



Personal Finance Manager (Python CLI Application)

Python 3.10+

Interface CLI

Testing pytest

Docker Ready

📌 Project Overview

Personal Finance Manager is a complete, modular, and production-ready **Python command-line application** designed to help users **track expenses, manage budgets, generate reports, visualize spending trends, and maintain financial discipline**.

This project demonstrates:

- Object-Oriented Programming (OOP)
- File handling & persistence (CSV & JSON)
- Data validation & error handling
- Reporting & visualization using matplotlib
- Unit testing using pytest
- Dockerized deployment
- Clean, scalable architecture

🎯 Key Features



Expense Management

- Add new expenses with validation
- Edit existing expense records
- Delete expenses with confirmation
- Persistent storage using CSV files



Reports & Analytics

- Category-wise expense summary
- Monthly expense reports
- Total & average expense calculations

Visual Charts (Auto-Saved as PNG)

- Category-wise spending chart
- Monthly spending trend
- Budget vs Actual comparison

 Charts are saved automatically in:

reports/

Budget Management

- Set category-wise monthly budgets
- Delete or update budgets
- Real-time alerts when:
 - Budget exceeds
 - Budget reaches warning threshold

Smart Alerts

-  Budget exceeded alerts
-  Budget nearing limit alerts
- Displayed immediately after adding expenses

Search Functionality

- Search by date
- Search by category
- Search by amount range
- Keyword-based search

Testing

- Unit tests for:
 - Validation logic
 - File handling
 - Budget calculations
- Automated testing using pytest



Docker Support

- Fully containerized application
 - Reproducible environment
 - One-command execution
-

Project Architecture

```
Finance_Manager/
├── src/
│   ├── main.py          # Application entry point
│   ├── menu.py          # CLI menu system & flow control
│   ├── expense.py        # Expense class (OOP model)
│   ├── file_manager.py  # CSV backup, restore & persistence
│   ├── budget_manager.py# Budget logic & alerts
│   ├── reports.py       # Reports & chart generation
│   └── utils.py         # Validation & helper utilities
│
├── charts/
│   ├── category_spending.png
│   ├── monthly_spending.png
│   └── budget_vs_actual.png
│
├── data/
│   ├── expenses.csv      # Expense data
│   └── budgets.json      # Budget data
│
├── backups/
│   └── expenses_backup_20251218_141937.csv
│
├── reports/
│   └── report_2024-02.csv
│
└── tests/
    ├── test_utils.py
    ├── test_file_manager.py
    ├── test_budget_manager.py
    └── test_expense_manager.py
│
└── Dockerfile
└── requirements.txt
└── README.md
└── .dockerignore
```

Installation & Setup

◆ Prerequisites

- Python **3.10+**
 - pip
 - Git (optional)
 - Docker (optional)
-

◆ Local Setup

```
git clone <your-github-repo-url>
cd Finance_Manager
pip install -r requirements.txt
python main.py
```

◆ Docker Setup (Recommended)

```
docker build -t finance-manager .
docker run -it finance-manager
```

 Ensures consistent execution across all environments.

Application Usage

Main Menu

- ```
1. Add New Expense
2. View All Expenses
3. Edit Expense
4. Delete Expense
5. View Category-wise Summary
6. Budget Management
7. Generate Monthly Report
8. Search Expenses
9. Backup / Restore Data
10. Generate Spending Charts
0. Exit
```

## charts & visualizations

Charts are generated using **matplotlib** and saved automatically.

### Available Charts:

- Category Spending
- Monthly Spending Trend
- Budget vs Actual

### Location:

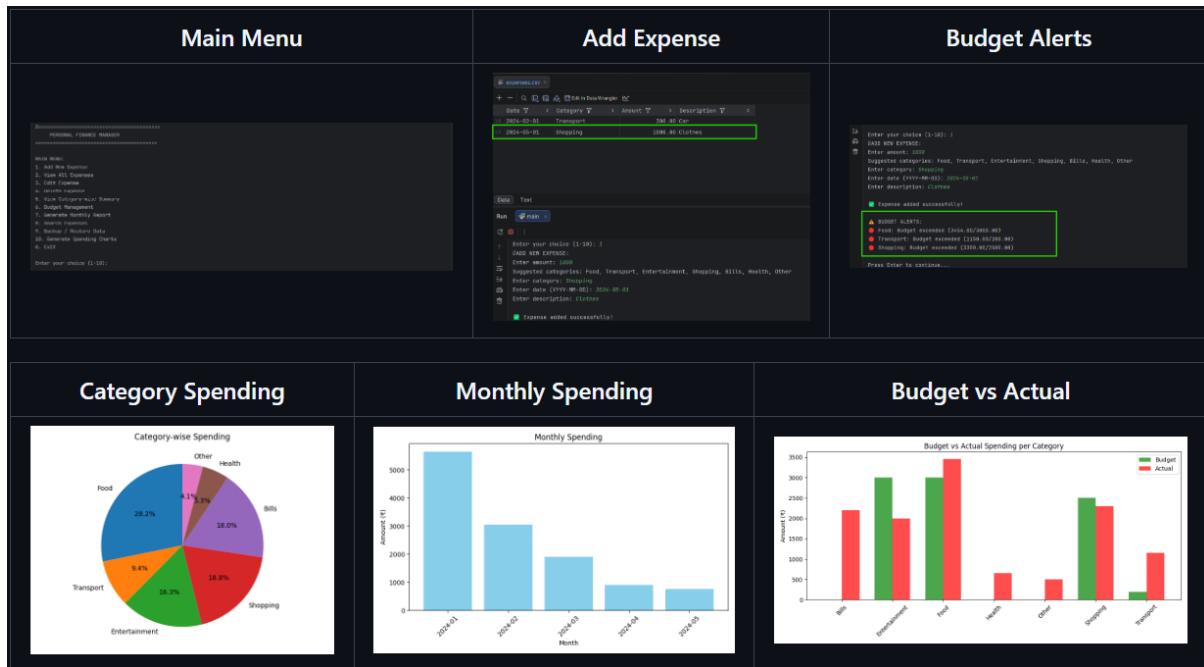
charts/

## screenshot suggestions (Add to GitHub)

Added to below folder:

screenshots/

## application screenshots



## Testing

Run all unit tests:

```
pytest -v
```

✓ Covers:

- Input validation
  - File persistence
  - Budget logic
- 

## Error Handling & Validation

- Invalid inputs handled gracefully
  - No crashes on user mistakes
  - Clear error messages
  - Safe file operations
- 

## Technical Highlights

- OOP design with clean separation of concerns
  - Modular and extensible codebase
  - CSV + JSON data persistence
  - Industry-standard testing approach
  - Dockerized for deployment
- 

## Future Enhancements

- GUI version (Tkinter / Streamlit)
- Cloud sync
- Multi-user support
- Database backend (SQLite/PostgreSQL)
- Data export (Excel / PDF)

---

## ★ Why This Project Matters

This project demonstrates **real-world Python engineering skills**, not just scripting:

- Architecture
- Testing
- Deployment
- Documentation
- Visualization

📌 This project was built as a complete end-to-end Python application to demonstrate real-world software engineering practices.