A PBL – 1 Synopsis on

Personal Finance Tracker

Submitted to Manipal University Jaipur

Towards the partial fulfilment for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In Computers Science and Engineering

2025-2026

By
Piyush Prakash Sharma
2427030222

MANIPAL UNIVERSITY
JAIPUR

Under the Guidance of

Ms.Harshika Mathur

_____

Sign of Supervisor

Department of Computer Science and Engineering
School of Computer Science and Engineering
Manipal University Jaipur
Jaipur, Rajasthan

I. Introduction to Problem (What, Why, How)

The "What": The Problem of Personal Financial Management

When a person tries to manage their finances with the help of some web or mobile app they are met with possibilities of data leaks and enormous paywalls and NO normal Person wants to go through the hassle of transforming raw, unstructured financial transaction data provided by the banks usually in form of CSV files into useful, categorized ,,high-level financial insights .Information which would provide invaluable information on their spending habits ,financial spending and flow of their earnings etc.

The "Why": The Failure of Manual Methods

A significant gap exists in the current software market.

1. **Commercial SaaS (e.g., YNAB, Quicken):** These tools are easy to use but require recurring subscription fees and significant data privacy compromises (linking live bank accounts to the cloud).
2. **Open-Source (e.g., Firefly III):** These solutions are free and private but are built for technical users, requiring skills in server self-hosting, database management, and scripting.

The "How": Fragmentation of the Modern PFM Market

The proposed solution is the **Finance Tracker**, an **Application based** tool. It is a lightweight, offline Python application that operates *only* on a local, user-provided CSV file. It automates the most laborious tasks (data cleansing, categorization, and report generation) and outputs the final analysis into a universally familiar Microsoft Excel file.

II. Literature Survey

A survey of 10 existing solutions reveals the specific trade-offs in the current market.

A. Cloud-Based Subscription Platforms

- **YNAB (You Need A Budget)**
  - **Pros:** Excellent zero-based budgeting system.
  - **Cons:** High subscription cost ($109/yr) and steep learning curve.
- **Monarch Money**
  - **Pros:** Comprehensive dashboard for net worth and investments.
  - **Cons:** Expensive ($99/yr) and fully cloud-dependent.
- **Quicken Simplifi**
  - **Pros:** User-friendly interface and good reporting.
  - **Cons:** Requires recurring subscription; less powerful than desktop predecessors.
- **Copilot Money**
  - **Pros:** Uses Machine Learning for auto-categorization; excellent UI.
  - **Cons:** Exclusive to Apple (iOS/Mac) ecosystem and US banks.
- **Rocket Money**
  - **Pros:** Specializes in identifying and cancelling subscriptions.
  - **Cons:** Privacy concerns regarding data access permissions.

B. Freemium Cloud Services

- **Empower Personal Dashboard**
  - **Pros:** Best-in-class for investment and net worth tracking (free).
  - **Cons:** Budgeting tools are rudimentary; primarily a lead-gen tool for advisors.
- **Wallet by BudgetBakers**
  - **Pros:** Extensive international bank support.
  - **Cons:** Bank synchronization is often unstable; free version is heavily restricted.

C. Spreadsheet & Local Alternatives

- **Tiller Money**
  - **Pros:** Pipes data directly into Google Sheets for maximum customization.
  - **Cons:** Requires high spreadsheet proficiency; paid subscription.
- **GnuCash**
  - **Pros:** Free, open-source, local-first privacy, professional double-entry accounting.
  - **Cons:** Extremely complex UI; strictly manual data entry.
- **HomeBank**
  - **Pros:** Free, lightweight, and simpler than GnuCash.
  - **Cons:** Lacks automation; relies entirely on manual file imports.

III. Comparative Study

Software	Hosting	Cost	Data Import	Key Feature	Key Weakness
YNAB	Cloud	High ($109/yr)	API Sync	Zero-Based Budgeting	Expensive
Monarch	Cloud	High ($99/yr)	API Sync	All-in-One Dashboard	Cloud Privacy Risk
Simplifi	Cloud	Medium	API Sync	Reporting	Subscription Model
Empower	Cloud	Free	API Sync	Wealth Tracking	Weak Budgeting
Copilot	Cloud	High	API Sync	ML Categorization	Mac/iOS Only
Rocket	Cloud	Freemium	API Sync	Sub. Management	Privacy Concerns
Tiller	Cloud	Medium	API Sync	Custom Sheets	High Difficulty
GnuCash	Local	Free	Manual	Double-Entry Books	Steep Learning Curve
HomeBank	Local	Free	Manual	Lightweight	No Automation
LocalFinTracker	**Local**	**Free**	**CSV Import**	**Private Automation**	**Static Categories**

IV. Problem Statement

To track their expenses, Users currently face only 2 choices:

Sacrifice privacy for the convenience of cloud-based automation.

OR

Accept the tedious process of manual entry for the security of offline software.

There is no streamlined, free tool that automates financial analysis from bank statements while guaranteeing 100% local data sovereignty.

V. Objective

1. **Develop an Automated Program/Software:** Create a Python application using pandas to clean and categorize raw bank CSV data into meaningful insights.

2. **Generate Visual Reports:** Utilize openpyxl to auto-generate Excel dashboards with summary tables and pie charts for immediate user feedback.

3. **Ensure Data Sovereignty:** Validate a local-execution model where all processing occurs on the client machine, eliminating third-party data risks.

VI. Planning of Work / Proposed Solution

Current Implementation (Methodology)

The current system is a functional Python script executing a linear data pipeline:

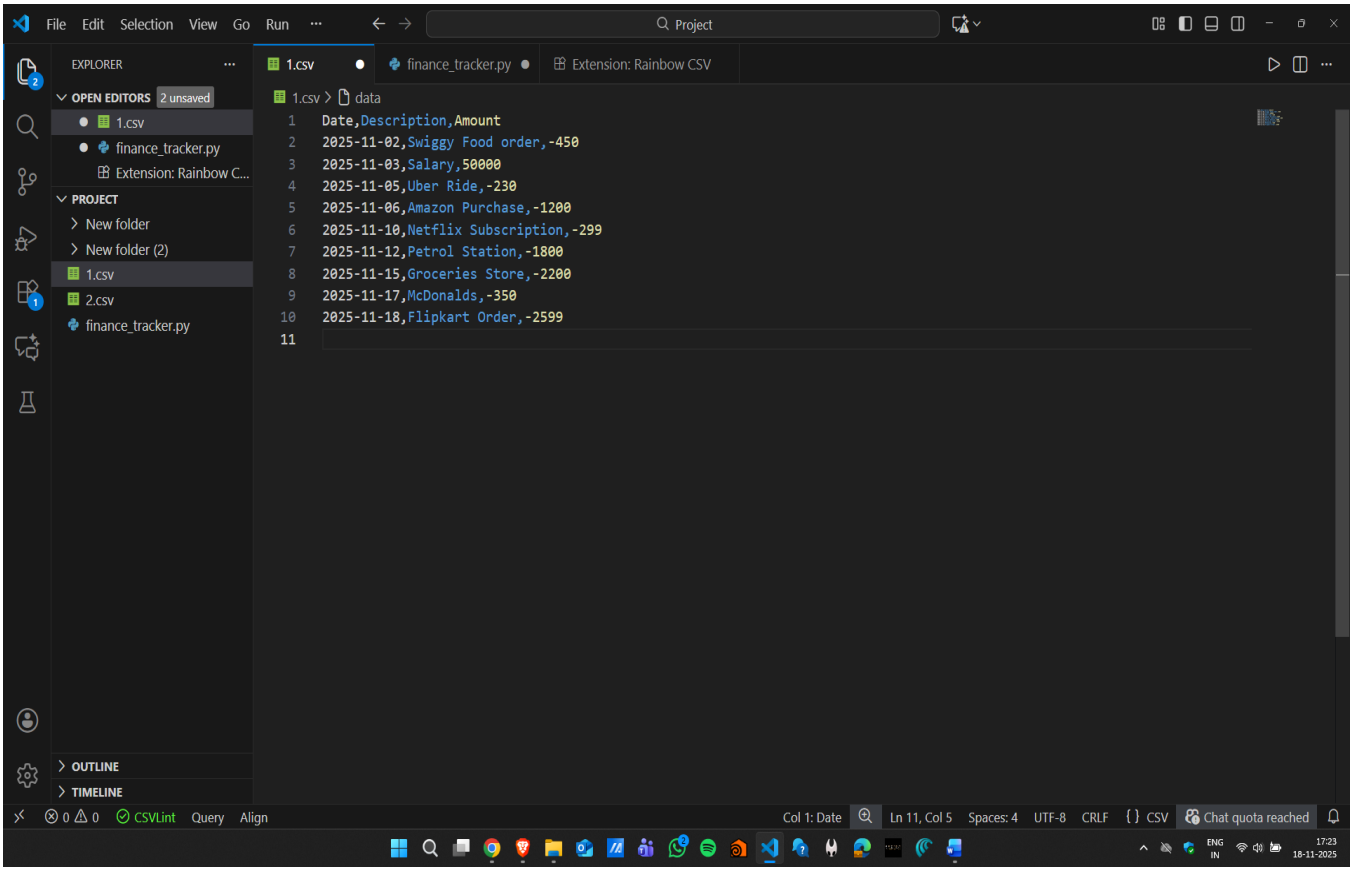
1. **Input :** The script accepts a raw CSV bank statement.
2. **Cleaning:** The clean_amount_column function sanitizes currency strings (removing '₹', commas) and standardizes data types.
3. **Categorization:** A keyword-matching algorithm maps transaction descriptions to categories (e.g., "Swiggy" → "Food") using a predefined dictionary.
4. **Reporting:** The openpyxl library generates a structured .xlsx file containing a transaction log, a category summary table, and an embedded Pie Chart.

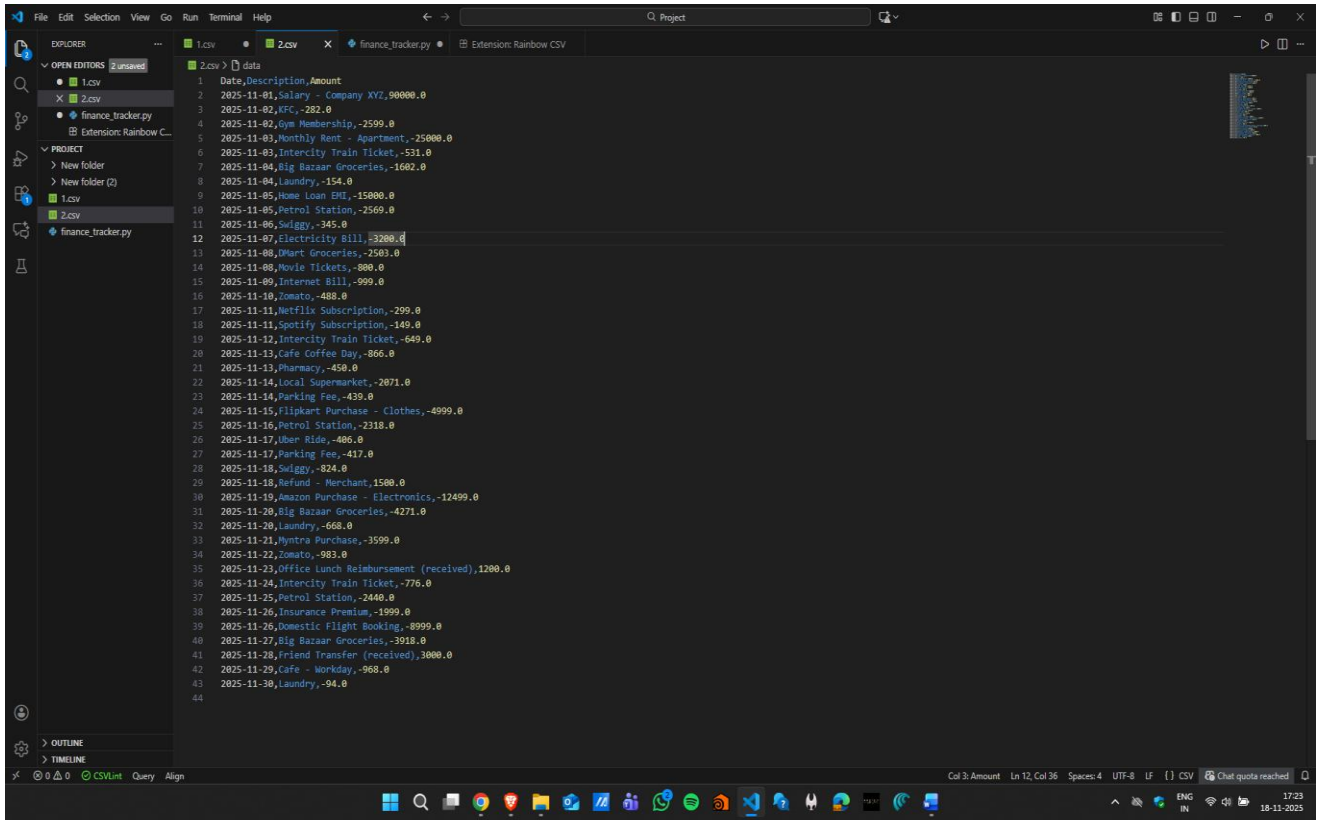
Future Enhancements

To expand beyond the current prototype, the project will integrate API and ML capabilities:

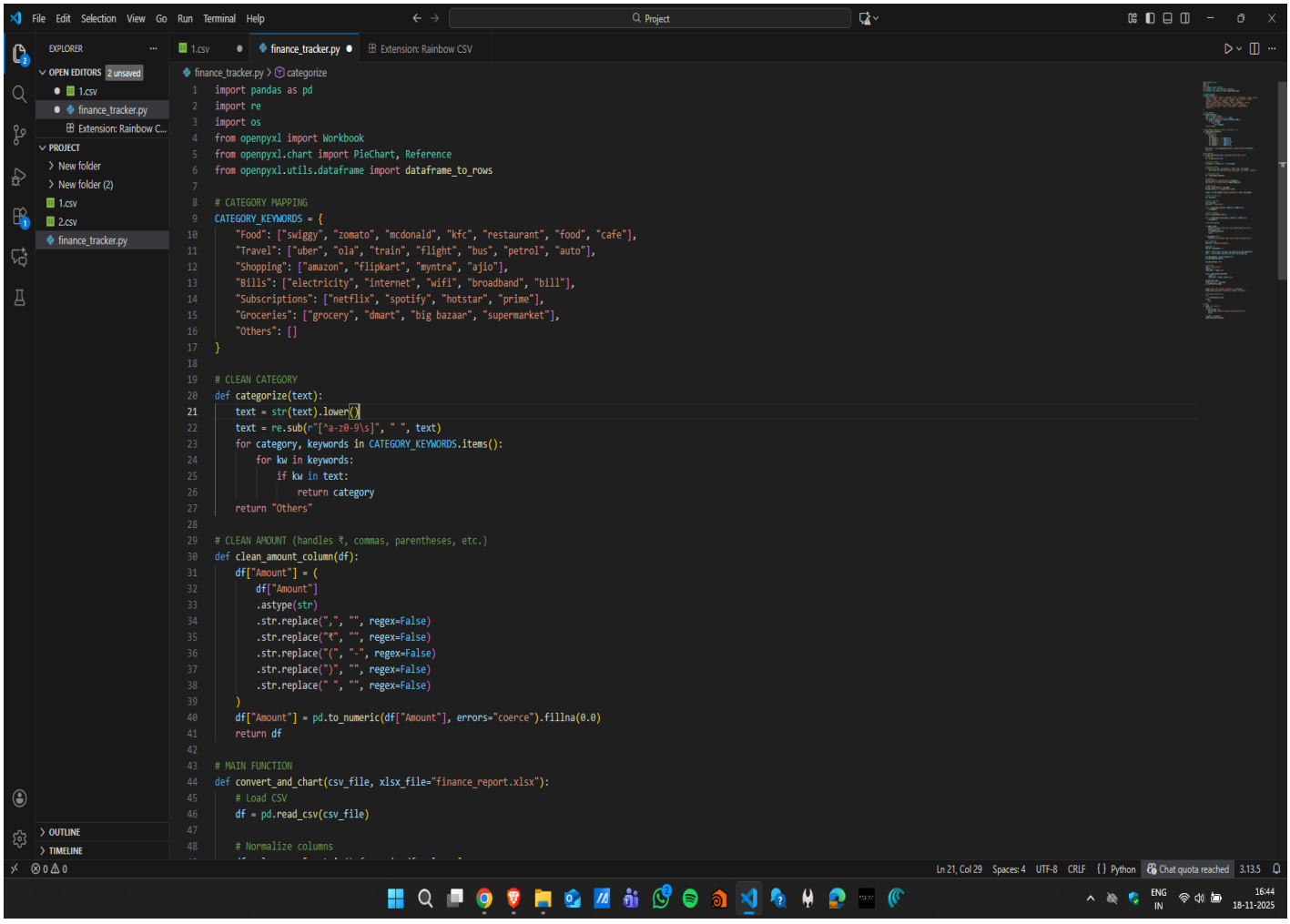
- **Phase 1: API Integration (Plaid):** Replace manual CSV downloads with the Plaid API. The application will fetch transactions directly using a locally stored access token, maintaining the "local-first" architecture while adding sync convenience.
- **Phase 2: Intelligent ML Categorization:** Replace static keywords with a scikit-learn Naive Bayes classifier. The system will learn from user corrections, creating a personalized model that improves over time without sharing data.
- **Phase 3: Predictive Forecasting:** Implement time-series forecasting to predict future spending trends based on historical data, alerting users to potential budget overruns.

VII. Screenshots

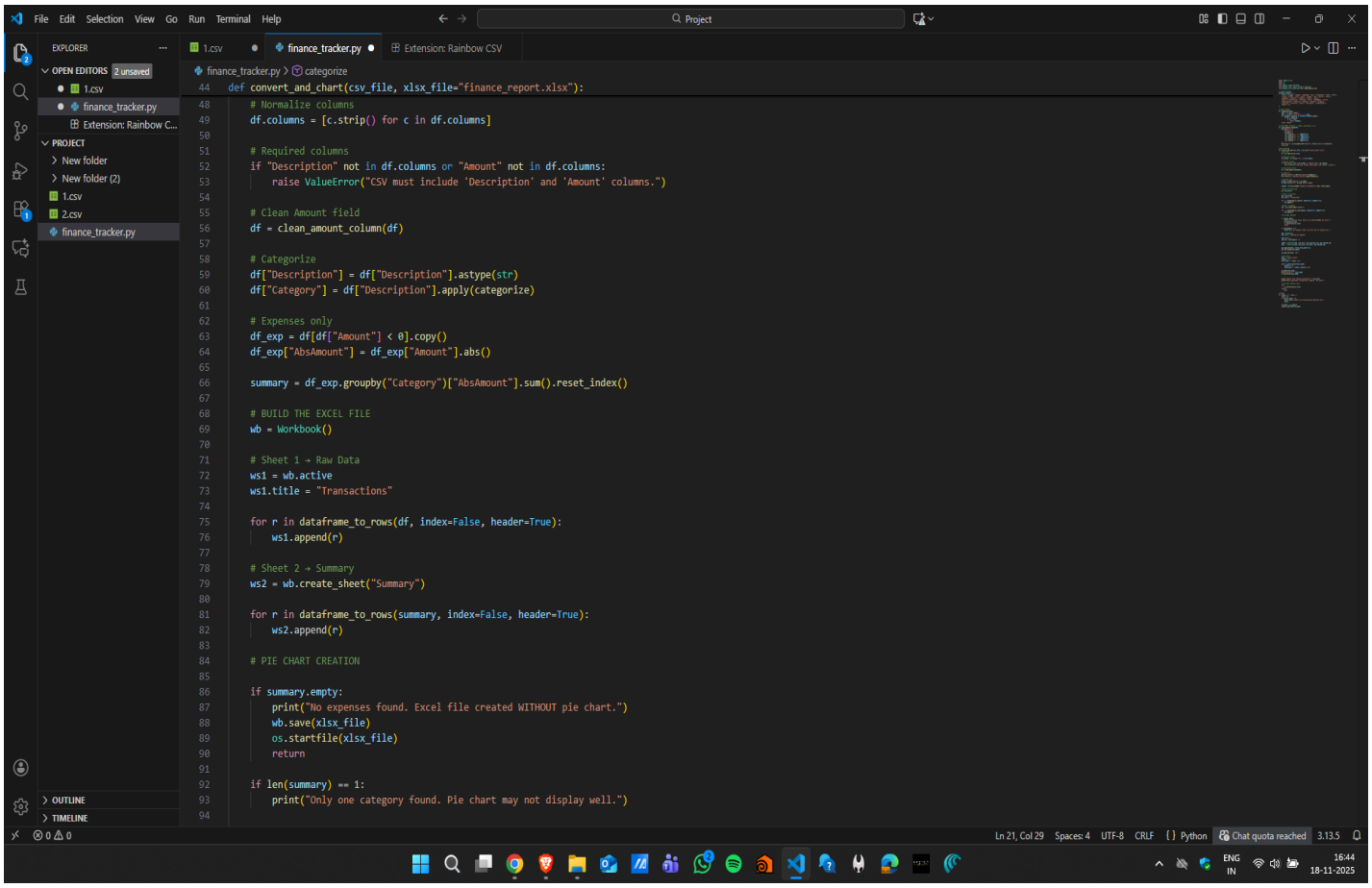




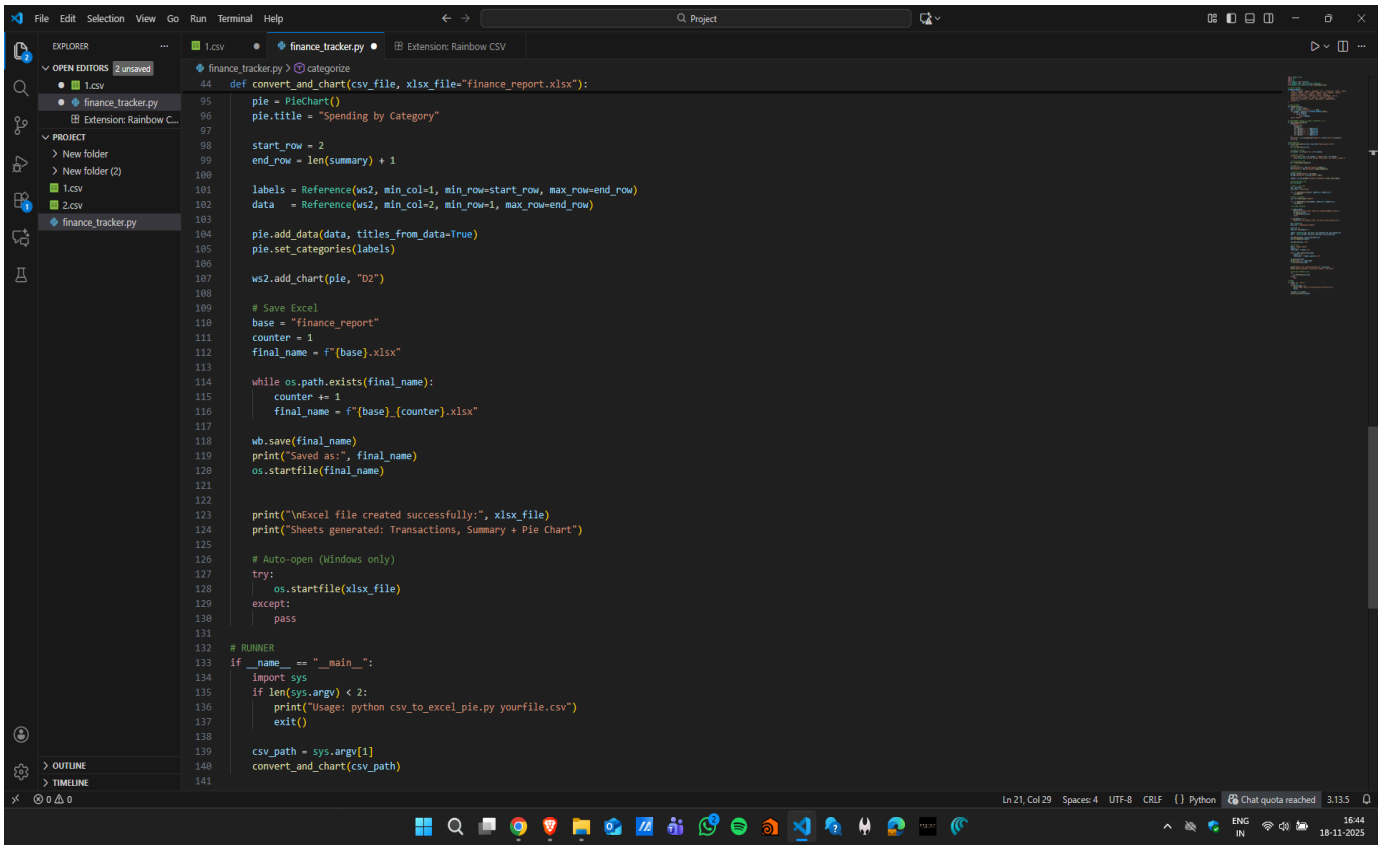
```
Date,Description,Amount
2025-11-01,Salary - Company XYZ,90000.0
2025-11-02,KFC,-282.0
2025-11-02,Gym Membership,-2599.0
2025-11-03,Monthly Rent - Apartment,-25000.0
2025-11-03,Intercity Train Ticket,-531.0
2025-11-04,Big Bazaar Groceries,-1602.0
2025-11-04,Laundry,-154.0
2025-11-05,Home Loan EMI,-15000.0
2025-11-05,Petrol Station,-2569.0
2025-11-06,Swiggy,-345.0
2025-11-07,Electricity Bill,-3200.0
2025-11-08,DMart Groceries,-2503.0
2025-11-08,Movie Tickets,-800.0
2025-11-09,Internet Bill,-999.0
2025-11-10,Zomato,-488.0
2025-11-11,Netflix Subscription,-299.0
2025-11-11,Spotify Subscription,-149.0
2025-11-12,Intercity Train Ticket,-649.0
2025-11-13,Cafe Coffee Day,-866.0
2025-11-13,Pharmacy,-450.0
2025-11-14,Local Supermarket,-2071.0
2025-11-14,Parking Fee,-439.0
2025-11-15,Flipkart Purchase - Clothes,-4999.0
2025-11-16,Petrol Station,-2318.0
2025-11-17,Uber Ride,-406.0
2025-11-17,Parking Fee,-417.0
2025-11-18,Swiggy,-824.0
2025-11-18,Refund - Merchant,1500.0
2025-11-19,Amazon Purchase - Electronics,-12499.0
2025-11-20,Big Bazaar Groceries,-4271.0
2025-11-20,Laundry,-668.0
2025-11-21,Myntra Purchase,-3599.0
2025-11-22,Zomato,-983.0
2025-11-23,Office Lunch Reimbursement (received),1200.0
2025-11-24,Intercity Train Ticket,-776.0
2025-11-25,Petrol Station,-2440.0
2025-11-26,Insurance Premium,-1999.0
2025-11-26,Domestic Flight Booking,-8999.0
2025-11-27,Big Bazaar Groceries,-3918.0
2025-11-28,Friend Transfer (received),3000.0
2025-11-29,Cafe - Workday,-968.0
2025-11-30,Laundry,-94.0
```



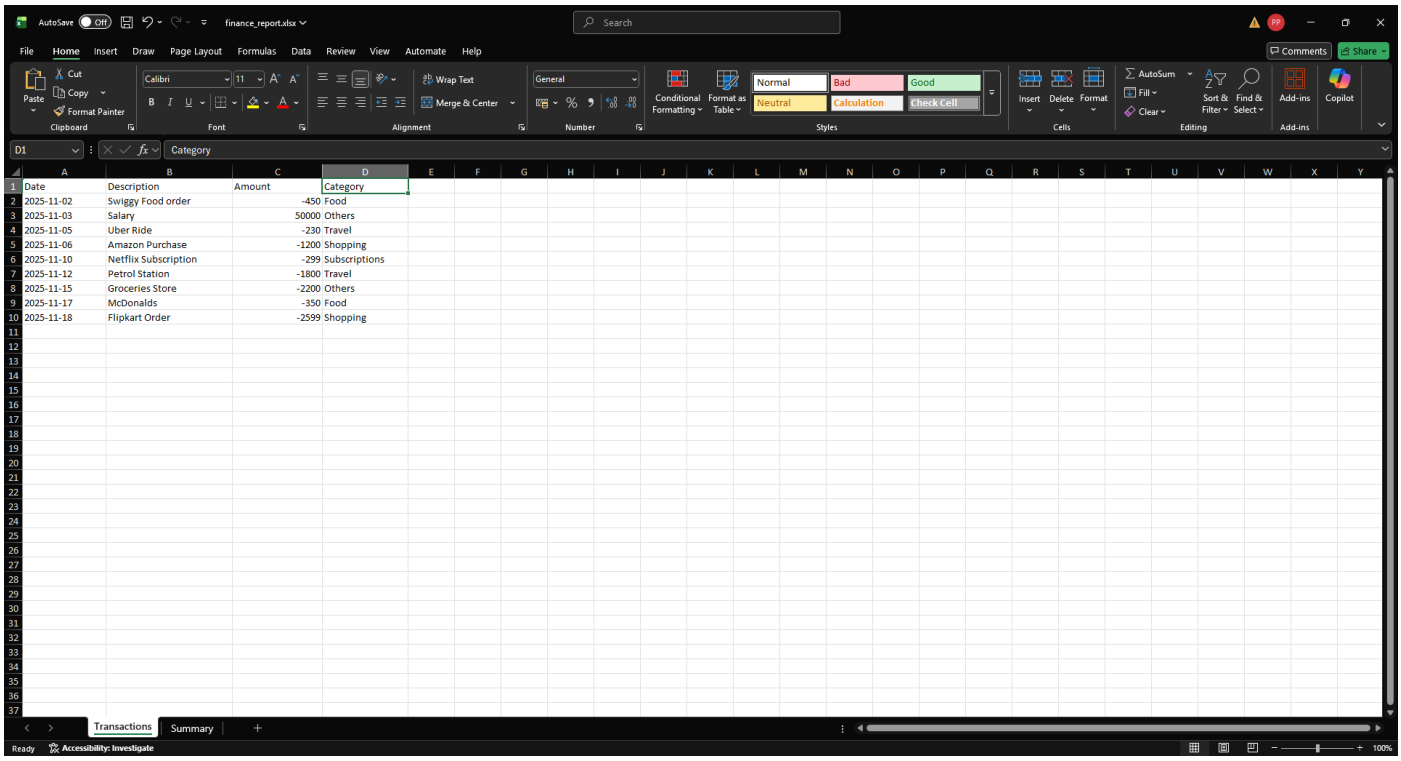
```python
import pandas as pd
import re
import os
from openpyxl import Workbook
from openpyxl.chart import PieChart, Reference
from openpyxl.utils.dataframe import dataframe_to_rows

# CATEGORY MAPPING
CATEGORY_KEYWORDS = {
    "Food": ["swiggy", "zomato", "mcdonald", "kfc", "restaurant", "food", "cafe"],
    "Travel": ["uber", "ola", "train", "flight", "bus", "petrol", "auto"],
    "Shopping": ["amazon", "flipkart", "myntra", "ajio"],
    "Bills": ["electricity", "internet", "wifi", "broadband", "bill"],
    "Subscriptions": ["netflix", "spotify", "hotstar", "prime"],
    "Groceries": ["grocery", "dmart", "big bazaar", "supermarket"],
    "Others": []
}

# CLEAN CATEGORY
def categorize(text):
    text = str(text).lower()
    text = re.sub(r"[^a-z0-9\s]", " ", text)
    for category, keywords in CATEGORY_KEYWORDS.items():
        for kw in keywords:
            if kw in text:
                return category
    return "Others"

# CLEAN AMOUNT (handles ₹, commas, parentheses, etc.)
def clean_amount_column(df):
    df["Amount"] = (
        df["Amount"]
        .astype(str)
        .str.replace(",", "", regex=False)
        .str.replace("₹", "", regex=False)
        .str.replace("(", "-", regex=False)
        .str.replace(")", "", regex=False)
        .str.replace(" ", "", regex=False)
    )
    df["Amount"] = pd.to_numeric(df["Amount"], errors="coerce").fillna(0.0)
    return df

# MAIN FUNCTION
def convert_and_chart(csv_file, xlsx_file="finance_report.xlsx"):
    # Load CSV
    df = pd.read_csv(csv_file)

    # Normalize columns
```



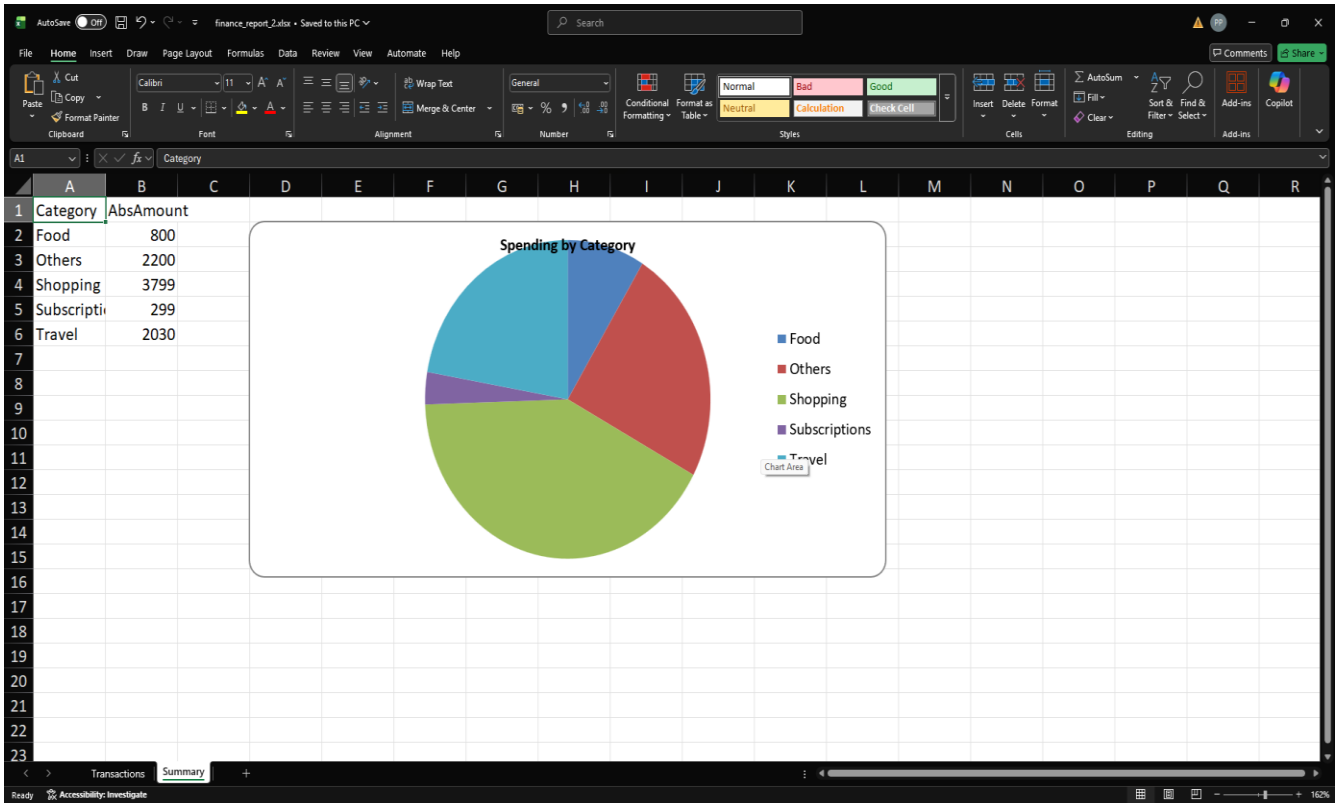
```python
def convert_and_chart(csv_file, xlsx_file="finance_report.xlsx"):
    # Normalize columns
    df.columns = [c.strip() for c in df.columns]

    # Required columns
    if "Description" not in df.columns or "Amount" not in df.columns:
        raise ValueError("CSV must include 'Description' and 'Amount' columns.")

    # Clean Amount field
    df = clean_amount_column(df)

    # Categorize
    df["Description"] = df["Description"].astype(str)
    df["Category"] = df["Description"].apply(categorize)

    # Expenses only
    df_exp = df[df["Amount"] < 0].copy()
    df_exp["AbsAmount"] = df_exp["Amount"].abs()

    summary = df_exp.groupby("Category")["AbsAmount"].sum().reset_index()

    # BUILD THE EXCEL FILE
    wb = Workbook()

    # Sheet 1 → Raw Data
    ws1 = wb.active
    ws1.title = "Transactions"

    for r in dataframe_to_rows(df, index=False, header=True):
        ws1.append(r)

    # Sheet 2 → Summary
    ws2 = wb.create_sheet("Summary")

    for r in dataframe_to_rows(summary, index=False, header=True):
        ws2.append(r)

    # PIE CHART CREATION

    if summary.empty:
        print("No expenses found. Excel file created WITHOUT pie chart.")
        wb.save(xlsx_file)
        os.startfile(xlsx_file)
        return

    if len(summary) == 1:
        print("Only one category found. Pie chart may not display well.")
```



```python
def convert_and_chart(csv_file, xlsx_file="finance_report.xlsx"):
    pie = PieChart()
    pie.title = "Spending by Category"

    start_row = 2
    end_row = len(summary) + 1

    labels = Reference(ws2, min_col=1, min_row=start_row, max_row=end_row)
    data   = Reference(ws2, min_col=2, min_row=1, max_row=end_row)

    pie.add_data(data, titles_from_data=True)
    pie.set_categories(labels)

    ws2.add_chart(pie, "D2")

    # Save Excel
    base = "finance_report"
    counter = 1
    final_name = f"{base}.xlsx"

    while os.path.exists(final_name):
        counter += 1
        final_name = f"{base}_{counter}.xlsx"

    wb.save(final_name)
    print("Saved as:", final_name)
    os.startfile(final_name)


    print("\nExcel file created successfully:", xlsx_file)
    print("Sheets generated: Transactions, Summary + Pie Chart")

    # Auto-open (Windows only)
    try:
        os.startfile(xlsx_file)
    except:
        pass

# RUNNER
if __name__ == "__main__":
    import sys
    if len(sys.argv) < 2:
        print("Usage: python csv_to_excel_pie.py yourfile.csv")
        exit()

    csv_path = sys.argv[1]
    convert_and_chart(csv_path)
```



Date	Description	Amount	Category
2025-11-02	Swiggy Food order	-450	Food
2025-11-03	Salary	50000	Others
2025-11-05	Uber Ride	-230	Travel
2025-11-06	Amazon Purchase	-1200	Shopping
2025-11-10	Netflix Subscription	-299	Subscriptions
2025-11-12	Petrol Station	-1800	Travel
2025-11-15	Groceries Store	-2200	Others
2025-11-17	McDonalds	-350	Food
2025-11-18	Flipkart Order	-2599	Shopping



Category	AbsAmount
Food	800
Others	2200
Shopping	3799
Subscriptio	299
Travel	2030

VIII. Bibliography

1. **Pandas Development Team.** (2024). *pandas: powerful Python data analysis toolkit*. Available: https://pandas.pydata.org
2. **Gazoni, E. & Clark, C.** (2024). *openpyxl: A Python library to read/write Excel 2010 xlsx/xlsm files*. Available: https://openpyxl.readthedocs.io
3. **Pedregosa, F. et al.** (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
4. **Plaid Inc.** (2024). *Plaid API Documentation*. Available: https://plaid.com/docs/
5. **You Need A Budget (YNAB).** (2024). *The Four Rules of Budgeting*. Available: https://www.ynab.com/the-four-rules