Expense data restored successfully.")
    except FileNotFoundError:
        print("Backup data file not found. Restore failed.")

#call the functions
log_expense()
calculate_total_expenses()
generate_expense_trends_chart()
generate_expense_report()
set_budget()
check_budget()
backup_expenses()
restore_expenses()
```