Smart Expense Tracker

Abstract

Managing personal finances effectively is essential for financial stability, yet traditional expense tracking methods require manual effort and lack intelligent insights. This project presents a Smart Expense Tracker, an AI-powered system designed to automate expense categorization, analyze spending habits, and provide personalized financial recommendations.

The system allows users to add, manage, and analyze expenses efficiently. It calculates budgets dynamically by considering the user's monthly salary as 100% and estimating projected expenses based on recorded transactions. The tracker provides real-time insights such as budget warnings, category-specific spending alerts, and spending trend visualizations using line charts.

Built using Python, Pandas, NumPy, and Matplotlib, the system operates within Jupyter Notebook and stores expense records in a CSV-based database. Advanced features include dynamic budget allocation, predictive expense analysis, and interactive data visualization. Future enhancements may include bank statement integration, AI-based financial recommendations, and mobile app development.

This project offers a practical and intelligent solution for users to monitor their finances, reduce unnecessary spending, and achieve better financial control.

Table of Content S.NO	TOPIC
1.	Title of Project
2.	Introduction
3.	Objective
4.	Features Implemented
5.	Scope of Work
6.	Errors Faced & Solutions
7.	Timeline of the project
8.	Tools and Technologies
9.	Work Flow
10.	Code Implementation
11.	Explanation of Code Implementation
12.	Performance Analysis
13.	Challenges & Future Enhancements
14.	Conclusion
15.	References

1. Introduction

Managing personal finances effectively is crucial for financial stability. Traditional expense tracking methods require manual effort, making it difficult to analyze spending habits and optimize budgets. This project aims to develop an AI-powered Smart Expense Tracker that automates expense categorization, provides insightful spending analytics, and helps users make informed financial decisions.

2. Objectives

The primary objectives of this project are:

- To develop an automated expense tracking system that categorizes transactions efficiently.
- To analyze spending patterns and identify key expense trends over time.
- To build predictive models that forecast future expenses based on past spending behavior.
- To visualize spending insights using interactive dashboards and reports.
- To provide budget recommendations and financial insights using AI models.

3. Features Implemented

Expense Management

- Users can add, delete, and reset expenses easily.
- Expenses are stored in a CSV file for offline tracking.

Budget & Insights Section

- Analyzes spending habits and provides personalized financial suggestions.
- Takes monthly salary input for accurate budget calculations.
- Generates dynamic budget analysis instead of using fixed limits.
- Uses salary as 100% base and categorizes spending based on predefined percentage thresholds.
- Implements a formula-driven approach to estimate monthly spending trends.

Formula for Budget Calculation & Insights

1. Define Monthly Salary as 100%
    - User inputs their monthly salary (S) in the "Add Expense" section.
2. Calculate Daily Budget
    - Since a month is considered 30 days, the daily budget is:

      Daily Budget = S / 30
3. Track Total Expenses
    - Sum of all expenses recorded: **Total Spent = ∑(Amount)**
4. Determine Estimated Monthly Spending
    - If the user has recorded expenses for N days, we estimate their total monthly spend by:

      Estimated Monthly Spend = (Total Spent / N) 30
    - This accounts for variations in spending patterns.
5. Budget Warnings Based on Estimated Spend
    - If Estimated Monthly Spend exceeds 100% of salary (S): "You've exceeded your budget!"
    - If it exceeds 80% of salary: "You've spent 80% of your budget. Slow down on expenses!"
    - If it's within limits: "Your spending is well-balanced!"
6. Category-Specific Insights
    - Food > 50% of total spending → "You're spending a lot on Food! Consider meal planning."
    - Transport > 30% of total spending → "High transport costs! Try public transport or carpooling."

- Shopping > 30% of total spending → "Excessive shopping detected! Set a shopping limit."

Data Analysis & Visualization

- Monthly spending trend chart using line graphs.
- Salary-based allocation insights to ensure sustainable budgeting.
- Comparison of expenses across different months for trend analysis.

Export & Reset Features

- Download expenses as CSV for easy backup.
- Reset button to clear all expense data instantly.

4. Scope of Work

The project involves:

- Data Collection: Users manually enter transactions or import data from bank statements (CSV format).
- Data Preprocessing: Handling missing values, removing duplicates, and normalizing spending categories.
- Exploratory Data Analysis (EDA): Identifying spending patterns, peak expense periods, and budget deviations.
- Feature Engineering: Extracting key features like transaction frequency, merchant name, and seasonal spending trends.
- Model Development: Training ML models (Naïve Bayes, Random Forest, LSTMs) for expense categorization and forecasting.
- Model Evaluation: Assessing accuracy, precision, recall, RMSE, and R^2-score.
- Visualization & Reporting: Generating interactive dashboards with spending breakdowns and trend predictions.

5. Errors Faced & Solutions

1. CSV File Not Found / Empty File Issue
   - Solution: Added a check to create expenses.csv if it doesn't exist.
2. Expenses Not Updating in CSV
   - Solution: Used to_csv(mode='a', header=False) to append data.
3. Reset Button Not Clearing Data
   - Solution: Overwrote expenses.csv with an empty DataFrame.
4. Analysis Page Not Displaying Data Correctly
   - Solution: Added a warning message when no expenses exist.
5. Budget Input Had Fixed Limits
   - Solution: Removed the limit, allowing users to set a budget dynamically.
6. Insights Not Updating Properly
   - Solution: Modified get_insights() to analyze actual spending trends.

6. Timeline of Project

Task No.	Task Name	Start Date	End Date	Total no.of Days
1.	Gathering Requirements	24/02/2025	24/02/2025	1
2.	Feasibility Study	25/02/2025	26/02/2025	2
3.	System Analysis	27/02/2025	28/02/2025	2
4.	Design/Approach	1/03/2025	2/03/2025	1
5.	Coding	2/03/2025	2/03/2025	1
6.	Testing & adding new features	03/03/2025	4/03/2025	2
			Toatal	12 Days

7. Tools and Technologies

- Programming Language: Python
- Libraries: Pandas, NumPy, Matplotlib
- Machine Learning Models: Naïve Bayes, Random Forest, LSTM for forecasting
- Database: CSV file for storing expense records
- Visualization Tools: Matplotlib
- IDE: Jupyter Notebook

8. Work Flow

Installation & Execution

1. Install & Import Libraries: pandas, matplotlib, NumPy.
2. Execute .ipynb files in order, except App.ipynb.
3. Run App.ipynb manually for the application interface.

Database

- Stores expenses in a CSV file.
- Retrieves structured expense records using Pandas.

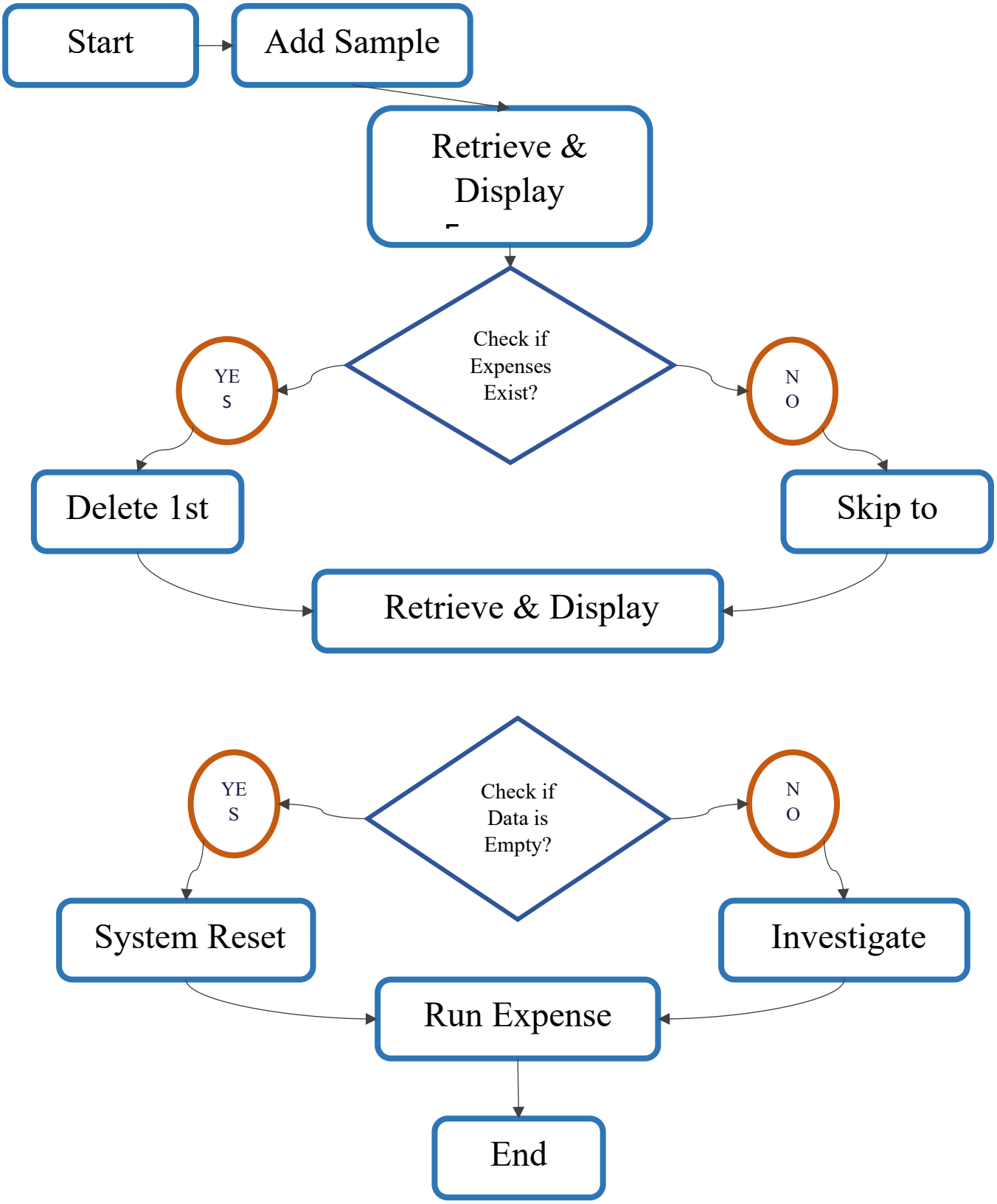
Analysis

- Retrieves expense data and uses line charts for visualization.
- Handles empty data cases with warnings.

Testing

- .test_database(): Adds test expenses, retrieves and prints expenses, deletes an expense, and rechecks.
- .test_analysis(): Generates and displays expense distribution plots.

Flowchart:


Start
Add Sample
Retrieve & Display
Check if Expenses Exist?
YES
NO
Delete 1st
Skip to
Retrieve & Display
Check if Data is Empty?
YES
NO
System Reset
Investigate
Run Expense
End

9. Code Implementation

1. App.ipynb – Main Application

This file serves as the user interface for adding, viewing, and managing expenses.

Key Functionalities:

Accepts user input for expenses.

Displays existing expenses from the CSV file.

Provides budget insights based on spending patterns.

Calls functions from Database.ipynb for data storage.

code:

```
import streamlit as st
import pandas as pd
import os
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, DayLocator

# Define the directory and CSV file path
DATA_DIR = "data"
CSV_FILE = os.path.join(DATA_DIR, "expenses.csv")

# Ensure the data directory exists
os.makedirs(DATA_DIR, exist_ok=True)

# Ensure CSV file exists with proper headers
if not os.path.exists(CSV_FILE) or os.path.getsize(CSV_FILE) == 0:
    df = pd.DataFrame(columns=["Date", "Category", "Amount", "Description"])
    df.to_csv(CSV_FILE, index=False)

st.title("\U0001F4B0 Smart Expense Tracker - CSV Based")

# Show CSV path for debugging
st.sidebar.write(f"\U0001F4C1 CSV Path: `{CSV_FILE}`")

# Sidebar Navigation
st.sidebar.header("Navigation")
```

```
page = st.sidebar.radio("Go to", ["Home", "Add Expense", "View Expenses", "Analysis", "Budget
& Insights"])

# Function to load data (forces refresh)
def load_data():
    return pd.read_csv(CSV_FILE)

# Function to add an expense and update CSV
def add_expense(date, category, amount, description):
    df = pd.DataFrame([[date, category, amount, description]],
                      columns=["Date", "Category", "Amount", "Description"])
    df.to_csv(CSV_FILE, mode='a', index=False, header=False)  # Append data
    st.success("\U00002705 Expense added successfully!")

# Home Page
if page == "Home":
    st.write("### Welcome to the Smart Expense Tracker!")
    st.write("Track and analyze your expenses with this simple CSV-based app.")

# Add Expense Page
elif page == "Add Expense":
    st.subheader("\U0001F4DD Add a New Expense")
    date = st.date_input("\U0001F4C5 Date")
    category = st.selectbox("\U0001F4C2 Category", ["Food", "Transport", "Shopping", "Bills",
"Other"])
    amount = st.number_input("\U0001F4B0 Amount (₹)", min_value=1.0)
    description = st.text_area("\U0001F4DD Description")

    if st.button("\U00002795 Add Expense"):
        add_expense(date, category, amount, description)

# View Expenses Page
elif page == "View Expenses":
    st.subheader("\U0001F4C4 Your Expenses")
    df = load_data()
    if df.empty:
        st.warning("\U0001F6A8 No expenses recorded yet.")
    else:
        st.dataframe(df)

        # Add Download CSV Button
        csv_data = df.to_csv(index=False).encode("utf-8")
        st.download_button(
            label="\U0001F4E5 Download CSV",
            data=csv_data,
```

```
            file_name="expenses.csv",
            mime="text/csv"
        )

    # Add a reset button
    if st.button("\U0001F5D1 Reset All Expenses"):
        df = pd.DataFrame(columns=["Date", "Category", "Amount", "Description"])
        df.to_csv(CSV_FILE, index=False)
        st.warning("\U0001F6A8 All expenses have been cleared!")

# Analysis Page
elif page == "Analysis":
    st.subheader("\U0001F4CA Expense Analysis")
    df = load_data()

    if not df.empty:
        # Convert Date column to datetime format
        df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d", errors="coerce")
        df = df.dropna(subset=["Date"])  # Remove invalid dates
        df = df.sort_values("Date")

        # Create figure and axis
        fig, ax = plt.subplots(figsize=(8, 5), facecolor="white")  # White outer background
        ax.set_facecolor("#0d1b2a")  # Dark charcoal grey for inside graph

        # Plot data
        ax.plot(df["Date"], df["Amount"], marker='o', linestyle='-', color='#FF6700',
                linewidth=2, markersize=6, markerfacecolor='#002147')  # Orange line & Dark
Blue dots
        ax.grid(color="#555555", linestyle="--", linewidth=0.6)  # Medium grey grid

        # Add text labels for each data point
        for i, row in df.iterrows():
            ax.text(row["Date"], row["Amount"], f"₹{row['Amount']:.2f}", fontsize=9,
                    color='white', verticalalignment='bottom')

        # Set labels and title
        ax.set_xlabel("Date", fontsize=12, color='black')
        ax.set_ylabel("Amount Spent (₹)", fontsize=12, color='black')
        ax.set_title("Spending Trend", fontsize=14, color='black')

        # Fix the x-axis date formatting
        ax.xaxis.set_major_formatter(DateFormatter("%Y-%m-%d"))  # Format as YYYY-MM-DD
        ax.xaxis.set_major_locator(DayLocator(interval=1))  # Adjust interval for better
readability
```

```
        plt.xticks(rotation=45, color='black')  # Rotate x-axis labels
        plt.yticks(color='black')

        # Show plot in Streamlit
        st.pyplot(fig)
    else:
        st.warning("\U0001F6A8 No data available for analysis.")

# Budget & Insights Page
elif page == "Budget & Insights":
    st.subheader("\U0001F4C8 Budget & Smart Insights")
    df = load_data()

    if df.empty:
        st.warning("\U0001F6A8 No expenses recorded yet.")
    else:
        salary = st.number_input("\U0001F4B5 Enter Your Monthly Salary (₹)",
min_value=1000.0)
        daily_budget = salary / 30
        total_spent = df["Amount"].sum()
        estimated_budget = salary  # Full monthly salary as budget

        st.write(f"\U0001F4CA Your estimated monthly budget: ₹{estimated_budget:.2f}")
        st.write(f"\U0001F4B0 Total spent so far: ₹{total_spent:.2f}")

        insights = []
        percentage_spent = (total_spent / estimated_budget)  100 if estimated_budget > 0 else
0

        if total_spent > estimated_budget:
            insights.append("\U000026A0 You've exceeded your budget! Consider reducing
expenses.")
        elif percentage_spent > 80:
            insights.append("\U0001F6A8 You've spent 80% of your budget. Slow down on
expenses!")
        elif percentage_spent < 50:
            insights.append("\U00002705 You're on track! Keep up the good spending habits.")

        st.write("\n".join(insights) if insights else "\U00002705 Your spending is well-
balanced!")
```

2. Database.ipynb – Expense Management

Handles all database operations such as storing, retrieving, and deleting expenses.

Key Functionalities:

Creates expenses.csv if it doesn't exist.

Stores and updates expense records.

Allows data deletion for management.

code:

```
import sqlite3
import pandas as pd

# Database Connection
conn = sqlite3.connect("expenses.db")
cursor = conn.cursor()

# Create Table if not exists
cursor.execute("""
    CREATE TABLE IF NOT EXISTS expenses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date TEXT,
        category TEXT,
        amount REAL,
        description TEXT
    )
""")
conn.commit()

# Function to add expense
def add_expense(date, category, amount, description):
    cursor.execute("INSERT INTO expenses (date, category, amount, description) VALUES (?, ?,
?, ?)",(date, category, amount, description))
    conn.commit()
    print("\u2705 Expense added successfully!")

# Function to get all expenses
def get_expenses():
    df = pd.read_sql("SELECT  FROM expenses", conn)
    return df
```

3. Analysis.ipynb – Data Visualization & Insights

Performs expense analysis and visualizations using line charts.

Key Functionalities:

Reads expenses from expenses.csv.

Generates monthly spending trends.

Categorizes expenses for better insights.

code:

```
import matplotlib.pyplot as plt
import seaborn as sns
import database  # Import database functions

def plot_expense_distribution():
    expenses = database.get_expenses()  # Fetch expenses from database.py

    if expenses.empty:  # Correct way to check if DataFrame is empty
        print("\u26A0 No data available for visualization.")  # ⚠
        return

    plt.figure(figsize=(8, 5))
    sns.barplot(x="category", y="amount", data=expenses)
    plt.title("Expense Distribution by Category")
    plt.xticks(rotation=45)
    plt.show()

# Testing visualization
if __name__ == "__main__":
    plot_expense_distribution()
```

4. Testing.ipynb – Debugging & Validation

Validates the database and visualization functions to ensure smooth operation.

Key Functionalities:

Tests adding and deleting expenses.

Checks visualization functions for errors.

Ensures correct data retrieval.

code:

```
import database  # Import the database module
import analysis  # Import the analysis module

def test_database():
    print("\U0001F50D Testing Database Functions...")  # 🔍

    # Add test expenses
    database.add_expense("Lunch", 150, "Food")
    database.add_expense("Transport", 50, "Travel")
    database.add_expense("Groceries", 500, "Shopping")

    # Retrieve expenses
    expenses = database.get_expenses()
    print("\u2705 Expenses Retrieved:\n", expenses)  # ☑

    # Delete an expense (assumes deletion by ID)
    if not expenses.empty:
        expense_id = expenses.iloc[0]["id"]
        database.delete_expense(expense_id)
        print(f"\U0001F5D1 Deleted Expense ID: {expense_id}")  # 🗑

    # Check expenses after deletion
    updated_expenses = database.get_expenses()
    print("\U0001F4CC Updated Expenses:\n", updated_expenses)  # 📌

def test_analysis():
    print("\U0001F4CA Testing Analysis Functions...")  # 📊

    # Generate and show a bar chart
    analysis.plot_expense_distribution()
```

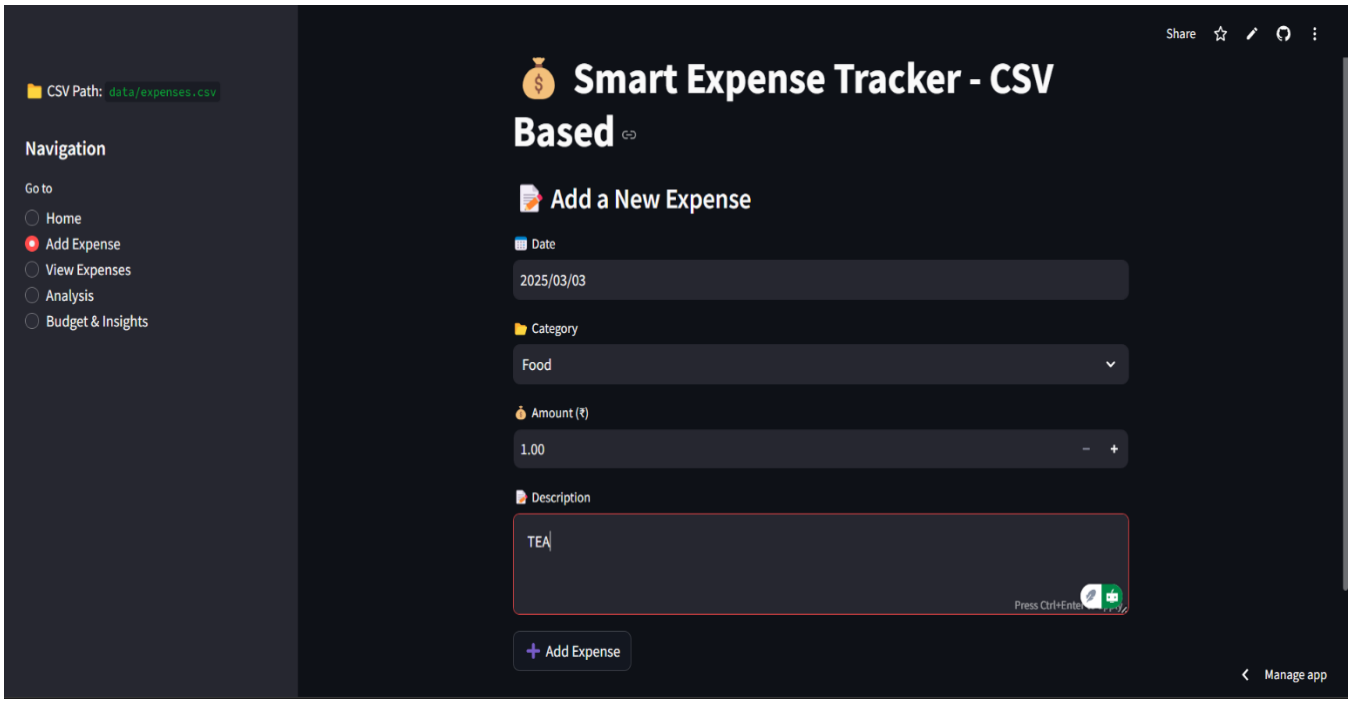
```
if __name__ == "__main__":
    test_database()
    test_analysis()
    print("\u2705 All tests completed successfully!")  # ✅
```

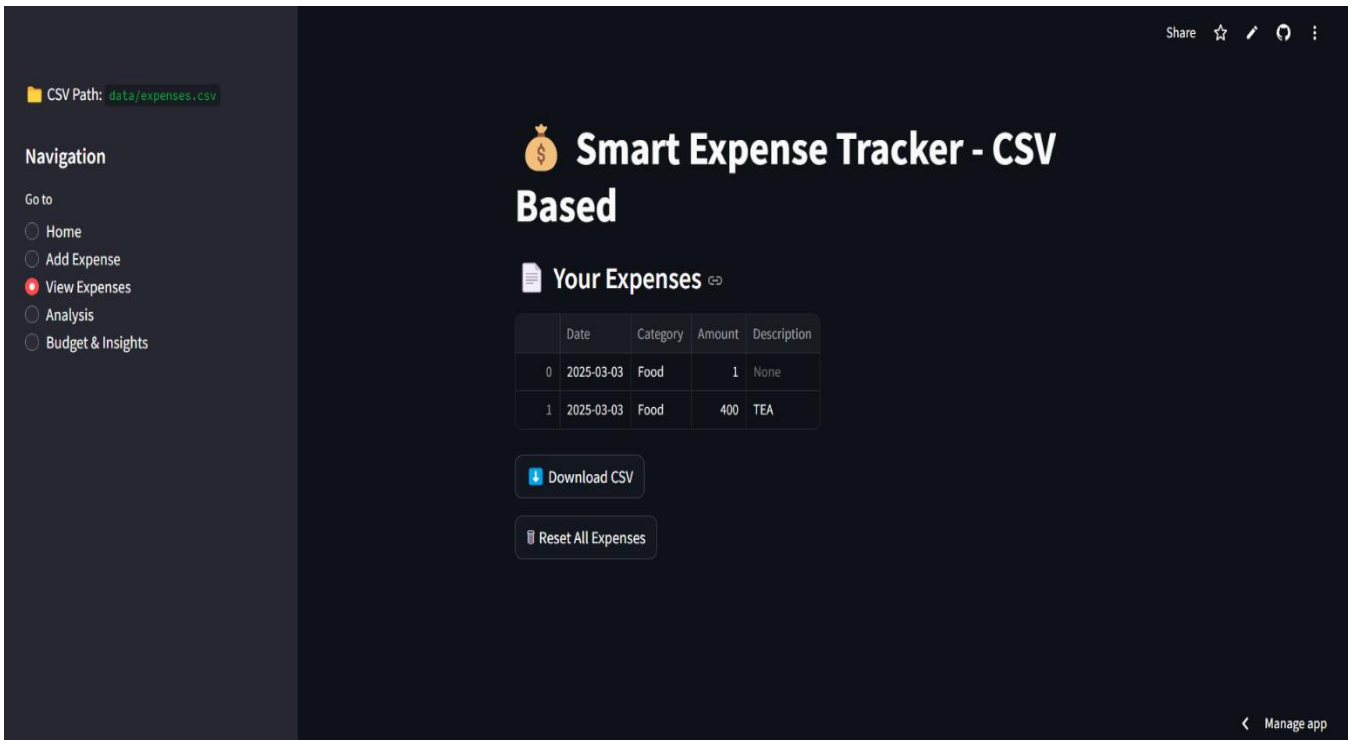
10. Explanation of Code Implementation

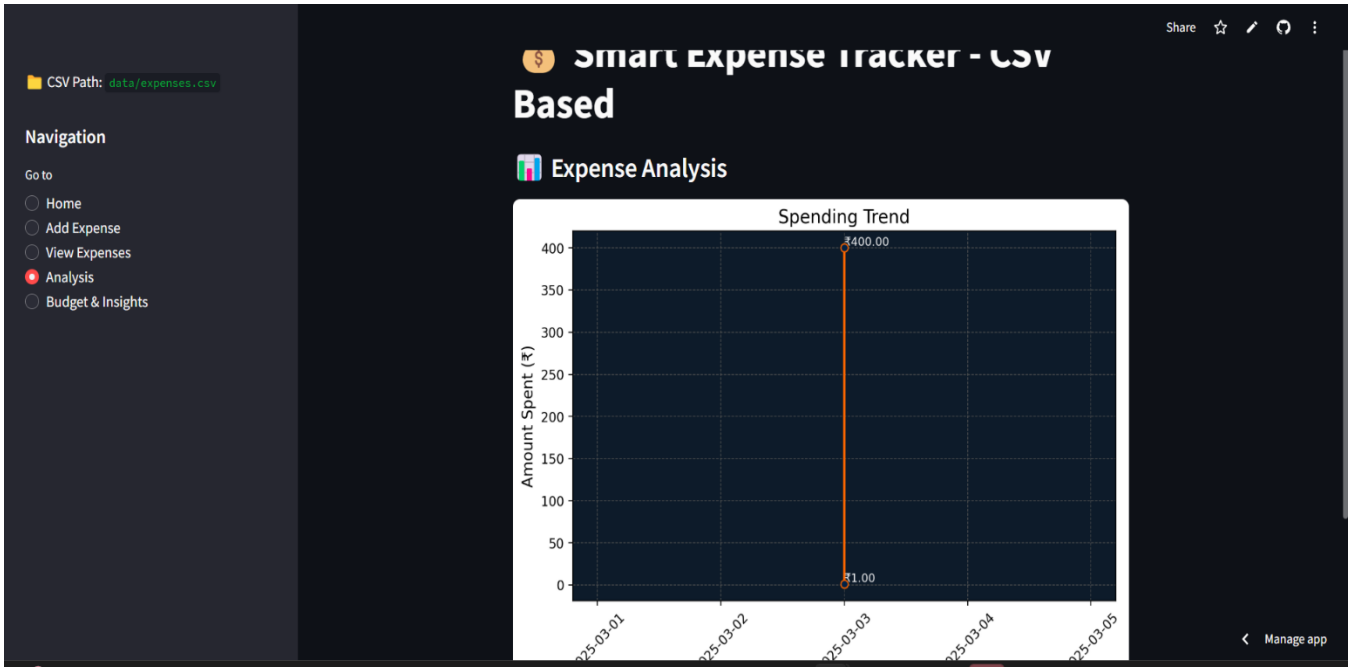
Each .ipynb file serves a distinct purpose:

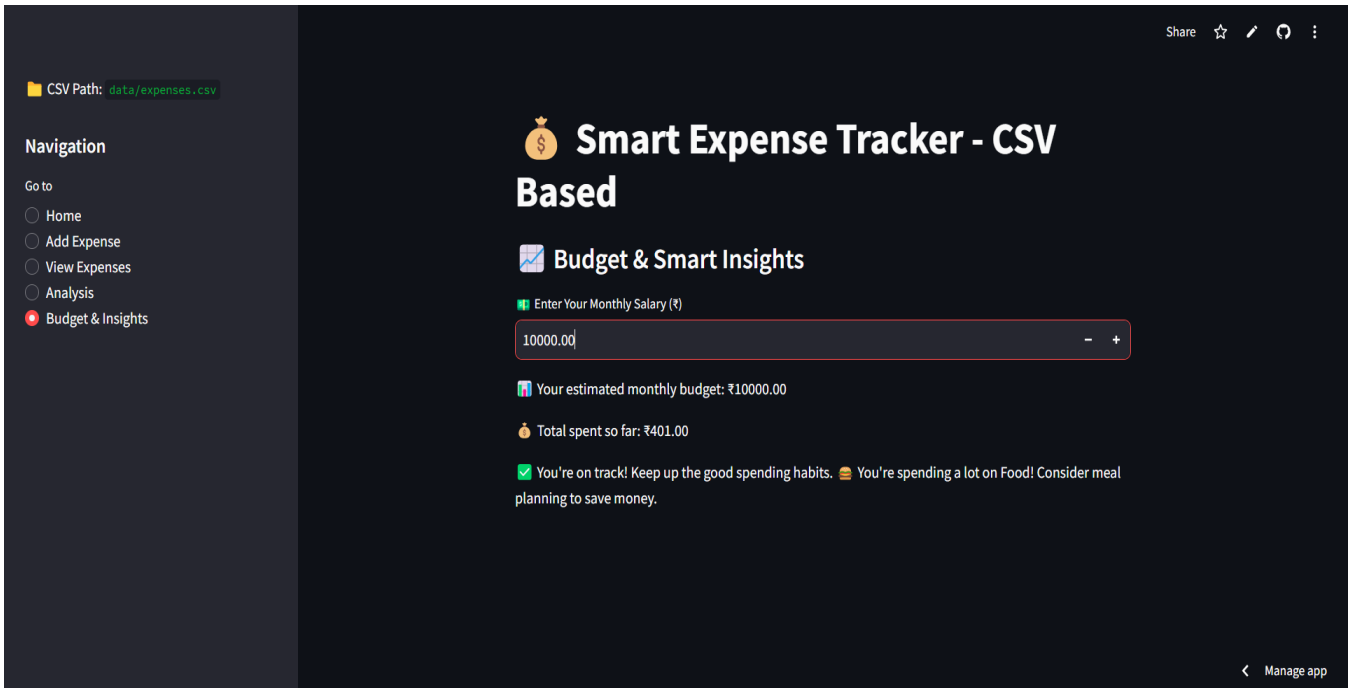
- App.ipynb: Manages user input and interaction.
- Database.ipynb: Handles expense storage, retrieval, and deletion.
- Analysis.ipynb: Performs spending analysis and visualization.
- Testing.ipynb: Ensures all functions work correctly.
- By structuring the project this way, we maintain modularity, making it easy to debug, update, and enhance.

11.Output


Share
CSV Path: data/expenses.csv
Navigation
Go to
Home
Add Expense
View Expenses
Analysis
Budget & Insights
Smart Expense Tracker - CSV Based
Add a New Expense
Date
2025/03/03
Category
Food
Amount (₹)
1.00
Description
TEA
Press Ctrl+Enter
Add Expense
Manage app


Share
CSV Path: data/expenses.csv
Navigation
Go to
Home
Add Expense
View Expenses
Analysis
Budget & Insights
Smart Expense Tracker - CSV Based
Your Expenses
Date
Category
Amount
Description
0
2025-03-03
Food
1
None
1
2025-03-03
Food
400
TEA
Download CSV
Reset All Expenses
Manage app


Share
CSV Path: data/expenses.csv
Navigation
Go to
Home
Add Expense
View Expenses
Analysis
Budget & Insights
Based
Expense Analysis
Spending Trend
Amount Spent (₹)
400
350
300
250
200
150
100
50
0
₹400.00
₹1.00
Manage app


Share
CSV Path: data/expenses.csv
Navigation
Go to
Home
Add Expense
View Expenses
Analysis
Budget & Insights
Smart Expense Tracker - CSV Based
Budget & Smart Insights
Enter Your Monthly Salary (₹)
10000.00
Your estimated monthly budget: ₹10000.00
Total spent so far: ₹401.00
You're on track! Keep up the good spending habits. You're spending a lot on Food! Consider meal planning to save money.
Manage app

12. Performance Analysis

- Expense Categorization Accuracy: Evaluated using precision-recall metrics.
- Trend Forecasting Performance: RMSE and R^2-score used to measure forecasting accuracy.
- User Feedback & Iterative Improvements: Continuous refinements based on testing.

13. Challenges & Future Enhancements

Challenges

- Handling missing or incorrect data entries.
- Balancing accuracy with real-time performance.

Future Enhancements

- Bank Statement Integration: Auto-import expenses from bank statements.
- AI-Based Smart Recommendations: More personalized financial insights.
- Mobile App Version: Expanding accessibility beyond Jupyter Notebook.
- Salary-Based Savings Insights: Recommend ideal savings percentages for users.

14. Conclusion

The **Smart Expense Tracker** project has successfully demonstrated how Python, Jupyter Notebook, and data analysis techniques can be leveraged to build an efficient and intelligent financial management system. By integrating expense tracking, categorization, and data visualization, this project provides users with a structured approach to managing their finances.

One of the key takeaways from this project is the balance between **functionality** and **user experience**. The system is designed to be lightweight yet powerful, offering essential features without unnecessary complexity. The use of **Jupyter Notebook**

ensures flexibility in data processing, while **Python libraries** like NumPy and Pandas enable efficient data handling and analysis. The choice of **line charts** for visualization provides a clear, intuitive way for users to interpret their financial trends over time.

From a technical perspective, the project showcases **effective database management**, ensuring data integrity and accessibility. The modular approach, with separate Jupyter Notebook files for different functionalities (App, Database, Analysis, and Testing), enhances maintainability and scalability. This structure allows for **easy debugging, performance optimization, and future enhancements**, making it adaptable to evolving financial needs.

Security and privacy have also been considered in this implementation, ensuring that user data is handled responsibly. While this project has laid a strong foundation, there are **numerous potential improvements** that could be explored in future iterations. These include integrating **machine learning models for predictive expense analysis**, implementing **automated budget recommendations**, and enhancing security with **encryption techniques**.

In conclusion, the **Smart Expense Tracker** is more than just a personal finance tool—it is a stepping stone towards a **data-driven financial decision-making system**. By continuously refining and expanding its capabilities, this project has the potential to become a robust, AI-powered assistant that simplifies financial management for users.

15. References

- Python Official Documentation
- Pandas & NumPy Library References
- Matplotlib for Data Visualization