PROJECT REPORT ON PERSONAL EXPENSE TRACKER

Name:	Adit Prasad
Registration No.	25BCE10115
Course:	Introduction to Problem Solving

1. Problem Definition

1.1 Background

I know how hard it can be, for the university student. The university student finds managing the pocket money a struggle. I have seen that the university student often spends money on snacks on printouts or on travel, without thinking about the pocket money. The university student watches the monthly pocket money add up quickly and then disappear. By the end of the month the university student usually runs out of cash. The university student has no idea where the cash went. The university student feels lost. The university student tried writing the pocket money details in a notebook. I see the university student always forgets the notebook. The university student also loses the page.

1.2 The Problem

The main problem is the lack of a way to track these expenses. The lack of a way to track these expenses makes keeping track for me. I have tried notes. Mental notes do not work. Mental notes fail when I try to remember each purchase. I have tried writing the expenses. Writing the expenses down is tedious. Writing the expenses down takes time I do not have. The Play Store offers apps. The apps, on the Play Store require account sign-ups. The apps, on the Play Store have ads. I look at the apps, on the Play Store. The apps on the Play Store are too complicated, for what I need.

1.3 The Solution

I used my Python programming skills to solve the problem for myself. I built a Personal Expense Tracker that runs on my laptop. The Personal Expense Tracker lets me log expenses away. The Personal Expense Tracker saves the expenses to a file so the data stays there when I open the Personal Expense Tracker the day. The goal was to build a Personal Expense Tracker that just works without any fluff. I like that the Personal Expense Tracker does what I need.

2. Requirement Analysis

Before I started coding I sat down. Wrote a list of what the program must do. I did not want to make the program too complex. I made sure the program stayed simple. I kept the program to the essentials.

2.1 Functional Requirements

I wanted the system to do the CRUD operations. The CRUD operations are Create, Read, Update, Delete.

- Adding Data: Adding Data needs you to enter the product name the category it belongs to like food or travel the price and the date. You type each piece of information in the form.

- Saving Data was a must. Saving Data made the program write all data to the expenses.csv file. Saving Data meant that I did not have to press the Save button each time.

- Viewing & Searching: I needed a way to see everything I have bought. I needed a way to search for an item, by its ID to check the date, for that item.

- Fixing Mistakes: I often type the number. Then I notice a mistake. I need a way to edit the entry. For example I might type 500 of 50. I also need a way to delete the entry entirely.

- The Reality Check: I am looking for a feature that calculates the Total Spent and shows the category breakdown. The feature lets me see if the Total Spent on food is too high.

2.2 Non-Functional Requirements

• Simplicity: The menu needed to be clear. Simplicity meant I did not have to guess which number, to press.

• Error Proofing: Error proofing means that if I type Ten of 10 for the price the program does not crash or close. Error proofing makes the program handle the mistake. Error proofing must be strong enough to restart. I want the program to keep working.

2.3 Hardware and Software

- OS: I built the program on Windows 11. The program is written in Python so the program runs fine on my friend’s Mac too.

- Tools: I used PyCharm to write the code. I like using PyCharm because PyCharm catches syntax errors. The code also runs in IDLE.

- Libraries: I used the csv library. The csv library is part of the library. The csv library let me avoid installing anything extra.

3. Top-Down Design / Modularization

When I first started I thought about writing the code in one file. I realized that the big file would become a mess. I realized that the big file would make debugging impossible. If the save function broke I did not want to scroll through two hundred lines of print statements to find the save function. I kept the code in files.

I chose the Top-Down approach. I split the project into two files:

3.1 Module Structure

1. Modules2.py (The Backend): I treat Modules2.py like the engine of the car. Modules2.py does the work. Modules2.py calculates totals. Modules2.py writes to the CSV file. Modules2.py modifies the list. Modules2.py does not interact with the user. Modules2.py only processes data.

2. Main.py (The Frontend): I wrote the dashboard. The dashboard shows the menu. The dashboard asks the user, for input. The dashboard tells Modules2 what to do. The separation made the code cleaner.

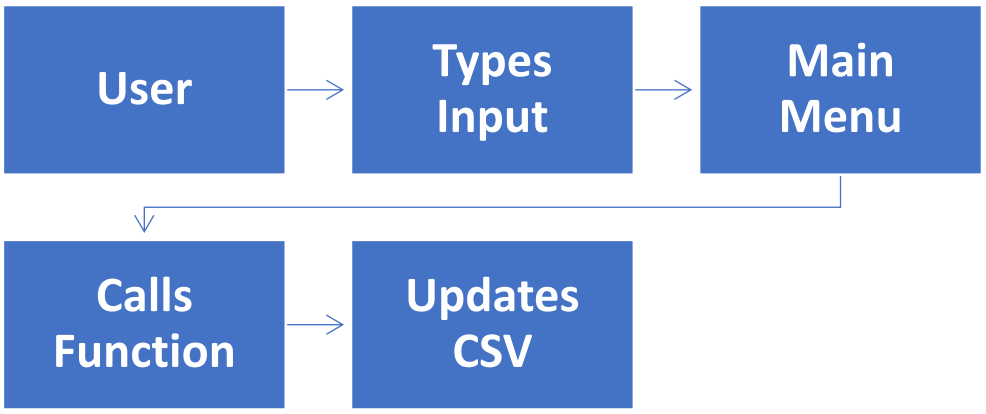
3.2 Data Structure Design

I had to decide how to store the data, in Python memory while the program ran. I tried the list. I tried the dictionary. When I tried the list I saw that I had to scan each entry to find the expense, with ID #105. When I tried the dictionary I saw that I could use the ID as the key and pull the data in one step. The dictionary is much faster.

Structure:

- Key: Product ID (Integer)
- Value: List [Name, Category, Price, Date]

Visual Data Flow



The diagram will show the User typing input. The diagram will show the Main Menu calling the Function. The diagram will show the Function updating the CSV.

4. Algorithm Development

I found that jumping into code often makes me get stuck. I write out the logic, for the parts first.

4.1 Algorithm: Add Expense & Autosave

The hardest part was making sure the unique IDs stayed unique and also needed data saving away.

1. Start.

2. Look in the dictionary for the existing ID number. Make sure the highest existing ID number does not overwrite a record.

3. Add 1 to that number to get the new ID.

4. Input: Ask the user for Name, Category, Price, and Date.

5. Sanitize: I made sure the Category is uppercase. For example I turned the Category food into FOOD. Later the report looks clean.

6. Update: I will add the entry, to the Dictionary. The Dictionary now includes the entry.

7. Autosave: The Autosave decision mattered. I wrote the code that opens expenses.csv. The code saves the data after I add the data. Autosave stops data loss if the computer crashes.

8. Stop.

4.2 Algorithm: Category Analysis

I wanted to see where my money went. I wrote this logic for the money:

1. Create an empty "totals" bucket (dictionary).
2. Check each expense that I have recorded.
3. Look at the category (e.g., "FOOD").
4. When FOOD is already, in the bucket add the price to FOOD.
5. If FOOD is not already, in the bucket start an entry for FOOD. Then FOOD will be, in the bucket.
6. Finally, print out the bucket list.

5. Implementation

Writing code taught me things. When I wrote the code I ran into errors. The errors were mostly, about file handling. I fixed the errors. Below are the parts of the code that I find interesting.

5.1 File Handling (Loading Data)

I ran the program and the program crashed because expenses.csv did not exist yet. I was surprised by the crash. I added a try-except block to the program to catch the missing file. Now when the file does not exist the program treats the situation as a start. The program creates a dictionary of showing an error. The program now runs without error.

```
def load_from_csv():
    # Initialize with default headers
    Expense = {'ID': ['Product Name', 'Product Category', 'Product Price', 'Date of 
Purchase']}

    try:
        # The newline='' is crucial for Windows!
        f = open(CSV_FILE, 'r', newline='', encoding='utf-8')
    except:
        # If file is missing (first run), just return empty dict
        return Expense

    reader = csv.reader(f)
    rows = list(reader)
    # ... parsing logic ...
    return Expense
```

5.2 Editing Logic

Editing was hard because I did not want to type the entry just to change the price. I built a sub-menu. The user can pick what the user wants to fix.

```
def Edit_Expense(n):
    searchby = int(input("Enter Product ID to Edit: "))
    # ... search loop ...
    if i == searchby:
        print("What do you want to change?")
        print("1. Name  2. Category  3. Price  4. Date")
        choice = int(input("Choice: "))

        if choice == 3:
            # Specific logic just for Price
            new_price = int(input("Enter New Price: "))
            n[searchby][2] = new_price
            flag = True
```

5.3 The Main Loop

I used a while loop in the file. The while True loop keeps the program alive. The while True loop stops when I tell the program to exit. The while True loop also handles autosave for actions. The while True loop works.

```
while True:
    print('1.Add  2.View  3.Search ... 9.Exit')
    choice = int(input('Choice: '))

    if choice == 1:
        m.Add_Expense(Expense)
        m.save_to_csv(Expense)
#Continues
    elif choice == 9:
        m.save_to_csv(Expense)
        print('Saved to expenses.csv')
        break
```

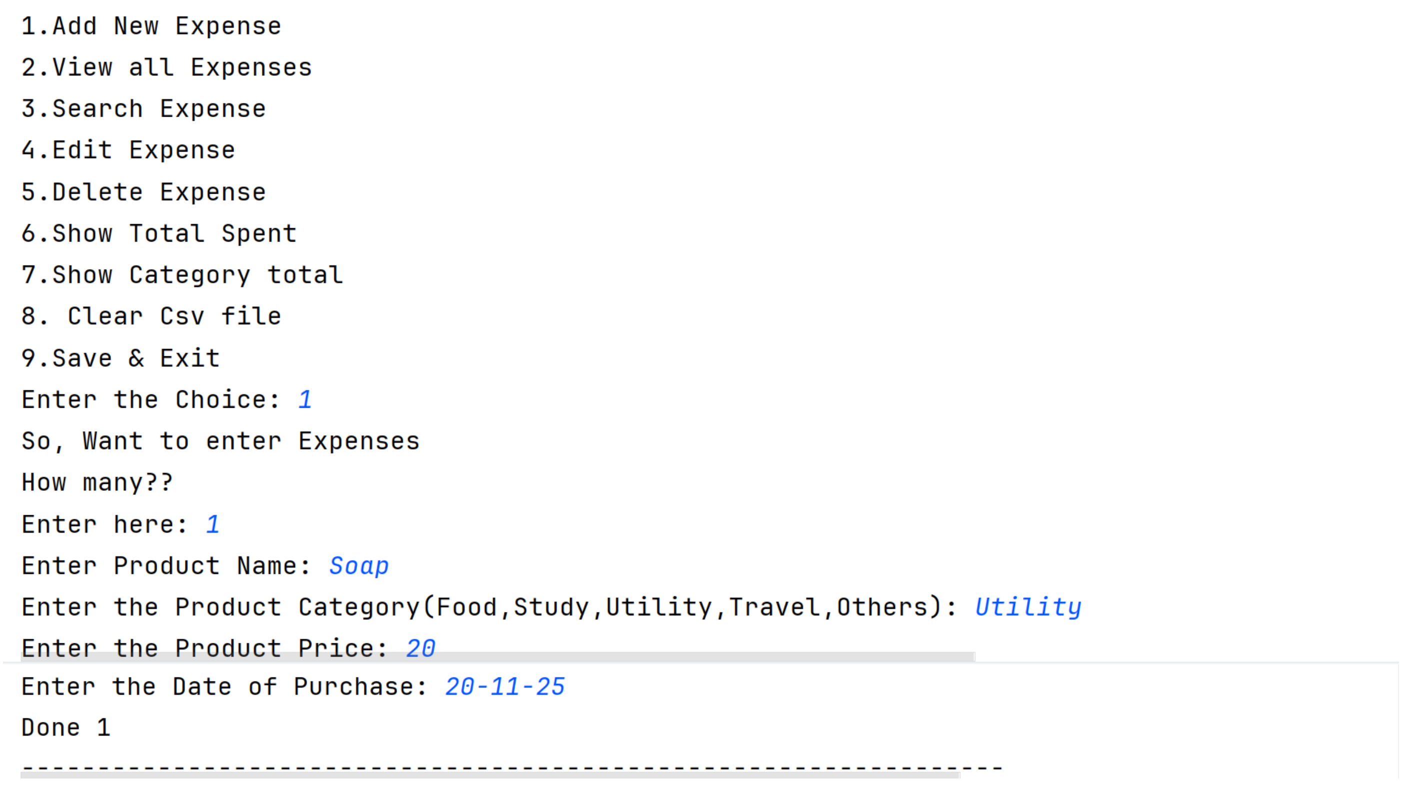
6. Testing and Refinement

I spent hours trying to break my code. I wanted to see if the code could hold up. I kept testing the code until the code would not break.

Case ID	What I Tested	Input	Result	Status
TC-01	**The Happy Path**	"Pizza", "Food", 500, "12-12-23"	Saved correctly.	**Pass**
TC-02	**The "Text" Price**	Price: "Five Hundred"	Crashed initially.	**Fixed** (Added int conversion)
TC-03	**Math Check**	Food: 100 + Food: 200	Total: 300	**Pass**
TC-04	**Deleting Ghosts**	Delete ID 500 (does not exist)	Printed "Record Not Found"	**Pass**
TC-05	**Fresh Install**	Deleted `expenses.csv` and ran code	Created new file automatically.	**Pass**

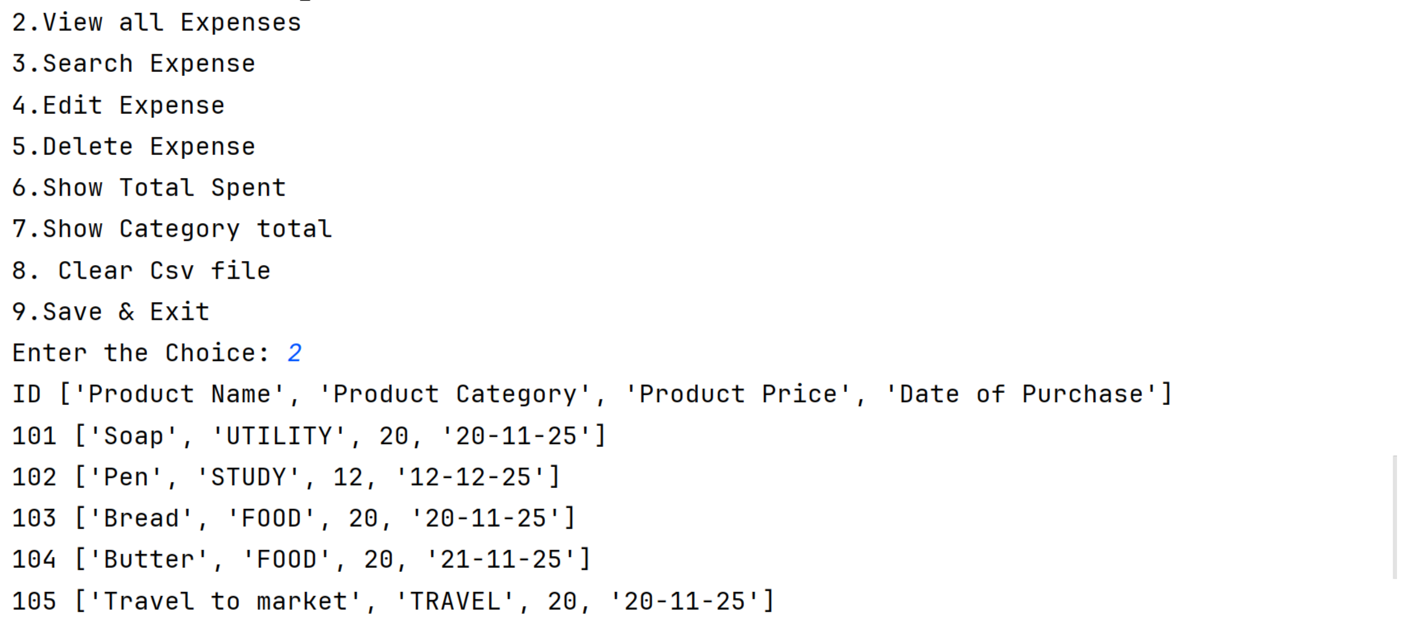
7. Screenshot And Sample Input Output

7.1 Add New Expense



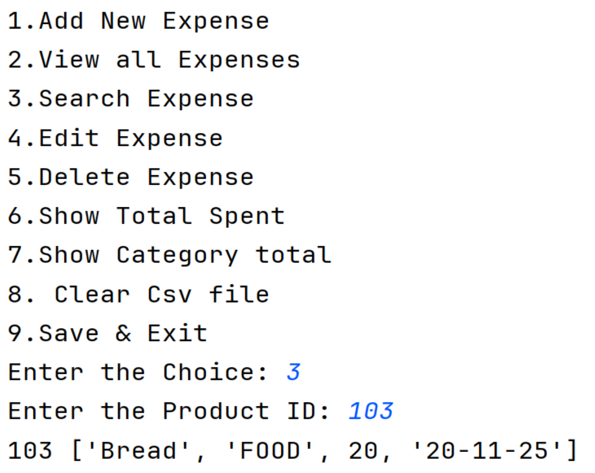
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 1
So, Want to enter Expenses
How many??
Enter here: 1
Enter Product Name: Soap
Enter the Product Category(Food,Study,Utility,Travel,Others): Utility
Enter the Product Price: 20
Enter the Date of Purchase: 20-11-25
Done 1
----------------------------------------------------------------
```

7.2 View All Expenses



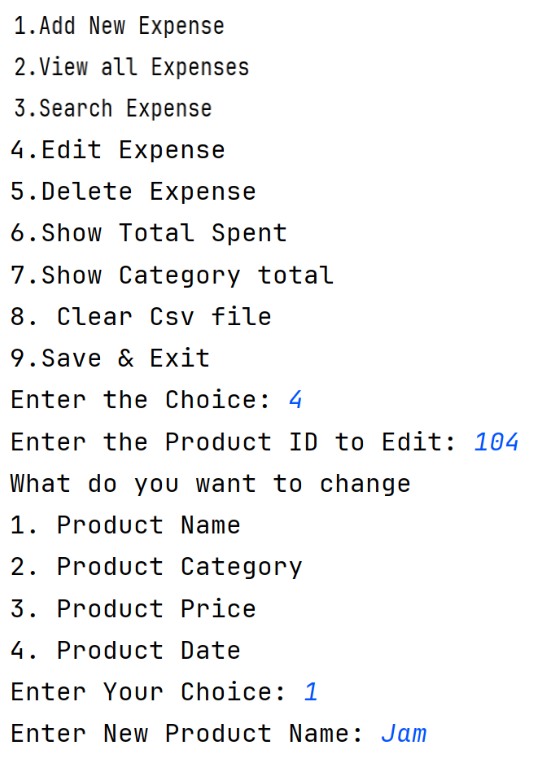
```
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 2
ID ['Product Name', 'Product Category', 'Product Price', 'Date of Purchase']
101 ['Soap', 'UTILITY', 20, '20-11-25']
102 ['Pen', 'STUDY', 12, '12-12-25']
103 ['Bread', 'FOOD', 20, '20-11-25']
104 ['Butter', 'FOOD', 20, '21-11-25']
105 ['Travel to market', 'TRAVEL', 20, '20-11-25']
```

7.3 Search Expenses



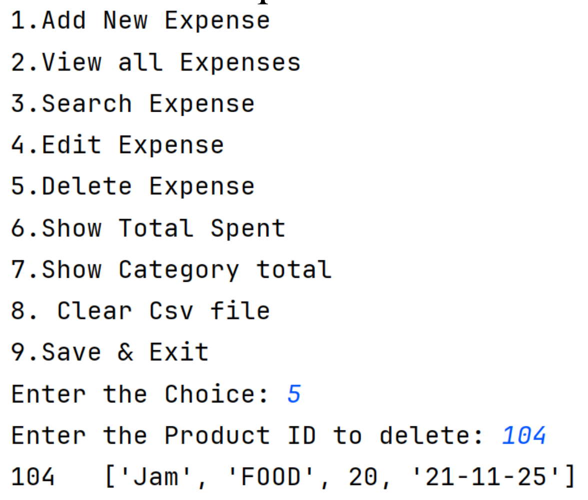
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 3
Enter the Product ID: 103
103 ['Bread', 'FOOD', 20, '20-11-25']
```

7.4 Edit Expense



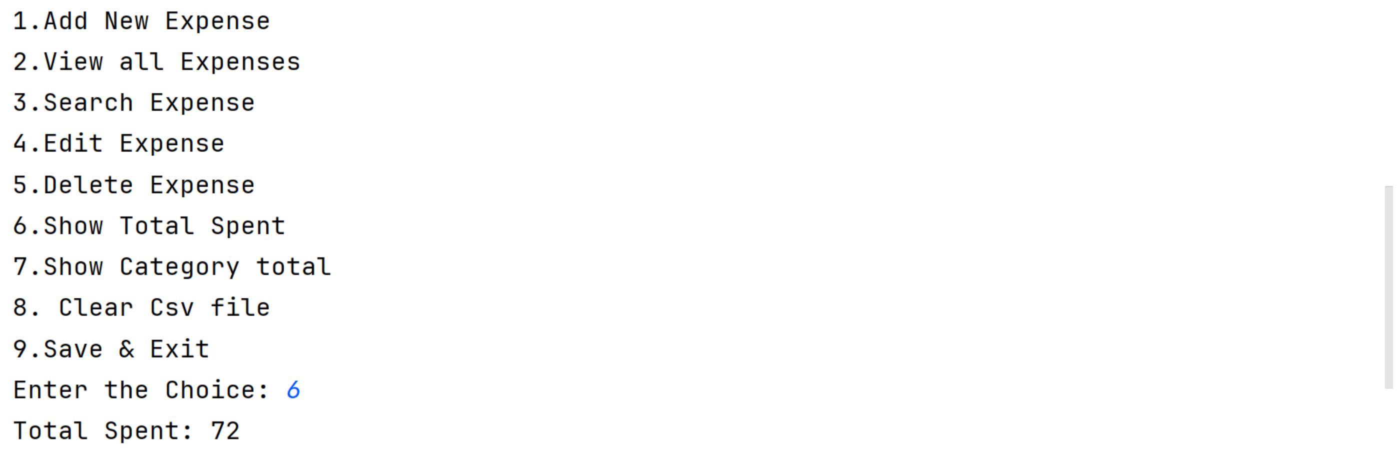
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 4
Enter the Product ID to Edit: 104
What do you want to change
1. Product Name
2. Product Category
3. Product Price
4. Product Date
Enter Your Choice: 1
Enter New Product Name: Jam
```

7.5 Delete Expense



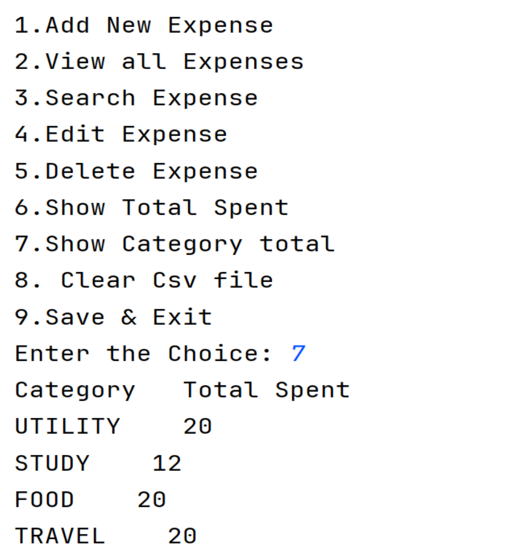
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 5
Enter the Product ID to delete: 104
104   ['Jam', 'FOOD', 20, '21-11-25']
```

7.6 Show Total Spent



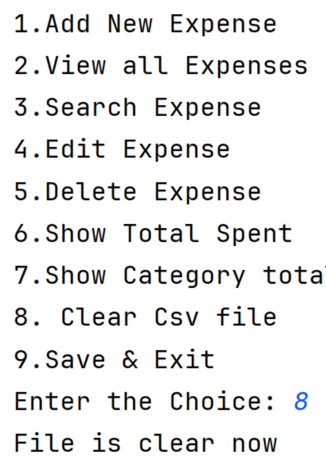
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 6
Total Spent: 72
```

7.7 Show Category wise Total



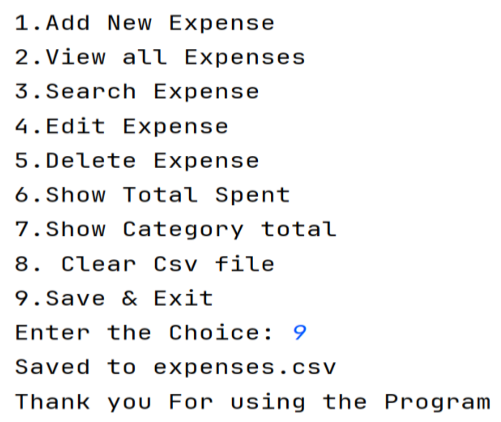
```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 7
Category   Total Spent
UTILITY    20
STUDY    12
FOOD    20
TRAVEL    20
```

7.8 Clear the CSV File



```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 8
File is clear now
```

7.9 Save And Exit



```
1.Add New Expense
2.View all Expenses
3.Search Expense
4.Edit Expense
5.Delete Expense
6.Show Total Spent
7.Show Category total
8. Clear Csv file
9.Save & Exit
Enter the Choice: 9
Saved to expenses.csv
Thank you For using the Program
```

8. Conclusion and Future Scope

8.1 Working on the project was very satisfying.

I started with a screen. I built a tool that I can use. The tool works. The project showed me why the Dictionaries give speed and why the File Handling is essential, for the software. The tool solved the problem of losing track of money. The modular design of the tool lets me add more to the tool later.

8.2 Future Scope

I am happy, with the way the system works. I have ideas for version 2.0. The system can use the ideas for version 2.0. The ideas, for version 2.0 can improve the system.

1. A GUI: I can use the command line. I want the GUI to be a GUI window, with buttons. The GUI built with Tkinter would look much better.
2. Visual Charts: Visual Charts uses matplotlib to draw a pie chart of my expenses. Visual Charts helps me see where my money goes.
3. Monthly Limits: I want a budget feature that lets the user set a budget, for example 5000 Rs. The budget feature warns the user when the user goes over the budget.