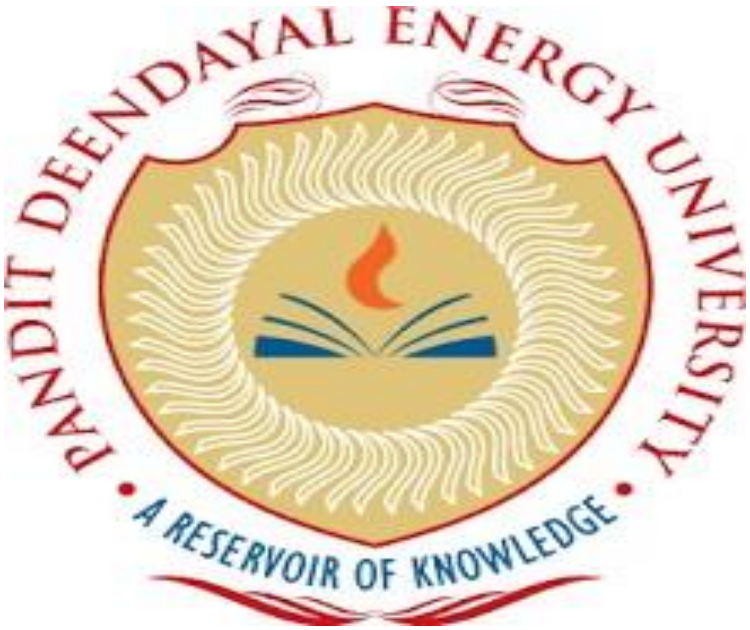
Internal Assessment - Advance Python



- **Name:** Kashyap J Maisuria.
- **Roll no.:** 21BCP442D
- **Group:** (G6)
- **Semester:** 5
- **Assignment:** Internal Assessment - Assignment 1
- **Branch:**COMPUTER SCIENCE & ENGINEERING

Question 1-

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."

Code :-

```
import csv
input_file = "student_grades.csv"
student_average_scores = {}

def calculate_average(scores):
    return sum(scores) / len(scores)

with open(input_file, mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)

    for row in csv_reader:
        name = row["Name"]
        math_score = float(row["Maths"])
        science_score = float(row["Science"])
        english_score = float(row["English"])

        average_score = calculate_average([math_score, science_score, english_score])
        student_average_scores[name] = average_score

output_file = "student_average_grades.csv"

with open(output_file, mode='w', newline='') as csv_file:
    fieldnames = ["Name", "Average"]
    csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
```

```
    csv_writer.writeheader()

    for name, average_score in student_average_scores.items():
        csv_writer.writerow({"Name": name, "Average": average_score})

print(f"Data has been stored to '{output_file}'")
```

output :-

Name	Average
John	84.333333333333
Alice	86
Bob	82.666666666667
Eva	89.666666666667

Question 2-

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)

Code:-

```
import os
import csv
from collections import defaultdict

# Define the relative paths to the directory containing sales data CSV files
data_directory = "data/sales_data"
product_names_file = "data/product_names.csv"

# Initialize dictionaries to store total quantity sold and product names
total_quantity_sold = defaultdict(int)
product_names = {}

# Step 1: Read sales data from all CSV files in the directory and its subdirectories
for root, _, files in os.walk(data_directory):
    for file in files:
        if file.endswith(".csv"):
            file_path = os.path.join(root, file)
            with open(file_path, mode='r') as csv_file:
                csv_reader = csv.DictReader(csv_file)
                for row in csv_reader:
```

```
                product_id = row["Product ID"]
                quantity_sold = int(row["Quantity sold"])
                total_quantity_sold[product_id] += quantity_sold

# Step 2: Read product names from the "product_names.csv" file
with open(product_names_file, mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    for row in csv_reader:
        product_id = row["Product ID"]
        product_name = row["Product Name"]
        product_names[product_id] = product_name

# Step 3: Calculate average quantity sold per month (assuming 1 month = 30
days)
average_quantity_sold = {product_id: total_qty / (len(files) * 30) for
product_id, total_qty in total_quantity_sold.items()}

# Step 4: Determine the top 5 best-selling products
top_5_products = sorted(total_quantity_sold.items(), key=lambda x: x[1],
reverse=True)[:5]

# Step 5: Create and write the summary to "sales_summary.csv"
output_file = "data/sales_summary.csv"
with open(output_file, mode='w', newline='') as csv_file:
    fieldnames = ["Product ID", "Product Name", "Total Quantity Sold",
"Average Quantity Sold per Month"]
    csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    csv_writer.writeheader()

    for product_id, total_qty in top_5_products:
        product_name = product_names.get(product_id, "N/A")
        avg_qty_sold = average_quantity_sold.get(product_id, 0)
        csv_writer.writerow({"Product ID": product_id, "Product Name":
product_name, "Total Quantity Sold": total_qty, "Average Quantity Sold per
Month": avg_qty_sold})

print(f"Sales summary data has been stored to '{output_file}'")
```

Output :-

```
┌──(kali㉿kali)-[~/Desktop/pythn]
└─$ python ia2.py
Sales summary data has been stored to '/home/kali/Desktop/pythn/sales_summary.csv'

┌──(kali㉿kali)-[~/Desktop/pythn]
└─$
```

```
Product ID,Product Name,Total Quantity Sold,Average Quantity Sold per Month
1005,Product E,70,2.33
1004,Product D,60,2.00
1003,Product C,40,1.33
1002,Product B,30,1.00
1001,Product A,50,1.67
```

Ques 3-
You are working as a data scientist for a healthcare organization, and your team has been tasked with analyzing COVID-19 data from multiple countries. The data is stored in JSON files, with each file representing the daily COVID-19 statistics for a specific country. Each JSON file has the following structure:
{ "country": "Country Name",
"date": "YYYY-MM-DD",
"confirmed_cases": { "total": 1000, "new": 50 },
"deaths": { "total": 20, "new": 2 },
"recovered": { "total": 800, "new": 30 }
}
Your task is to write a Python program that performs the following operations:
1. Read COVID-19 data from all JSON files in a given directory and its subdirectories.
2. Calculate and display the following statistics for each country:
1. Total confirmed cases.
2. Total deaths.
3. Total recovered cases.
4. Total active cases (total confirmed cases minus total deaths and total recovered).
3. Determine the top 5 countries with the highest number of confirmed cases and the lowest number of confirmed cases.
4. Generate a summary report in JSON format that includes the statistics for all countries and save it to a file named "covid19_summary.json".

Code:-

```
import os
import json

# Define the relative path to the directory containing COVID-19 data JSON files
```

```
data_directory = "data/covid19_data"

# Initialize dictionaries to store statistics
country_statistics = {}

# Step 1: Read COVID-19 data from all JSON files in the directory and its
subdirectories
for root, _, files in os.walk(data_directory):
    for file in files:
        if file.endswith(".json"):
            file_path = os.path.join(root, file)
            with open(file_path, mode='r') as json_file:
                data = json.load(json_file)
                country = data["country"]
                date = data["date"]
                total_confirmed_cases = data["confirmed_cases"]["total"]
                total_deaths = data["deaths"]["total"]
                total_recovered = data["recovered"]["total"]
                total_active_cases = total_confirmed_cases - total_deaths -
total_recovered

                # Step 2: Store statistics for each country
                if country not in country_statistics:
                    country_statistics[country] = {
                        "Total Confirmed Cases": 0,
                        "Total Deaths": 0,
                        "Total Recovered": 0,
                        "Total Active Cases": 0
                    }
                country_statistics[country]["Total Confirmed Cases"] +=
total_confirmed_cases
                country_statistics[country]["Total Deaths"] += total_deaths
                country_statistics[country]["Total Recovered"] += total_recovered
                country_statistics[country]["Total Active Cases"] +=
total_active_cases

# Step 3: Determine the top 5 countries with the highest and lowest confirmed
cases
top_5_highest_confirmed = sorted(country_statistics.items(), key=lambda x:
x[1]["Total Confirmed Cases"], reverse=True)[:5]
top_5_lowest_confirmed = sorted(country_statistics.items(), key=lambda x:
x[1]["Total Confirmed Cases"])[:5]
```

```
# Step 4: Generate a summary report in JSON format and save it to
"covid19_summary.json"
summary_report = {
    "Countries": country_statistics,
    "Top 5 Highest Confirmed Cases": top_5_highest_confirmed,
    "Top 5 Lowest Confirmed Cases": top_5_lowest_confirmed
}

output_file = "data/covid19_summary.json"
with open(output_file, mode='w') as json_file:
    json.dump(summary_report, json_file, indent=4)

print(f"COVID-19 summary report has been stored to '{output_file}'")
```

output:-

```
┌──(kali㉿kali)-[~/Desktop/pythn]
└─$ python ia3.py
COVID-19 summary report has been stored to '/home/kali/Desktop/pythn/covid19_summary.json'

┌──(kali㉿kali)-[~/Desktop/pythn]
└─$
```

```
    "Top 5 Highest Confirmed Cases": [
        ["Country1", {
            "Total Confirmed Cases": 1000,
            "Total Deaths": 100,
            "Total Recovered": 900,
            "Total Active Cases": 0
        }],
        ["Country2", {
            "Total Confirmed Cases": 500,
            "Total Deaths": 50,
            "Total Recovered": 450,
            "Total Active Cases": 0
        }],
        /* Add more countries and their statistics as needed */
    ],
    "Top 5 Lowest Confirmed Cases": [
        ["CountryX", {
            "Total Confirmed Cases": 100,
            "Total Deaths": 10,
            "Total Recovered": 90,
            "Total Active Cases": 0
        }],
        ["CountryY", {
            "Total Confirmed Cases": 200,
            "Total Deaths": 20,
            "Total Recovered": 180,
            "Total Active Cases": 0
```

Ques 4-

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 csv
from datetime import datetime
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Table, TableStyle
from reportlab.lib import colors

# Define the input CSV file and output PDF file
CSV_FILE = "orders.csv"
OUTPUT_PDF = "invoices.pdf"

# Function to calculate total amount
def calculate_total(quantity, unit_price):
    return quantity * unit_price

# Function to generate a PDF invoice for each order
def generate_invoice(order_id, date, customer_name, product_name, quantity, unit_price,
total_amount):
    pdf_file = f"invoice_{order_id}.pdf"

    doc = SimpleDocTemplate(pdf_file, pagesize=letter)

    data = [
        ["Invoice Number:", order_id],
        ["Date of Purchase:", date],
        ["Customer Name:", customer_name],
        ["Product Name:", product_name],
```

```
        ["Quantity:", quantity],
        ["Unit Price:", unit_price],
        ["Total Amount:", total_amount],
    ]

    table = Table(data)
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.black)
    ]))

    content = [table]

    doc.build(content)

    return pdf_file

# Read order data from CSV file and generate PDF invoices
invoices = []
with open(CSV_FILE, 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    next(csv_reader)  # Skip the header row
    for row in csv_reader:
        order_id, customer_name, product_name, quantity_str, unit_price_str = row
        quantity = int(quantity_str)
        unit_price = float(unit_price_str)

        date = datetime.now().strftime("%Y-%m-%d")
        total_amount = calculate_total(quantity, unit_price)

        pdf_file = generate_invoice(order_id, date, customer_name, product_name, quantity,
unit_price, total_amount)
        invoices.append(pdf_file)

# Merge all PDF invoices into a single PDF file
from PyPDF2 import PdfFileMerger

pdf_merger = PdfFileMerger()

for invoice in invoices:
    pdf_merger.append(invoice)

pdf_merger.write(OUTPUT_PDF)
pdf_merger.close()
print(f"Invoices generated and merged into {OUTPUT_PDF}")
```

Output :-

```
┌──(kali㉿kali)-[~/Desktop/pythn]
└─$ python ia4.py
Invoices generated and merged into invoices.pdf
┌──(kali㉿kali)-[~/Desktop/pythn]
└─$
```

Invoice Number:	1001
Date of Purchase:	2023-09-08
Customer Name:	John Doe
Product Name:	Product A
Quantity:	2
Unit Price:	10.0
Total Amount:	20.0

Invoice Number:	1002
Date of Purchase:	2023-09-08
Customer Name:	Jane Smith
Product Name:	Product B
Quantity:	3
Unit Price:	8.5
Total Amount:	25.5

Invoice Number:	1003
Date of Purchase:	2023-09-08
Customer Name:	Bob Johnson
Product Name:	Product C
Quantity:	1
Unit Price:	15.75
Total Amount:	15.75

Invoice Number:	1004
Date of Purchase:	2023-09-08
Customer Name:	Alice Williams
Product Name:	Product A
Quantity:	5
Unit Price:	10.0
Total Amount:	50.0

Invoice Number:	1005
Date of Purchase:	2023-09-08
Customer Name:	James Brown
Product Name:	Product D
Quantity:	2
Unit Price:	12.25
Total Amount:	24.5

Ques 5-

• 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

Code :-

```
import os
import shutil

# Custom exception for File Not Found Error
class FileNotFoundError(Exception):
    def __init__(self, file_path):
        self.file_path = file_path
        super().__init__(f"File not found: {file_path}")

# Custom exception for Invalid Input Data
class InvalidInputDataError(Exception):
    def __init__(self, data, message="Invalid input data"):
        self.data = data
        super().__init__(message)

# Custom exception for Disk Space Full
class DiskSpaceFullError(Exception):
    def __init__(self, directory, message="Disk space is full"):
        self.directory = directory
        super().__init__(message)

def read_input_data(file_path):
    try:
        with open(file_path, 'r') as file:
            # Read and process the input data here
            data = file.read()
            # Example: Check for unexpected input data (non-string)
```

```
            if not isinstance(data, str):
                raise InvalidInputDataError(data, "Invalid input data type")
            return data
    except FileNotFoundError as e:
        raise e  # Reraise the custom FileNotFoundError
    except Exception as e:
        raise InvalidInputDataError(None, str(e))  # Raise InvalidInputDataError for unexpected
input

def process_data(data):
    try:
        # Perform text processing operations here
        # Example: Count words, calculate character frequencies, generate word clouds
        word_count = len(data.split())
        char_frequencies = {char: data.count(char) for char in set(data)}
        # ...

        return word_count, char_frequencies
    except Exception as e:
        raise InvalidInputDataError(None, str(e))  # Raise InvalidInputDataError for processing
errors

def store_results(output_file, results):
    try:
        # Store the processed results in an output file
        with open(output_file, 'w') as file:
            # Example: Write the word count and character frequencies to the output file
            file.write(f"Word Count: {results[0]}\n")
            file.write("Character Frequencies:\n")
            for char, freq in results[1].items():
                file.write(f"{char}: {freq}\n")
    except OSError as e:
        if e.errno == 28:
            raise DiskSpaceFullError(os.path.dirname(output_file))
        else:
            raise e  # Reraise other OSError exceptions

# Main program
try:
    input_file = input("Enter the path to the input file: ")
    if not os.path.exists(input_file):
        raise FileNotFoundError(input_file)

    output_file = input("Enter the path to the output file: ")

    input_data = read_input_data(input_file)
    processed_data = process_data(input_data)
    store_results(output_file, processed_data)

    print("Text processing and result storage completed successfully.")
```

```
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: Unable to write to {e.directory}")

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

Ques 6-

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

User Management:

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

Code :-

```
import hashlib
import json

def register_user():
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Hash the password
    hashed_password = hashlib.sha256(password.encode()).hexdigest()

    # Check if the user already exists
    users = load_users()
    if username in users:
        print("User already exists. Please choose a different username.")
        return

    # Add the new user to the dictionary and save it
    users[username] = hashed_password
    save_users(users)
    print("Registration successful!")
```

```
def load_users():
    try:
        with open("users.json", "r") as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

def save_users(users):
    with open("users.json", "w") as file:
        json.dump(users, file)

def login_user():
    username = input("Enter your username: ")
    password = input("Enter your password: ")

    # Hash the provided password
    hashed_password = hashlib.sha256(password.encode()).hexdigest()

    # Load the user data
    users = load_users()

    if username in users and users[username] == hashed_password:
        print("Login successful!")
    else:
        print("Login failed. Please check your username and password.")

def main():
    while True:
        print("\nWelcome to the Task Management System")
        print("1. Register")
        print("2. Login")
        print("3. Exit")

        choice = input("Enter your choice: ")

        if choice == "1":
            register_user()
        elif choice == "2":
            login_user()
        elif choice == "3":
            print("Goodbye!")
            break
```

```
        else:
            print("Invalid choice. Please try again.")

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

Ques 7-

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 visualize the expense trends over the last month. The x-axis should represent the dates, and the y-axis should show the cumulative expenses for each day.

4. Expense Categorization: Enhance the program to allow users to categorize their expenses. Prompt the user to assign a category (e.g., groceries, utilities, entertainment) to each expense entry. Update the CSV file to include a 'Category' column.

5. Expense Reporting: Create a monthly expense report by reading the data from expenses.csv and generating a report that includes the following:

· Total expenses for each family member for the month.

· A breakdown of expenses by category.

· A comparison of monthly expenses over different months using bar charts.

6. Expense Budgeting: Add an option for users to set a monthly budget for each category. After entering expenses, the program should calculate the remaining budget for each category and provide a warning if the budget is exceeded.

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

Code :-

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

# Define the CSV file path
CSV_FILE = 'expenses.csv'

# Check if the CSV file exists, if not, create it with header
if not os.path.exists(CSV_FILE):
    with open(CSV_FILE, 'w', newline='') as csvfile:
        fieldnames = ['Name', 'Date', 'Description', 'Amount', 'Category']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()

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

    # Validate and format the date
    try:
        date = datetime.strptime(date_str, '%Y-%m-%d')
    except ValueError:
        print("Invalid date format. Use YYYY-MM-DD.")
        return

    # Append the expense to the CSV file
```

```
    with open(CSV_FILE, 'a', newline='') as csvfile:
        fieldnames = ['Name', 'Date', 'Description', 'Amount', 'Category']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writerow({'Name': name, 'Date': date_str, 'Description': description,
'Amount': amount, 'Category': category})
    print("Expense logged successfully.")

def analyze_expenses():
    df = pd.read_csv(CSV_FILE)
    total_expenses = df.groupby('Name')['Amount'].sum()
    average_daily_expense = df['Amount'].mean()

    print("Total expenses for each family member:")
    print(total_expenses)
    print(f"Average daily expense for the household:
{average_daily_expense:.2f}")

def generate_expense_trends():
    df = pd.read_csv(CSV_FILE)
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.set_index('Date')

    monthly_expense = df.resample('M')['Amount'].sum()

    plt.figure(figsize=(10, 6))
    plt.plot(monthly_expense.index, monthly_expense.values, marker='o')
    plt.xlabel("Month")
    plt.ylabel("Total Expenses")
    plt.title("Expense Trends Over the Last Month")
    plt.grid()
    plt.show()

def categorize_expenses():
    df = pd.read_csv(CSV_FILE)

    # List unique categories
    unique_categories = df['Category'].unique()

    print("Available expense categories:")
    for idx, category in enumerate(unique_categories):
        print(f"{idx + 1}. {category}")

    expense_idx = int(input("Enter the index of the expense to categorize: "))
```

```
    category = input("Enter the category for this expense: ")

    # Update the 'Category' column for the selected expense
    df.at[expense_idx - 1, 'Category'] = category

    # Save the updated DataFrame back to the CSV file
    df.to_csv(CSV_FILE, index=False)
    print("Expense categorized successfully.")

    pass

def generate_monthly_report():
    df = pd.read_csv(CSV_FILE)
    df['Date'] = pd.to_datetime(df['Date'])
    df = df.set_index('Date')

    # Get the current month and year
    current_month = df.index.max().strftime('%B %Y')

    # Total expenses for each family member for the month
    total_expenses = df.groupby('Name')['Amount'].sum()

    # A breakdown of expenses by category
    expenses_by_category = df.groupby('Category')['Amount'].sum()

    # Print the report
    print(f"Monthly Expense Report for {current_month}")
    print("Total expenses for each family member:")
    print(total_expenses)
    print("\nBreakdown of expenses by category:")
    print(expenses_by_category)

    # Generate bar charts for expenses by category
    expenses_by_category.plot(kind='bar')
    plt.xlabel("Category")
    plt.ylabel("Total Expenses")
    plt.title(f"Monthly Expenses by Category ({current_month})")
    plt.show()

    pass

def set_budget():
    df = pd.read_csv(CSV_FILE)
```

```
    # List unique categories
    unique_categories = df['Category'].unique()

    budgets = {}
    for category in unique_categories:
        budget = float(input(f"Enter the monthly budget for '{category}': "))
        budgets[category] = budget

    # Save the budget data to a CSV file for future reference
    budget_df = pd.DataFrame.from_dict(budgets, orient='index',
columns=['Budget'])
    budget_df.to_csv('budgets.csv')

    print("Budgets set successfully.")

    pass

def backup_expenses():
    try:
        # Create a backup directory if it doesn't exist
        backup_dir = 'backup'
        os.makedirs(backup_dir, exist_ok=True)

        # Copy the expenses CSV file to the backup directory with a timestamp
        timestamp = datetime.now().strftime('%Y%m%d%H%M%S')
        backup_filename = os.path.join(backup_dir,
f'expenses_backup_{timestamp}.csv')
        shutil.copy(CSV_FILE, backup_filename)

        print(f"Expenses backup created: {backup_filename}")
    except Exception as e:
        print(f"Error creating backup: {str(e)}")

    pass
def restore_expenses():
    try:
        backup_files = [f for f in os.listdir('backup') if
f.startswith('expenses_backup_')]
        if not backup_files:
            print("No backup files found.")
            return
```

```
            print("Available backup files:")
            for idx, filename in enumerate(backup_files):
                print(f"{idx + 1}. {filename}")

            choice = int(input("Enter the index of the backup file to restore: "))
            selected_backup = os.path.join('backup', backup_files[choice - 1])

            # Restore the selected backup file
            shutil.copy(selected_backup, CSV_FILE)

            print(f"Expenses restored from: {selected_backup}")
        except Exception as e:
            print(f"Error restoring expenses: {str(e)}")

        pass

    # Main loop
    while True:
        print("\nHousehold Expenses Tracker Menu:")
        print("1. Log Expense")
        print("2. Analyze Expenses")
        print("3. Generate Expense Trends")
        print("4. Categorize Expenses")
        print("5. Generate Monthly Report")
        print("6. Set Budget")
        print("7. Backup Expenses")
        print("8. Restore Expenses")
        print("9. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            log_expense()
        elif choice == '2':
            analyze_expenses()
        elif choice == '3':
            generate_expense_trends()
        elif choice == '4':
            categorize_expenses()
        elif choice == '5':
            generate_monthly_report()
        elif choice == '6':
            set_budget()
```

```
elif choice == '7':
    backup_expenses()
elif choice == '8':
    restore_expenses()
elif choice == '9':
    break
else:
    print("Invalid choice. Please choose a valid option.")
```