In [61]:
```
##### BUDGET TRACKER ####
import csv
from datetime import datetime

#1-Function to add new expenses
def add_expenses():
    print("\nAdding new expenses...")

    # 1-> Get the first input - date of transaction
    date_input = input("Date YYYY-MM-DD: ")
    # Keep asking untill the date is valid
    while not is_valid_date(date_input):
        print("Invalid date! Please enter a valid date in YYYY-MM-DD format.")
        date_input = input("Date YYYY-MM-DD: ") # ask for date input again

    # 2-> Input Category expenses and check if it in category_list
    category_input = in_category(input("Category (Food/Closing/Car/Misc): ").strip(), 'Categories')

    # 3-> Input Amount of expenses, catch a ValueError if not a number
    while True:
        try:
            amount_input = round(float(input("Amount: ")), 2)
            break
        except ValueError:
            print("Invalid input! Please enter a valid number.")

    # 4-> Input Description for expenses. Must be not empty
    description_input = if_empty(input("Description: ").strip(), 'Description')

    # When all of the entries are valid collect them in dictionary
    new_expense = {
        'date': date_input,
        'category': category_input,
        'amount': amount_input,
        'description': description_input
    }

    # Add new expenses to the all_expenses dictionary
    all_expenses.append(new_expense)
```

```
#2-Function to show all expenses
def view_expenses():
    print("\nShowing expenses...")
    # Sort the list of expenses by date YYYY-MM-DD
    sorted_all_expenses = sorted(all_expenses, key=lambda x: datetime.strptime(x['date'], "%Y-%m-%d"))
    # Print the header
    print(f"\n{'Date':<12} | {'Category':<10} | {'Amount':<11} | Description")
    print("-" * 50)
    # Print sorted entries
    for exp in sorted_all_expenses:
        print(f"{exp['date']:<12} | {exp['category']:<10} | ${exp['amount']:<11} | {exp['description']}")

#3-Function to track the budget and check for over budget
def track_budget ():
    # Would user like add a new budget
    add_new = input("\nWould you like to enter a new budget? y/n: ").lower()
    while add_new != "y" and add_new != "n":
        add_new = input("\nPlease, make your choice y/n").lower()

    ################## YES add a new budget ---> START ###################
    if add_new == "y":
        print("Adding a new budget...\n")

        # 1-> Get month-year input MM-YYYY
        month_year = input("\nEnter month-year to plan a budget (MM-YYYY): ").strip()
        # Check month-year format
        while not is_valid_date(month_year):
            print("Invalid date! Please enter a valid date.")
            month_year = input("Date MM-YYYY: ")
        # Changing budget_list[] date format for future comparation with all_expenses[] date format
        parts = month_year.split("-")
        month_year = f"{parts[1]}-{parts[0]}"

        # 2-> Input budget and catch a ValueError if not a number
        while True:
            try:
                amount_input = round(float(input(f"Enter your budget for {month_year}: ")), 2)
                break
            except ValueError:
                print("Invalid input! Pleace enter a valid number.")
```

```
        # Get existing values of 'expenses' and 'saved' if the mounth is in the budget_list,
        # othewise default  0.0
        existing_budget = next((bdg for bdg in budget_list if bdg['date'] == month_year), None)
        # Create a new dictionary
        new_budget = {
            'date': month_year,
            'budget': amount_input,
            'expenses': existing_budget['expenses'] if existing_budget else 0.0,
            'saved': existing_budget['saved'] if existing_budget else 0.0
        }

        # Check for duplicates and remove old entry
        date_exists = any(entry['date'] == new_budget['date'] for entry in budget_list)
        if date_exists:
            print(f"Budget for {new_budget['date']} is already exist !")
            yes_no = input("Would you like to overwrite existing entry Y/N: ").lower()
            while yes_no != 'y' and yes_no != 'n':
                yes_no = input("\nPlease, make your choice y/n").lower()
            if yes_no == 'y':
                budget_list[:] = [entry for entry in budget_list if entry['date'] != new_budget['date']]
            else:
                print("Budget not overwritten.")
                new_budget = None
        if new_budget:
            # Add the new budget to the list
            budget_list.append(new_budget)
            # Write the updated budget list back to CSV
            # Save_to_file(budget_list, csv_budget_file)
            print("\nUPDATED:")
            print(f"Your budget for {new_budget['date']} is: ${new_budget['budget']}")

    ################### YES add a new budget--> FINISH ####################

    ################### NO add a new budget---> START ####################
#### Show the budget statement with expenses and saved amount ####

    # Convert categorys in budget_list and all_expenses from string format to float
    str_to_float(all_expenses, budget_list)

    # Cconvert date format in budget_list from MM-YYYY to YYYY-MM
```

```
    for dat in budget_list:
        parts = dat['date'].split("-")
        if len(parts[0]) == 2:
            dat["date"] = f"{parts[1]}-{parts[0]}"

    # Store total expenses for the month
    for bdg in budget_list:
        summ = 0.0     #reset expenses to 0.0 before each month analyzing
        for xps in all_expenses:
            if xps['date'].startswith(bdg['date']):
                summ += xps['amount']
        bdg['expenses'] = summ
        if bdg['expenses'] == 0:
            bdg['saved'] = 0.0
        else:
            bdg['saved'] = bdg ['budget'] - bdg['expenses']

    # Write the updated budget list back to CSV
    save_to_file(budget_list, csv_budget_file)
    print_budget(budget_list)

    # If the expenses are within the budget
    current_date = f"{datetime.now().year}-{datetime.now().month:02d}"
    for bdg in budget_list:
        if bdg['date'] == current_date:
            sav = bdg['saved']
            if sav < 0:
                print("\n !!!! Your budget for current month is exceeded. !!!!")
            elif 0 < bdg['saved'] < 250:
                print("\n !!!! You have less then $250 left for the month. !!!!")


    ################## NO add a new budget---> FINISH ######################

#4-Function to save into .csv file
def save_to_file(f_save, csv_name):
    #print("\nSaving to file...")
    # Write the updated list of expenses back to .CSV
    with open(csv_name, 'w', newline='') as csv_file:
        fieldnames = f_save[0].keys()
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
```

```
        writer.writeheader()
        writer.writerows(f_save)
    print("\nSaved to file...")

#5-Function to save expenses and exit
def save_and_exit():
    print("\nSaving data and exiting...")
    # Write the updated list of expenses back to .CSV
    save_to_file(all_expenses, csv_expenses_file)
    return False

# Read .csv file to list of dictionaries
def read_from_file(csv_name, list_name):
    try:
        with open(csv_name, 'r') as csv_file:
            csv_content = csv.DictReader(csv_file)
            for row in csv_content:
                list_name.append(row)
    except FileNotFoundError:
        pass

# Is valid date format
def is_valid_date(new_exp_data):
    # check for budget_list date format MM-YYYY
    if len(new_exp_data) == 7 and new_exp_data[2] == '-':
    # convert to YYYY-MM-DD format
        parts = new_exp_data.split("-")
        new_exp_data = f"{parts[1]}-{parts[0]}-01"
    # check for all_expenses list date format YYYY-MM-DD
    if len(new_exp_data) != 10 or new_exp_data[4] != '-' or new_exp_data[7] != '-':
        return False
   # ensure date format is valid
    year, month, day = new_exp_data.split('-')
    current_year = datetime.now().year
    current_month = datetime.now().month
    return(
        year.isdigit() and len(year) == 4 and
        int(year) == current_year and
        month.isdigit() and 1<= int(month) <=12 and
        int(month) <= current_month and
        day.isdigit() and 1<= int(day) <=31
```

```
    )

# Convert numbers from string format to float
def str_to_float(f_expenses, f_budget=None):
    # Process expenses
    for exps in f_expenses:
        exps['amount'] = float(exps['amount'])
    # Process budget if provided
    for bdg in f_budget:
        bdg['budget'] = float(bdg['budget'])
        bdg['expenses'] = float(bdg['expenses'])
        bdg['saved'] = float(bdg['saved'])

# Print budget in tabular format
def print_budget(f_budget):
    #Sort budget_list by date in any format %m-%Y or %Y-%m
    sorted_f_budget = sorted(f_budget, key=lambda x: datetime.strptime(
        x['date'], "%m-%Y" if "-" in x['date'] and len(x['date'].split("-")[0]) == 2 else "%Y-%m"))

    print("\nYour budget by month for 2025")
    # Print the header
    print(f"\n{'Data':<9} | {'Budget':<10} | {'Total expenses':<17} | Saved")
    print("-" * 55)
    # Print sorted entries
    for dat in sorted_f_budget:
        print(f"{dat['date']:<9} | {dat['budget']:<10} | {dat['expenses']:<17} | {dat['saved']}")

# Is input empty
def if_empty(num, name):
    while len(num) == 0:
        num = input(f"Input is empty, please reenter your {name}").strip()
    return num

# If the Category is in the Set
def in_category(input_name, name):
    #category_list = ['Food', 'Closing', 'Car', 'Misc']
    while input_name not in category_list:
        input_name = input(f"Please reenter one of available {name} \n\t Food/Closing/Car/Misc:\t").strip()
    return input_name
```

```
# Interactive menu and user input handling
def print_menu():
    print("\nWhat do you like to do?\n")
    print("1. Add new expenses")
    print("2. See my expenses")
    print("3. Track the budget")
    print("4. Save expenses to the file")
    print("5. Save the expenses and exit")

    while True:
        try:
            choice = int(input("\nEnter the number of your choice: "))
            if 1 <= choice <= 5:
                return choice
            else:
                print("Invalid choice. Please try again.")
        except ValueError:
            print("Invalid input. Please enter a number.")

############################## global scope ###################################

category_list = ['Food', 'Closing', 'Car', 'Misc']
csv_budget_file = 'budget_editable_2.csv'
csv_expenses_file = 'all_expenses_01.csv'
all_expenses = []
budget_list = []
# Read .csv file to list of dictionaries
read_from_file(csv_expenses_file, all_expenses)
read_from_file(csv_budget_file, budget_list)

             ############### Main loop to handle user interaction ###############
def main():

    menu_actions = {
        1: add_expenses,
        2: view_expenses,
        3: track_budget,
        4: lambda: save_to_file(all_expenses, csv_expenses_file),
        5: save_and_exit
    }
```

```
    # Keep the program running until the user exits
    while True:
        user_choice = print_menu()
        action = menu_actions[user_choice]

        lambda: save_to_file(all_expenses, csv_expenses_file),
        # Execute the chosen function. If it returns False, exit the loop.
        if action() is False:
            break

if __name__ == "__main__":
    main()
```

```
What do you like to do?

1. Add new expenses
2. See my expenses
3. Track the budget
4. Save expenses to the file
5. Save the expenses and exit
Saving data and exiting...

Saved to file...
```

In [ ]: