Smart Expense Tracker – Project Report

Title: Smart Expense Tracker

Submitted by: Kurra Sanjay

Register Number: 25MIM10079

Course: Introduction to Problem Solving & Programming

Academic Year: 2025–2026

Date: 28 November 2025

2. Introduction

Money management is important, but many students and beginners do not track their daily expenses. They only notice that money is finished at the end of the month, without knowing where it went. A small program that records expenses and shows simple summaries can already help a lot.

This project is a Python console-based Smart Expense Tracker. It allows the user to add expenses with category, see the list of all expenses, view category-wise totals, set a monthly budget, and check savings using monthly income. The project uses basic programming concepts like variables, lists, dictionaries, functions, loops, conditionals, and input/output.

3. Problem Statement

People usually do not know how much they spend on Food, Travel, Shopping, Bills, etc. Many times, they cross their monthly budget without any warning. They also do not calculate how much they actually save from their income.

Main problems:

- No simple way to record and see daily expenses quickly.
- No basic alerts when spending is near or above budget.
- No quick calculation of savings.

Objective:

Build a small Python program where a user can enter expenses, see total and category-wise spending, set a budget, and calculate savings.

4. Functional Requirements

- User can add a new expense with:
    - amount
    - category (Food/Travel/Shopping/Bills/Others)
    - optional note
- User can view all recorded expenses.

- Program calculates and shows total expenses.
- Program calculates and shows category-wise totals.
- Program identifies and shows the category with the highest spending.
- User can set and view a monthly budget.
- Program warns the user when total expenses:
    - cross 80% of the budget
    - cross 100% of the budget
- User can enter monthly income and see savings = income − total expenses.
- Program works using a simple numbered menu in the console.

5. Non-functional Requirements

- Easy to use with clear menu and messages.
- Runs on any computer with Python 3 installed.
- Does not crash on wrong inputs; shows error messages instead.
- Code is organized into separate functions for each main task.
- Solution is simple and suitable for an introductory programming course.

6. System Architecture

The program is a single Python file with:

- A **main loop** that:
    - shows the menu
    - takes the user's choice
    - calls the correct function
- **Input logic** to:
    - read expense details
    - read budget value
    - read income value
- **Processing logic** to:
    - calculate total expenses
    - calculate category totals
    - find highest spending category

    - compare total with budget
    - calculate savings
- **Output logic** to:
    - print all expenses
    - print total and category summaries
    - print budget and savings messages

All data (expenses, budget, income) is stored in memory using a list and simple variables while the program is running.

7. Design Diagrams (Explained in Text)

7.1 Use Case

- User adds expenses.
- User views all expenses.
- User views total and category summary.
- User sets budget and receives alerts.
- User enters income and views savings.
- User exits the program.

7.2 Workflow

1. Start program and show menu.
2. User chooses an option.
3. Program calls the related function:
    a. Add expense
    b. View expenses
    c. Show total
    d. Show category summary
    e. Set/view budget
    f. Enter income & view savings
4. After finishing, program returns to menu.
5. If user chooses “Exit”, program ends.

7.3 Sequence (Add Expense Example)

- User selects "Add Expense".
- Program asks for amount, category, note.
- Program validates inputs (amount positive, category not empty).
- Program saves expense to the list.
- Program checks budget (if set) and prints warning if needed.
- Program goes back to main menu.

7.4 Components

Logical components:

- Main menu and loop.
- Expense handling (add and list).
- Summary calculations (total, category, top category).
- Budget handling (set and check).
- Income and savings calculation.

7.5 ER (Logical Data Model)

- **Expense**: amount, category, note.
- **Budget**: monthly budget value.
- **Income**: monthly income value.

Total expenses and savings are calculated using these.

8. Design Decisions & Rationale

- **Console program:** Simple to run and matches course level.
- **Python:** Course language; easy to understand.
- **In-memory list:** No database needed; keeps project small and simple.
- **List of dictionaries for expenses:** Makes it easy to add, read, and summarize data.
- **Separate functions:** Each task is in its own function, so code is clear and easy to modify.

- **80% and 100% budget alerts:** Give the user a basic idea of when they are close to or over their limit.

9. Implementation Details

Main data:

- expenses – list of expense records.
- monthly_budget – number or None.
- monthly_income – number or None.

Each expense record looks like:

{"amount": 150.0, "category": "Food", "note": "Dinner"}

Main functions (names can match your code):

- add_expense()
- view_expenses()
- total_expenses()
- category_summary()
- set_budget()
- set_income_and_savings()

Main loop:

- While True:
    - print menu
    - get choice
    - call corresponding function
    - break when choice is Exit.

10. Screenshots / Results

#Menu

```
==== Expense Tracker ====
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice:
```

#Add expense

```
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 1

--- Add Expense ---
Enter amount: 500
Enter category (Food/Travel/Shopping/Bills/Others): bills
Short note/description (optional): car
Expense added successfully!
```

#View all expenses

```
==== Expense Tracker ====
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 2

--- All Expenses ---
1. Amount: 500.0 | Category: bills | Note: car
```

#Category Summary

```
==== Expense Tracker ====
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 4

--- Category-wise Summary ---
bills: 500.0

You are spending the most on: bills (500.0)
```

Budget Alert

```
==== Expense Tracker ====
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 5

--- Set / View Monthly Budget ---
Do you want to update the budget? (y/n): y
Enter new monthly budget amount: 500
Monthly budget set to: 500.0
```

```
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 1

--- Add Expense ---
Enter amount: 400
Enter category (Food/Travel/Shopping/Bills/Others): food
Short note/description (optional): none
Expense added successfully!
Warning: You have exceeded your monthly budget!
```

#Income and savings

```
==== Expense Tracker ====
1. Add Expense
2. View All Expenses
3. Total Expenses
4. Category Summary
5. Set / View Monthly Budget
6. Enter Income & View Savings
7. Exit
Enter choice: 6

--- Income & Savings ---
Enter this month's income: 7000
Total Expenses: 900.0
Savings: 6100.0
Good job! You are saving money this month.
```

11. Testing Approach

Testing was done manually by running the program and trying different inputs:

- Adding several valid expenses and checking if totals are correct.
- Trying invalid inputs:
    - letters for amount
    - negative amount
    - empty category
- Checking category summary with multiple categories.
- Setting a budget and adding expenses to:
    - stay below 80%
    - cross 80%
    - cross 100%
- Entering income and verifying:
    - case where income > expenses (savings positive)
    - case where income = expenses
    - case where income < expenses (overspending).

12. Challenges Faced

- Handling wrong inputs without stopping the program.
- Finding the highest spending category correctly.
- Designing a clean and simple menu.
- Remembering to declare global for budget and income variables inside functions.

These issues were solved by careful use of try/except, loops, and step-by-step testing.

13. Learnings & Key Takeaways

- How to use **functions, lists, and dictionaries** together in one project.
- Importance of **input validation** for stable programs.
- How to build a **menu-driven** console application.
- Turning raw numbers into **useful information**, such as top category and savings.
- Testing different cases to find and fix logic errors.

14. Future Enhancements

Possible future improvements:

- Saving and loading expenses from a file (CSV/JSON).
- Adding dates and filtering by month.
- Adding separate budget per category.
- Drawing simple graphs using Python plotting libraries.
- Adding a GUI using Tkinter.

15. References

- Course material for Introduction to Problem Solving & Programming.
- Official Python documentation: https://docs.python.org/3/
- Online tutorials for Python basics (functions, lists, dictionaries, input/output).