SNBP's INTERNATIONAL SCHOOL MANJRI

ACADEMIC YEAR - 2024-25

Name – Aarush Pathak

CBSE Roll No –

SNBP's INTERNATIONAL SCHOOL, MANJRI.

ACADEMIC YEAR : 2024-2025

CERTIFICATE

COMPUTER SCIENCE FOR AISSCE

EXAM 2024-25

This is to certify that the Computer Science (083) Project was done by: Aarush Pathak, of Grade XII, B ,Board Registration Number _____, under my guidance and supervision. It is further certified that the work is original.

Signature of the principal **Signature of the External Examiner**

Signature of the Subject teacher

Date: **School Seal:**

SNBP's INTERNATIONAL SCHOOL, MANJRI

SOFTWARE

FOR

"EXPENSE MANAGEMENT SYSTEM"

DEVELOPED AT

SNBP's INTERNATIONAL SCHOOL

TOOLS USED

FRONT END- PYTHON

BACK END- CSV

PREFACE

The computers have gained a lot of importance in the past five decades. Most of our day-to-day jobs are being influenced by the use of computers. Nowadays, computers are used for performing almost every function, which were performed by humans in the past. In some areas such as science and technology, targets can't be achieved without the use of computers. The characteristics that make the computer so important include its extraordinary speed, large storage capacity, accuracy, and consistency.

Today computers play a great role in various fields and a large number of industries are using computers for maintaining cashbooks, sales books, purchases, etc. Computers can also be used for the designing of various products. Computers provide many options for designing the products.

The analysis of the project has been undertaken with utmost sincerity and honesty and we will be extremely satisfied if the effort is appreciated.

INDEX

ACKNOWLEDGEMENT

I take this opportunity to express my profound sense of gratitude and respect to all those who helped me throughout this venture.

I owe my regards to Mr/Ms/Mr **Sweta Sachwani Ratnapal** Principal of my School for his/her cooperation and valuable support and for giving us the opportunity to undertake this project work and providing the necessary infrastructure.

I would like to express my heartfelt thanks to my revered teacher Mr/Ms/Mrs. **SHEELA SAJJAN SHETTY** for his/her valuable guidance. This project is his visualization and owes a lot of its functionality to her.

Last but not the least, I owe my overwhelming gratitude to my family and friends who gave me constant support and motivation to continue with this endeavor.

Requirements Analysis

1. CSV Module

- **Purpose**: The csv module is essential for handling the storage and retrieval of user and expense data in the data.csv and records.csv files. It facilitates reading and writing to CSV files, making it an efficient choice for lightweight data storage.
- **Usage**:
  - Reading user data for login and registration (csv.reader).
  - Storing and updating expense records (csv.writer).
- **Dependencies**: Built into Python, no external dependencies required.

---

2. OS Module

- **Purpose**: The os module is required to interact with the file system. It is used to create necessary CSV files if they do not exist, ensuring that the application runs smoothly without manual setup.
- **Usage**:
  - Checking for the existence of data.csv and records.csv.
  - Clearing the terminal screen for better user interface experience (os.system('cls') or os.system('clear')).
- **Dependencies**: Built into Python, no external dependencies required.

---

3. Ast Module

- **Purpose**: The ast (Abstract Syntax Trees) module is used for converting string representations of dictionaries back into dictionary objects. This is crucial for parsing and working with data stored in CSV files where the records are saved as string representations.
- **Usage**:
  - Converting stringified expense records from CSV files back into dictionaries for manipulation.
- **Dependencies**: Built into Python, no external dependencies required.

---

4. Matplotlib Module

- **Purpose**: The matplotlib module is used to generate visual reports for the user's expenses. It provides a flexible plotting library that allows for the creation of bar charts, pie charts, and other graphical representations of data.
- **Usage**:
  - Plotting bar charts and pie charts to visualize categorized expenses.
  - Displaying monthly or yearly expense breakdowns in graphical form.
- **Dependencies**: Requires installation via pip install matplotlib.

---

5. Rich Module

- **Purpose**: The rich module is used to improve the terminal interface with enhanced formatting, color, and text layout. It is essential for displaying information in a clear, aesthetically pleasing format, helping users interact with the expense tracker more effectively.
- **Usage**:
  - Printing formatted tables of user details and expense summaries.
  - Using markdown to display welcome messages, home page options, and other sections of the application.
- **Dependencies**: Requires installation via pip install rich.

---

6. Datetime Module

- **Purpose**: The datetime module handles date and time operations, essential for adding timestamps to expense transactions and generating reports for specific time periods.
- **Usage**:
  - Storing and formatting dates for transactions.
  - Filtering expenses by month and year for reporting purposes.
- **Dependencies**: Built into Python, no external dependencies required.

PYTHON SOURCE CODE

```
'''

Expense Tracker made for Grade XII - CBSE Computer Science Project

'''

welcometxt = '''
# 🎉 Welcome to the Expense Tracker! 🎉

## 📊 Overview
The Expense Tracker is a simple and efficient tool designed to help you manage your personal finances.
You can easily track your expenses, categorize them, and review your spending habits over time.

## 🛠️ Features
- **User-Friendly Interface**: Navigate through the application with ease.
- **Categorization**: Organize your expenses into major categories:
  - Food & Dining
  - Transport
  - Housing & Utilities
  - Health & Fitness
  - Fun & Leisure
- **Custom Notes**: Add specific notes to each expense for better tracking.
- **Reports**: Generate monthly and yearly reports to analyze your spending patterns.
'''

loginmain = '''
# 🔐 Login Page

## Welcome to the Expense Tracker!

Please log in to your account or create a new account to get started.

---

### (1) **🔑 Login with Existing User**
### (2) **🆕 Create a New User**
### (3) **👋 Go back to Welcome Page**
'''

hometxt = '''
# 💼 Welcome to the Expense Tracker

---

## 🏠 Home Page

### Choose an Option:
1. ➕ **Add a New Expense**
2. 📜 **View All Expenses**
3. 🔍 **Search for an Expense (Using note or date)**
4. ❌ **Delete an Expense**
5. 📊 **Generate Expense Report**
6. 🚪 **Logout**

---
'''

```

```
searchtext = '''
# Please select search type -

**(1) Search by date 📅**

**(2) Search by note 📝**

**(3) Go back to Home Page 🏠**

'''

categories = '''
# Pick a category of expense

### 1. 🍕 Food & Dining
### 2. 🚗 Transport
### 3. 🏠 Housing & Utilities
### 4. 🏋 Health & Fitness
### 5. 🎉 Fun & Leisure
### 6. 🛠 Others
'''

import csv
import os
import ast
import matplotlib.pyplot as plt

from rich.console import Console
from rich.table import Table
from rich.markdown import Markdown
from rich.panel import Panel
from rich.progress import track
from datetime import datetime

currentDate = datetime.now()
currentMMYY = currentDate.strftime("%m/%Y")
currentUser = None
currentData = {}
dataFile = "./data.csv"
recFile = "./records.csv"

defaultdata = {currentMMYY :
               {"total" : 0,
                "food" : 0, "transport" : 0, "housing" : 0,
                "health" : 0, "leisure" : 0, "other" : 0,
                "transacs" : []}
                }

monthMap = {1: "January", 2: "February", 3: "March", 4: "April",
          5: "May", 6: "June", 7: "July", 8: "August", 9: "September",
          10: "October", 11: "November", 12: "December"}

console = Console()

# Creating Required Files
if not os.path.exists(dataFile):
    with open(dataFile, "a", newline = "") as file:
        pass
if not os.path.exists(recFile):
    with open(recFile, "a", newline = "") as file:
```

```
        pass

# Clearing the Terminal
def cls():
    if os.name == 'nt':  # For Windows
        os.system('cls')
    else:  # For MacOS and Linux
        os.system('clear')

# Welcome Page
def welcome():
    cls()
    console.print(Markdown(welcometxt))
    input("\nPress Enter to continue to the login page...")
    login()

# Checking is user is registered
def isRegistered(username):
    with open(dataFile, "r", newline = "") as file:
        rdr = csv.reader(file)
        for row in rdr:
            if row[0] == username:
                return True
    return False

# Formats date to required format for storage
def formatDate(date):
    day, month, year = date.split("/")

    formatted_day = f"{int(day):02}"
    formatted_month = f"{int(month):02}"

    return f"{formatted_day}/{formatted_month}/{year}"

# Formats the category type from raw storage type to something fancier
def formatCategory(cat):
    if cat == "food":
        return "🍕 Food & Dining"
    elif cat == "transport":
        return "🚗 Transport"
    elif cat == "housing":
        return "🏠 Housing & Utilities"
    elif cat == "health":
        return "🏋️ Health & Fitness"
    elif cat == "leisure":
        return "🎉 Fun & Leisure"
    else:
        return "🛠️ Others"

# Saves data to CSV file
def saveDataToCSV():
    global currentUser, currentData
    rows = []
    user_found = False

    with open(recFile, "r", newline="") as file:
        rdr = csv.reader(file)
        for row in rdr:
            if row[0] == currentUser:
                rows.append([currentUser, currentData])
```

```
                user_found = True
            else:
                rows.append(row)

    if not user_found:
        rows.append([currentUser, currentData])

    with open(recFile, "w", newline="") as file:
        wtr = csv.writer(file)
        wtr.writerows(rows)

# Add a new expense
def addExpense():
    cls()
    global currentUser, currentData, monthMap

    try:
        ddmmyy = formatDate(str(input("Enter date (eg - 25/05/2025 for 25 May 2025): ")))
        day, month, year = map(int, ddmmyy.split("/"))
    except (ValueError, IndexError):
        console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
        input("\nPress Enter to try again...")
        return addExpense()

    if month < 1 or month > 12:
        console.print(Panel("Please enter a valid month (1-12)...",
                            title="Message!", style="bold red"))
        input("\nPress Enter to try again...")
        return addExpense()

    mmyy = str(ddmmyy[3:])

    if mmyy not in currentData:
        currentData[mmyy] = {
            "total": 0,
            "food": 0,
            "transport": 0,
            "housing": 0,
            "health": 0,
            "leisure": 0,
            "other": 0,
            "transacs": []
        }

    cost = int(input("Enter amount spent : "))
    currentData[mmyy]["total"] += cost

    cls()
    console.print(Markdown(categories))
    try:
        cat = int(input("Enter category number : "))
        if cat < 1 or cat > 6:
            console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
            input("\nPress Enter to try again...")
            return addExpense()
    except:
        console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
        input("\nPress Enter to try again...")
```

```
236.         return addExpense()
237.
238.     category_key = ['food', 'transport', 'housing', 'health', 'leisure', 'other'][cat -
1]
239.     currentData[mmyy][category_key] += cost
240.
241.     cls()
242.     note = str(input("Enter note for transaction (leave blank for empty) : "))
243.     newtransac = {"date": day, "amt" : cost, "category": category_key, "note": note if
note != "" else None}
244.
245.     currentData[mmyy]['transacs'].append(newtransac)
246.     saveDataToCSV()
247.
248.     console.print(Panel("Expense added successfully!", title="Success!", style="bold
green"))
249.     input("\nPress Enter to return to the homepage...")
250.     return home()
251.
252. # View Expenses
253. def viewExpense():
254.     cls()
255.     global currentUser, currentData
256.
257.     try:
258.         mmyy = formatDate("00/"+str(input("Enter month to view expenses for (eg - 05/2025
for May 2025) : ")))[3:]
259.         month, year = map(int, mmyy.split("/"))
260.     except (ValueError, IndexError):
261.         console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
262.         input("\nPress Enter to try again...")
263.         return viewExpense()
264.
265.     if month < 1 or month > 12:
266.         console.print(Panel("Please enter a valid month (1-12)...",
267.                             title="Message!", style="bold red"))
268.         input("\nPress Enter to try again...")
269.         return viewExpense()
270.
271.     if mmyy not in currentData:
272.         console.print(Panel("Records for this month do not exist!", title="Message!",
style="bold red"))
273.         input("\nPress Enter to go to home page...")
274.         return home()
275.
276.     currentTransacs = currentData[mmyy]["transacs"]
277.
278.     table = Table(title=f"{monthMap[month]} {year} Detailed Summary", title_style="bold
cyan")
279.
280.     table.add_column("Date", style="bold green", overflow="wrap")
281.     table.add_column("Amount", style="yellow", overflow="wrap")
282.     table.add_column("Category", style="yellow", overflow="wrap")
283.     table.add_column("Note", style="purple", overflow="wrap")
284.
285.     for record in currentTransacs:
286.         table.add_row(str(record["date"]), str(record["amt"]),
287.                       formatCategory(record["category"]), record["note"])
288.
289.     console.print(table)
```

```
290.     console.print(Panel(f"Would you like to view transactions from another month?",
291.                         title="Message!", style="bold yellow"))
292.     res = input("\nRespond with (y) to confirm or (n) to cancel: ")
293.     if res.lower() == "y":
294.         return viewExpense()
295.     else:
296.         return home()
297. 
298. # Search for an expense
299. def searchExpense():
300.     cls()
301.     global currentUser, currentData
302. 
303.     console.print(Markdown(searchtext))
304.     ans = int(input("\nEnter your input: "))
305. 
306.     if ans == 1:
307.         cls()
308.         try:
309.             ddmmyy = formatDate(str(input("Enter date of expense (eg - 25/05/2025 for 25
May 2025) : ")))
310.             date, month, year = map(int, ddmmyy.split("/"))
311.         except (ValueError, IndexError):
312.             console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
313.             input("\nPress Enter to try again...")
314.             return searchExpense()
315. 
316.         if month < 1 or month > 12:
317.             console.print(Panel("Please enter a valid month (1-12)...",
318.                                 title="Message!", style="bold red"))
319.             input("\nPress Enter to try again...")
320.             return searchExpense()
321. 
322.         if ddmmyy[3:] not in currentData:
323.             console.print(Panel(f"No search results found for {ddmmyy}",
title="Message!", style="bold red"))
324.             input("\nPress Enter to go to home page...")
325.             return home()
326. 
327.         currentTransacs = currentData[ddmmyy[3:]]["transacs"]
328.         found = False
329. 
330.         table = Table(title=f"Search Results for {ddmmyy}", title_style="bold cyan")
331. 
332.         table.add_column("Date", style="bold green", overflow="wrap")
333.         table.add_column("Amount", style="yellow", overflow="wrap")
334.         table.add_column("Category", style="yellow", overflow="wrap")
335.         table.add_column("Note", style="purple", overflow="wrap")
336. 
337.         for record in currentTransacs:
338.             if record["date"] == date:
339.                 table.add_row(formatDate(str(record["date"]) + "/" + ddmmyy[3:]),
str(record["amt"]),
340.                               formatCategory(record["category"]), record["note"])
341.                 found = True
342. 
343.         if found:
344.             console.print(table)
345.             console.print(Panel(f"Would you like to search for another transaction?",
346.                         title="Message!", style="bold yellow"))
```

```
347.             ans2 = input("\nRespond with (y) to confirm or (n) to cancel: ")
348.             if ans2.lower() == "y":
349.                 return searchExpense()
350.             else:
351.                 return home()
352.         else:
353.             console.print(Panel(f"No search results found for {ddmmyy}",
title="Message!", style="bold red"))
354.             input("\nPress Enter to go to home page...")
355.             return home()
356.
357.     elif ans == 2:
358.         cls()
359.         query = str(input("\nPlease enter search query : "))
360.
361.         cls()
362.
363.         table2 = Table(title=f"Search Results for \"{query}\"", title_style="bold cyan")
364.
365.         table2.add_column("Date", style="bold green", overflow="wrap")
366.         table2.add_column("Amount", style="yellow", overflow="wrap")
367.         table2.add_column("Category", style="yellow", overflow="wrap")
368.         table2.add_column("Note", style="purple", overflow="wrap")
369.
370.         found2 = False
371.
372.         for item2 in currentData:
373.             transacs = currentData[item2]["transacs"]
374.
375.             for record2 in transacs:
376.                 if query in record2["note"]:
377.                     table2.add_row(formatDate(str(record2["date"]) + "/" + item2),
str(record2["amt"]),
378.                     formatCategory(record2["category"]), record2["note"])
379.                     found2 = True
380.
381.         if found2:
382.             console.print(table2)
383.             console.print(Panel(f"Would you like to search for another transaction?",
384.                         title="Message!", style="bold yellow"))
385.             ans3 = input("\nRespond with (y) to confirm or (n) to cancel: ")
386.             if ans3.lower() == "y":
387.                 return searchExpense()
388.             else:
389.                 return home()
390.         else:
391.             console.print(Panel(f"No search results found for {query}", title="Message!",
style="bold red"))
392.             input("\nPress Enter to try again...")
393.             return searchExpense()
394.
395.     elif ans == 3:
396.         return home()
397.
398.     else:
399.         cls()
400.         console.print(Panel(f"Invalid Input!",
401.                         title="Message!", style="bold red"))
402.         input("\nPress Enter to go back to home page...")
403.         return home()
404.
```

```
405. # Delete Expense
406. def deleteExpense():
407.     cls()
408.     global currentUser, currentData
409.
410.     try:
411.         mmyy = str(input("Enter month for deletion (eg - 05/2025 for May 2025): "))
412.         mmyy = formatDate("00/" + mmyy)[3:]
413.         if mmyy not in currentData:
414.             cls()
415.             console.print(Panel("No records found for this month!", title="Message!",
style="bold red"))
416.             input("\nPress Enter to return to home page...")
417.             return home()
418.     except:
419.         cls()
420.         console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
421.         input("\nPress Enter to try again...")
422.         return deleteExpense()
423.
424.     currentTransacs = currentData[mmyy]["transacs"]
425.     if not currentTransacs:
426.         cls()
427.         console.print(Panel("No records to delete!", title="Message!", style="bold red"))
428.         input("\nPress Enter to return to the homepage...")
429.         return home()
430.
431.     table = Table(title=f"Delete Expense", title_style="bold cyan")
432.     table.add_column("Index", style="cyan")
433.     table.add_column("Date", style="magenta")
434.     table.add_column("Amount (₹)", style="green")
435.     table.add_column("Category", style="yellow")
436.     table.add_column("Note", style="blue")
437.
438.     index = 1
439.     for t in currentTransacs:
440.         table.add_row(str(index), str(t["date"]), str(t["amt"]),
formatCategory(t["category"]), str(t["note"] or ""))
441.         index += 1
442.
443.     console.print(table)
444.     try:
445.         deleteIndex = int(input("Enter index of transaction to delete: ")) - 1
446.         if deleteIndex < 0 or deleteIndex >= len(currentTransacs):
447.             raise ValueError
448.     except ValueError:
449.         console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
450.         input("\nPress Enter to try again...")
451.         return deleteExpense()
452.
453.     deleted = currentTransacs.pop(deleteIndex)
454.     currentData[mmyy]["total"] -= deleted["amt"]
455.     category_key = deleted["category"]
456.     currentData[mmyy][category_key] -= deleted["amt"]
457.     saveDataToCSV()
458.
459.     console.print(Panel("Expense deleted successfully!", title="Success!", style="bold
green"))
460.     input("\nPress Enter to return to the homepage...")
```

```
461.     return home()
462.
463. def expenseReport():
464.     cls()
465.     global currentUser, currentData
466.     console.print(Markdown("# 📊 Expense Report"))
467.
468.     try:
469.         mmyy = str(input("\nEnter month/year for report (eg - 05/2025 for May 2025): "))
470.         mmyy = formatDate("00/" + mmyy)[3:]
471.         if mmyy not in currentData:
472.             console.print(Panel("No records found for this month!", title="Message!",
style="bold red"))
473.             input("\nPress Enter to return to home page...")
474.             return home()
475.     except:
476.         console.print(Panel("Please enter a valid input...", title="Message!",
style="bold red"))
477.         input("\nPress Enter to try again...")
478.         return expenseReport()
479.
480.     total_expenses = currentData[mmyy]["total"]
481.     categories = {
482.         "Food & Dining": currentData[mmyy]["food"],
483.         "Transport": currentData[mmyy]["transport"],
484.         "Housing & Utilities": currentData[mmyy]["housing"],
485.         "Health & Fitness": currentData[mmyy]["health"],
486.         "Fun & Leisure": currentData[mmyy]["leisure"],
487.         "Others": currentData[mmyy]["other"]
488.     }
489.
490.     table3 = Table(title=f"Expense Report for {monthMap[int(mmyy[:2])]} {mmyy[3:]}",
title_style="bold cyan")
491.     table3.add_column("Category", style="bold green")
492.     table3.add_column("Amount (₹)", style="yellow")
493.
494.     for category, amount in categories.items():
495.         table3.add_row(category, str(amount))
496.
497.     table3.add_row("\nTotal", "\n" + str(total_expenses))
498.
499.     cls()
500.     console.print(table3)
501.
502.     console.print(Panel("Would you like to generate a graphical report?",
title="Message!", style="bold yellow"))
503.     res4 = input("\nRespond with (y) to confirm or (n) to cancel: ")
504.
505.     while True:
506.         cls()
507.         if res4.lower() == "y":
508.             console.print(Markdown("# Choose the representation format:"))
509.             console.print(Markdown("1. 📊 Bar Chart\n2. 📈 Line Graph\n3. 🥧 Pie
Chart"))
510.
511.             try:
512.                 choice = int(input("\nEnter your choice : "))
513.             except ValueError:
514.                 cls()
```

```
515.                 console.print(Panel("Invalid input! Please enter a number (1, 2, or 3).",
title="Message!", style="bold red"))
516.                 return home()
517.
518.             if choice == 1:
519.                 plt.figure(figsize=(12, 6))
520.                 colors = ['#FF5733', '#33FF57', '#3357FF', '#FFC300', '#DAF7A6',
'#FF33F6']
521.                 bars = plt.bar(list(categories.keys()), list(categories.values()),
color=colors, width=0.4)
522.
523.                 plt.xlabel("Categories", fontsize=14, fontweight='bold')
524.                 plt.ylabel("Amount (₹)", fontsize=14, fontweight='bold')
525.                 plt.title(f"Expense Report for {monthMap[int(mmyy[:2])]} {mmyy[3:]}",
fontsize=16, fontweight='bold')
526.                 plt.xticks(rotation=15)
527.                 plt.yticks(fontsize=12)
528.                 plt.grid(axis='y', linestyle='--', alpha=0.7)
529.
530.                 for bar in bars:
531.                     yval = bar.get_height()
532.                     plt.text(bar.get_x() + bar.get_width()/2, yval + 5, int(yval),
ha='center', va='bottom', fontsize=10)
533.
534.                 plt.tight_layout()
535.                 plt.show()
536.
537.             elif choice == 2:
538.                 plt.figure(figsize=(12, 6))
539.                 plt.plot(list(categories.keys()), list(categories.values()), marker='o',
linestyle='-', color='b')
540.
541.                 plt.xlabel("Categories", fontsize=14, fontweight='bold')
542.                 plt.ylabel("Amount (₹)", fontsize=14, fontweight='bold')
543.                 plt.title(f"Expense Report for {monthMap[int(mmyy[:2])]} {mmyy[3:]}",
fontsize=16, fontweight='bold')
544.                 plt.xticks(rotation=15)
545.                 plt.grid()
546.
547.                 plt.tight_layout()
548.                 plt.show()
549.
550.             elif choice == 3:
551.                 plt.figure(figsize=(5, 5))
552.
553.                 labelList = []
554.                 for thing in categories.keys():
555.                     if categories[thing] != 0:
556.                         labelList.append(thing)
557.
558.                 valueList = []
559.                 for item2 in categories.values():
560.                     if item2 != 0:
561.                         valueList.append(item2)
562.
563.                 plt.pie(valueList, labels = labelList, autopct='%1.1f%%', startangle=140)
564.
565.                 plt.title(f"Expense Distribution for {monthMap[int(mmyy[:2])]}
{mmyy[3:]}", fontsize=16, fontweight='bold')
566.                 plt.axis('equal')
567.
```

```
568.                 plt.tight_layout()
569.                 plt.show()
570.
571.             else:
572.                 console.print(Panel("Invalid choice! Please select 1, 2, or 3.",
title="Message!", style="bold red"))
573.
574.             cls()
575.             console.print(Markdown("# Please enter your choice - "))
576.             console.print(Markdown("### Type 'y' to view data in another format"))
577.             console.print(Markdown("### Type 'n' to exit to home page"))
578.             response = input("\nEnter your choice : ")
579.             if response.lower() != "y":
580.                 break
581.         else:
582.             return home()
583.
584.     return home()
585.
586. # Creating a new user account
587. def createAccount():
588.     cls()
589.
590.     details = {"Username" : "", "Email" : "", "Phone Number" : 0, "Password" : ""}
591.
592.     #Filling Values in dictionary from user input
593.     for key in details.keys():
594.         try:
595.             if key == "Phone Number" : details[key] = int(input(f"\nEnter {key} : "))
596.             else : details[key] = str(input(f"\nEnter {key} : "))
597.         except ValueError:
598.             cls()
599.             console.print(Panel(f"Please Enter a valid 10 digit phone number...",
600.                         title="Message!", style="bold red"))
601.             input("\nPress Enter to go back to login menu...")
602.             return login()
603.
604.     # Check Empty Fields
605.     if details["Username"] == "" or details["Email"] == "" or details["Password"] == "":
606.         cls()
607.         console.print(Panel(f"Some fields are left blank, please try again!",
608.                         title="Message!", style="bold red"))
609.         input("\nPress Enter to go back to login menu...")
610.         return login()
611.
612.     username = details["Username"]
613.
614.     # Checking if the username already exists
615.     check = isRegistered(details["Username"])
616.     if check:
617.         console.print(Panel(f"User {username} already exists",
618.                         title="Message!", style="bold red"))
619.         input("\nPress Enter to go back to login menu...")
620.         return login()
621.
622.     # Confirm User Details
623.     final = Table(title="Expense Tracker", title_style="bold cyan")
624.
625.     final.add_column("Details", style="bold green")
626.     final.add_column("Data Entered", style="yellow")
627.
```

```
628.     for key in details.keys():
629.         final.add_row(str(key), str(details[key]))
630.
631.     cls()
632.     console.print(Panel(f"Would you like to confirm account creation?",
633.                         title="Message!", style="bold yellow"))
634.     console.print(final)
635.     res = input("\nRespond with (y) to confirm or (n) to cancel: ")
636.     if res.lower() != "y":
637.         return login()
638.
639.     # Register user details to backend
640.     with open(dataFile, "a+", newline = "") as file:
641.         wtr = csv.writer(file)
642.         wtr.writerow(details.values())
643.
644.     with open(recFile, "a+", newline = "") as file:
645.         global defaultdata
646.         wtr = csv.writer(file)
647.         wtr.writerow([details["Username"], defaultdata])
648.
649.     # Print Completion
650.     console.print(Panel(f"User {username} successfully registered!",
651.                         title="Message!", style="bold green"))
652.     input("\nPress Enter to go back to login menu...")
653.     return login()
654.
655. # User Login
656. def login():
657.     cls()
658.     global currentUser, currentData
659.
660.     console.print(Markdown(loginmain))
661.
662.     try:
663.         res = int(input("\nEnter your input : "))
664.     except:
665.         cls()
666.         console.print(Panel(f"Please enter a valid input!",
667.                         title="Message!", style="bold red"))
668.         input("\nPress Enter to go back to login menu...")
669.         return login()
670.
671.     if res == 1:
672.         pass
673.     elif res == 2:
674.         createAccount()
675.         return login()
676.     elif res == 3:
677.         return welcome()
678.     else:
679.         console.print(Panel(f"Please enter a valid input!",
680.                         title="Message!", style="bold red"))
681.         input("\nPress Enter to go back to login menu...")
682.         return login()
683.
684.     cls()
685.
686.     try:
687.         username = str(input("\nPlease enter your username : "))
688.     except:
```

```
689.         cls()
690.         console.print(Panel(f"Please enter a valid input!",
691.                         title="Message!", style="bold red"))
692.         input("\nPress Enter to go back to login menu...")
693.         return login()
694. 
695.     # Checking if user is registered or not
696.     check = isRegistered(username)
697.     if check == False:
698.         console.print(Panel(f"User {username} is not registered!",
699.                         title="Message!", style="bold red"))
700.         input("\nPress Enter to go back to login menu...")
701.         return login()
702. 
703.     # Verifying login and redirecting to homepage
704.     cls()
705.     password = str(input(f"\nHey {username}, please enter your password : "))
706. 
707.     with open(dataFile, "r", newline = "") as file:
708.         rdr = csv.reader(file)
709.         for row in rdr:
710.             if row[0] == username and row[3] == password:
711.                 currentUser = username
712.                 with open(recFile, "r", newline = "") as file:
713.                     rdr = csv.reader(file)
714.                     for row in rdr:
715.                         if row[0] == username:
716.                             currentData = ast.literal_eval(row[1])
717.                             break
718.                 console.print(Panel(f"User {username} successfully logged in!",
719.                         title="Message!", style="bold green"))
720.                 input("\nPress Enter to go to the homepage...")
721.                 return home()
722. 
723.             elif row[0] == username and row[3] != password:
724.                 console.print(Panel(f"Incorrect password entered!",
725.                         title="Message!", style="bold red"))
726.                 input("\nPress Enter to back to the login page again...")
727.                 return login()
728. 
729. # User Logout
730. def logout():
731.     global currentUser, currentData
732. 
733.     currentUser = None
734.     currentData = {}
735.     return main()
736. 
737. # Homepage for the app
738. def home():
739.     cls()
740.     global currentUser
741. 
742.     console.print(Markdown(hometxt))
743. 
744.     try:
745.         res = int(input("\nEnter your input : "))
746. 
747.         if res == 1:
748.             return addExpense()
749.         elif res == 2:
```

```
750.             return viewExpense()
751.         elif res == 3:
752.             return searchExpense()
753.         elif res == 4:
754.             return deleteExpense()
755.         elif res == 5:
756.             return expenseReport()
757.         elif res == 6:
758.             return logout()
759.         else:
760.             cls()
761.             console.print(Panel(f"Invalid Input Entered...(Only 1, 2, 3, 6 working for
now)",
762.                                 title="Message!", style="bold red"))
763.             input("\nPress Enter to back to the home page again...")
764.             return home()
765. 
766.     except:
767.         cls()
768.         console.print(Panel(f"Invalid Input Entered...(Only 1 and 6 working for now)",
769.                             title="Message!", style="bold red"))
770.         input("\nPress Enter to back to the home page again...")
771.         return home()
772. 
773. # Parent Function of all Functions
774. def main():
775. 
776.     welcome()
777. 
778.     login()
779. 
780. main()
781. 
782. 
```

OUTPUTS

Welcome Page

🎉 **Welcome to the Expense Tracker!** 🎉

📊 Overview

The Expense Tracker is a simple and efficient tool designed to help you manage your personal finances. You can easily track your expenses, categorize them, and review your spending habits over time.

🛠️Features

- **User-Friendly Interface**: Navigate through the application with ease.
- **Categorization**: Organize your expenses into major categories:
  - Food & Dining
  - Transport
  - Housing & Utilities
  - Health & Fitness
  - Fun & Leisure
- **Custom Notes**: Add specific notes to each expense for better tracking.
- **Reports**: Generate monthly and yearly reports to analyze your spending patterns.

Press Enter to continue to the login page...

Login Page

🔐 **Login Page**

Welcome to the Expense Tracker!

Please log in to your account or create a new account to get started.

---

(1) 👤 Login with Existing User

(2) 🆕 Create a New User

(3) 👋 Go back to Welcome Page

Enter your input :

Creating a New User

```
Enter Username : 1
Enter Email : test@gmail.com
Enter Phone Number : 5555511111
Enter Password : Secure@123
```

```
Message!
Would you like to confirm account creation?
```

Expense Tracker

Details	Data Entered
Username	1
Email	test@gmail.com
Phone Number	5555511111
Password	Secure@123

```
Respond with (y) to confirm or (n) to cancel: y
Message!
User 1 successfully registered!

Press Enter to go back to login menu...
```

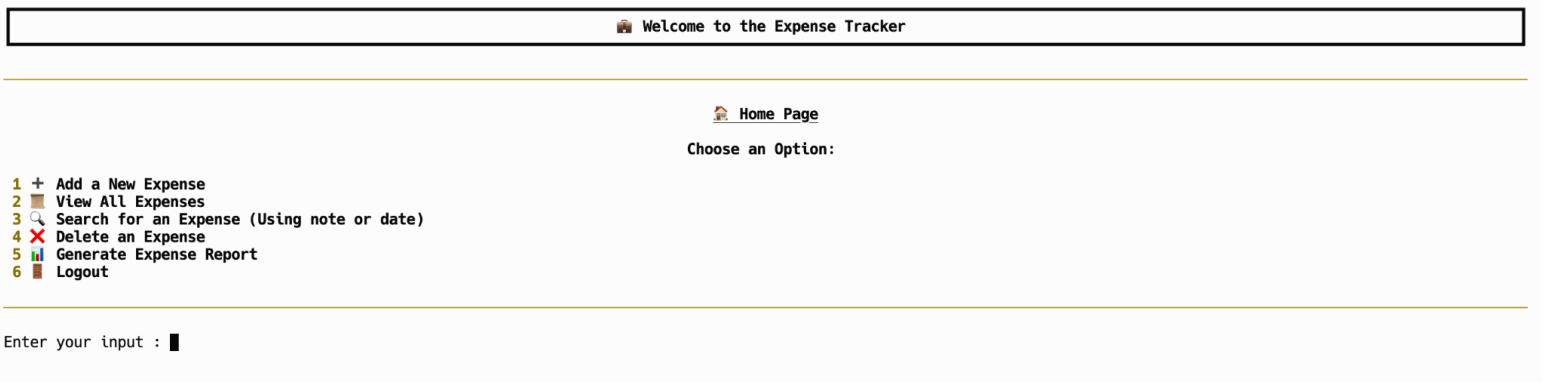
Login with Existing User

```
Please enter your username : 1
```

```
Hey 1, please enter your password : Secure@123
Message!
User 1 successfully logged in!

Press Enter to go to the homepage...
```

Home Page



Adding a New Expense

Viewing Expenses

```
Enter month to view expenses for (eg - 05/2025 for May 2025) : 06/2023
                June 2023 Detailed Summary
```

Date	Amount	Category	Note
15	3026	🏠 Housing & Utilities	
12	2034	🍕 Food & Dining	
28	4023	🎉 Fun & Leisure	
30	5734	🏋️Health & Fitness	Gym Memebership
8	4832	🛠️Others	Travel
3	299000	🚗 Transport	New car

```
Message!
Would you like to view transactions from another month?

Respond with (y) to confirm or (n) to cancel: 
```

Searching for an Expense

```
Please select search type -

(1) Search by date 📅
(2) Search by note 📝
(3) Go back to Home Page 🏠
Enter your input: 1
```

```
Enter date of expense (eg - 25/05/2025 for 25 May 2025) : 15/06/2023
            Search Results for 15/06/2023
```

Date	Amount	Category	Note
15/06/2023	3026	🏠 Housing & Utilities	

```
Message!
Would you like to search for another transaction?

Respond with (y) to confirm or (n) to cancel: 
```

Deleting an Expense

```
Enter month for deletion (eg - 05/2025 for May 2025): 06/2023
```

Delete Expense

Index	Date	Amount (₹)	Category	Note
1	15	3026	🏠 Housing & Utilities	
2	12	2034	🍕 Food & Dining	
3	28	4023	🎉 Fun & Leisure	
4	30	5734	🏋️Health & Fitness	Gym Memebership
5	8	4832	🛠️Others	Travel
6	3	299000	🚗 Transport	New car

```
Enter index of transaction to delete: 6
```

Success!

Expense deleted successfully!

```
Press Enter to return to the homepage...
```

Generating Expense Report

📊 Expense Report

```
Enter month/year for report (eg - 05/2025 for May 2025): 06/2023
```

Expense Report for June 2023

Category	Amount (₹)
Food & Dining	2034
Transport	0
Housing & Utilities	3026
Health & Fitness	5734
Fun & Leisure	4023
Others	4832
Total	19649

Message!

Would you like to generate a graphical report?

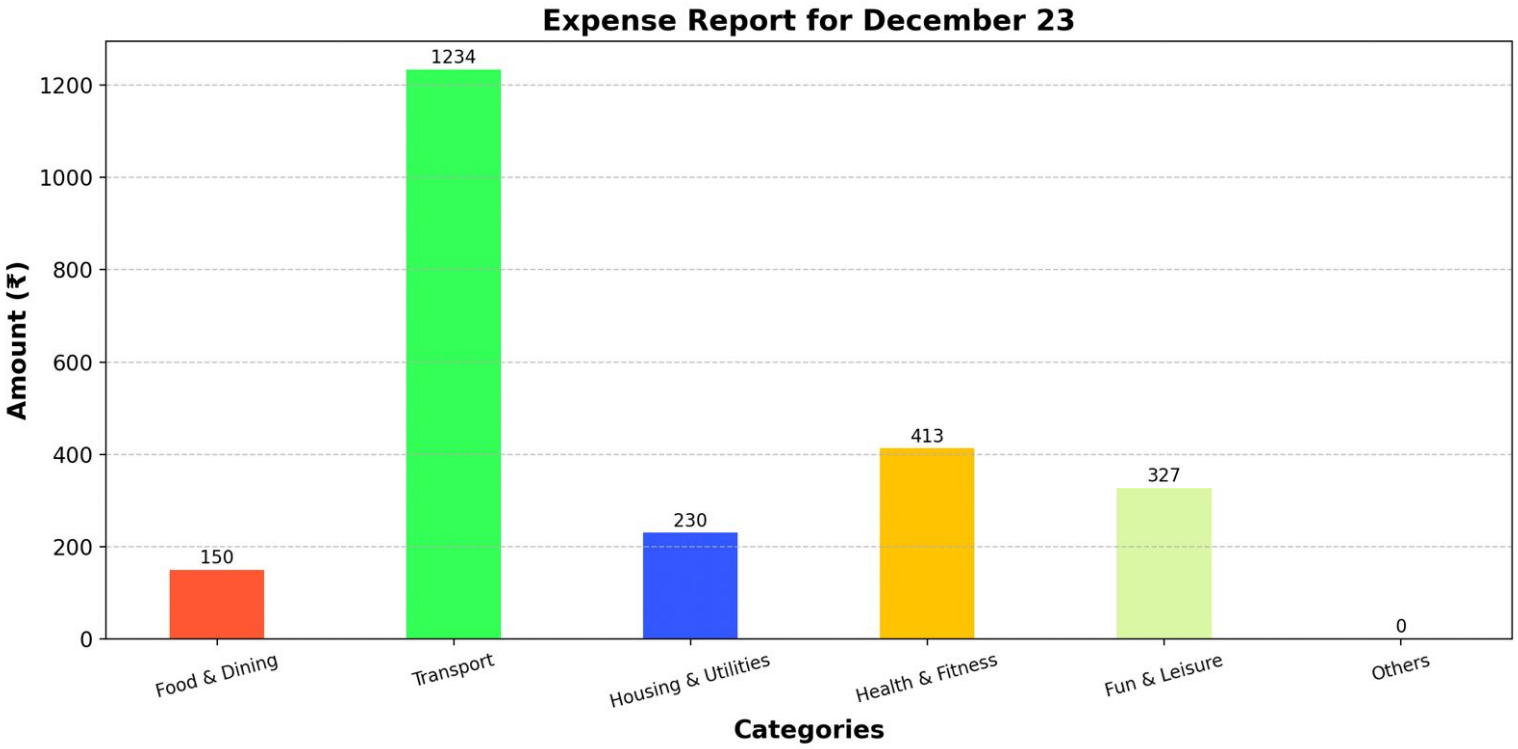
```
Respond with (y) to confirm or (n) to cancel: y
```

Choose the representation format:

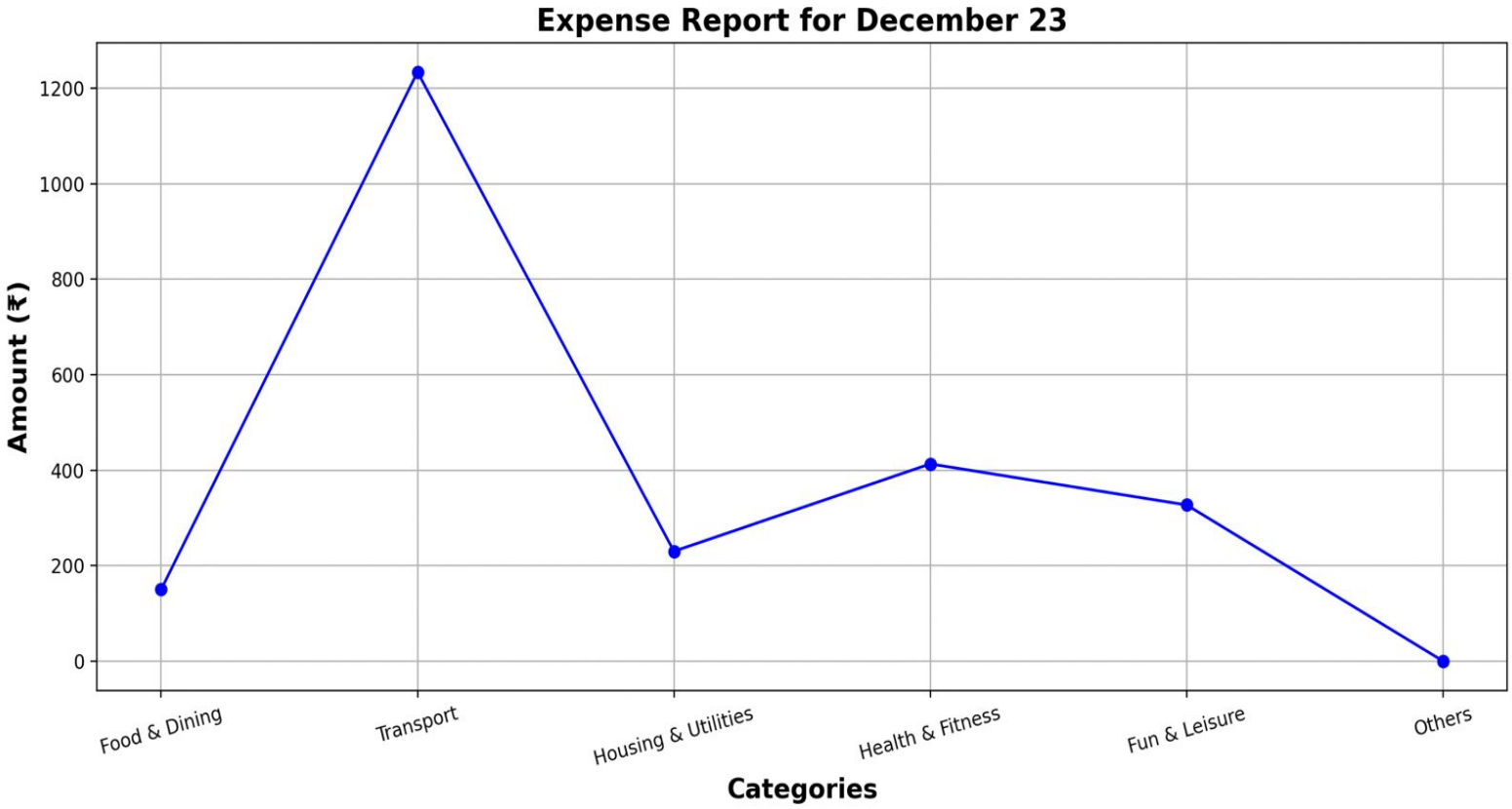
1 📊 Bar Chart
2 📈 Line Graph
3 🥧 Pie Chart

```
Enter your choice :
```

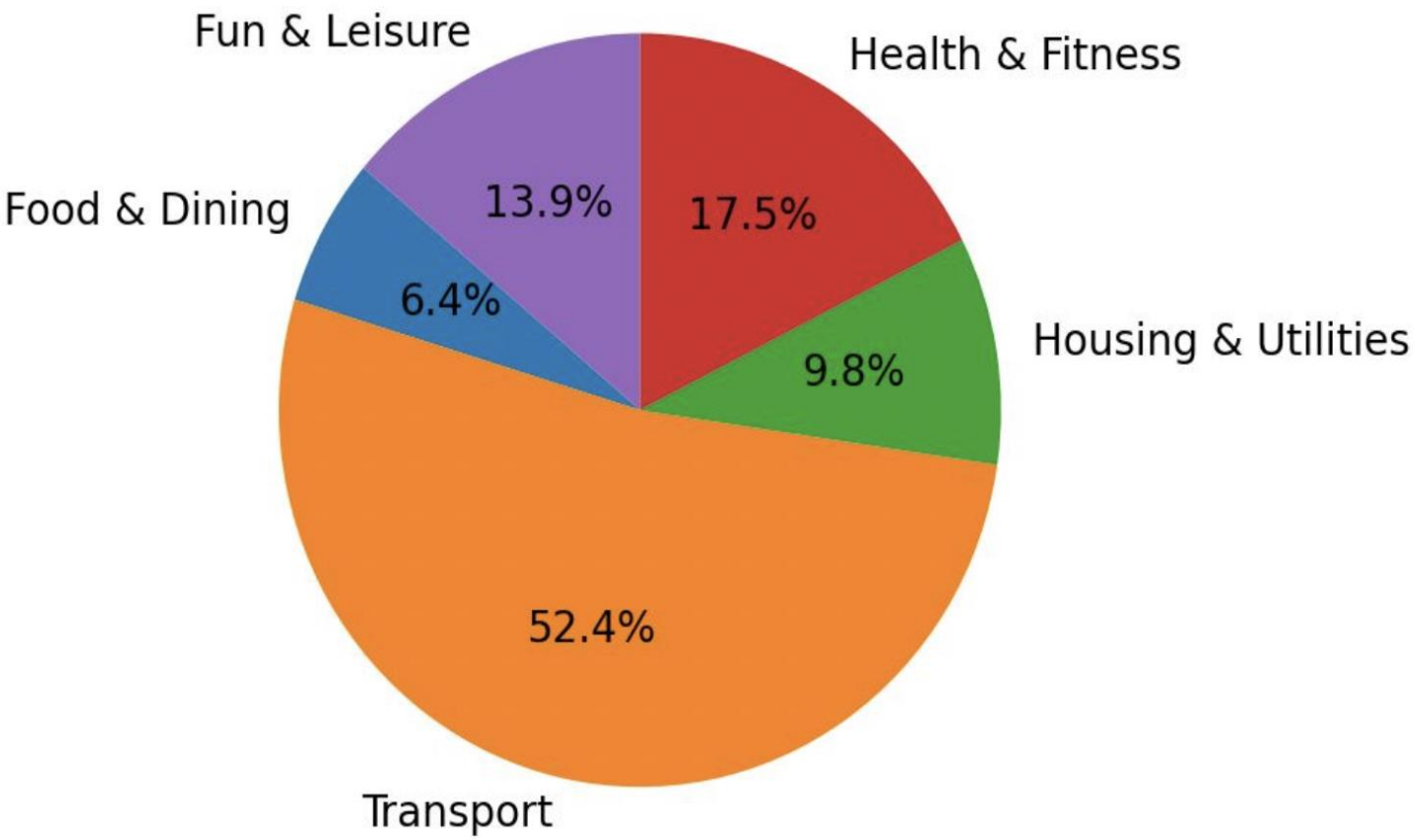
Bar Chart (Matplotlib)



Line Graph (Matplotlib)



Pie Chart (Matplotlib)



CSV FILES

Structuring the CSV File

```
###data.csv###

[username, email, phno, password]

###records.csv###
[username, {month/year :

                {total : int,

                food : int,
                transport : int,
                housing : int,
                health : int,
                leisure : int,
                other : int
                transacs : [{date : int, amt : int, cat : int, note : str}, ..........]
                }

        }
]

###currentData###
currentdata = {"currentMMYY" :
               {"total" : 0,
                "food" : 0, "transport" : 0, "housing" : 0,
                "health" : 0, "leisure" : 0, "other" : 0,
                "transacs" : []}
                }

###transac###
newtransac = {"date": day, "amt" : cost, "category": category_key, "note": note}
```

Data CSV File

data.csv

```
1,test@gmail.com,5555511111,Secure@123
```

Records CSV File

records.csv

```
1,"{'10/2024': {'total': 0, 'food': 0, 'transport': 0, 'housing': 0,
'health': 0, 'leisure': 0, 'other': 0, 'transacs': []}, '06/2023':
{'total': 19649, 'food': 2034, 'transport': 0, 'housing': 3026,
'health': 5734, 'leisure': 4023, 'other': 4832, 'transacs': [{'date':
15, 'amt': 3026, 'category': 'housing', 'note': None}, {'date': 12,
'amt': 2034, 'category': 'food', 'note': None}, {'date': 28, 'amt':
4023, 'category': 'leisure', 'note': None}, {'date': 30, 'amt': 5734,
'category': 'health', 'note': 'Gym Memebership'}, {'date': 8, 'amt':
4832, 'category': 'other', 'note': 'Travel'}]}, '06/2025': {'total':
1034, 'food': 0, 'transport': 1034, 'housing': 0, 'health': 0,
'leisure': 0, 'other': 0, 'transacs': [{'date': 21, 'amt': 1034,
'category': 'transport', 'note': None}]}}"
```

BIBLIOGRAPHY

- **Wikipedia**
  https://www.wikipedia.org/

- **Python**
  https://www.python.org/

- **Python's "csv" Module**
  https://docs.python.org/3/library/csv.html
- **Python's "os" Module**
  https://docs.python.org/3/library/os.html
- **Python's "datetime" Module**
  https://docs.python.org/3/library/datetime.html
- **Python's "matplotlib" Module**
  https://matplotlib.org
- **Python's "ast" Module**
  https://docs.python.org/3/library/ast.html

- **Textualize's "rich" Module**
  https://github.com/Textualize/rich

- **Class 11th & 12th Computer Science Books**