PSEUDOCODE

```
Import json
Initialize global transactions dictionary

FUNCTION load_transactions()
  GLOBAL transactions
  TRY
    WITH OPEN 'Your_transactions.json' for reading as file
    READ file contents into file_content
    IF file_content is not empty
      PARSE file_content as JSON and store in transactions
    ELSE
      INITIALIZE transactions as an empty dictionary
  EXCEPT FileNotFoundError
    INITIALIZE transactions as an empty dictionary

FUNCTION save_transactions()
  GLOBAL transactions
  TRY
    CREATE formatted_transactions as an empty dictionary
    FOR EACH category, category_transactions pair in transactions
      CREATE a new list in formatted_transactions with category as the key
      FOR EACH transaction in category_transactions
        CREATE a new dictionary with 'amount' and 'date' fields
        ADD the new dictionary to the list in formatted_transactions

    WITH OPEN 'Your_transactions.json' for writing as file
    WRITE "{" to file
    FOR EACH index, (category, transactions_list) pair in formatted_transactions
      WRITE category to file
      WRITE "[\n" to file
      FOR EACH i, transaction in transactions_list
        WRITE JSON representation of transaction
        IF i is not the last transaction
```

```
            WRITE ",\n" to file
          ELSE
            WRITE "\n" to file
        IF index is not the last category
          WRITE "],\n" to file
        ELSE
          WRITE "]\n" to file
      WRITE "}" to file
    EXCEPT KeyError
      PRINT error message
    EXCEPT IOError or json.JSONDecodeError
      PRINT error message


FUNCTION read_bulk_transactions_from_file()
  TRY
    WITH OPEN 'Your_transactions.txt' for reading as text_file
    INITIALIZE content_added as False

    FOR EACH line in text_file
      REMOVE leading/trailing whitespace from the line
      IF line is not empty
        SPLIT line into category, amount, date
        CONVERT amount to a number
        CREATE transaction dictionary with 'amount' and 'date'

        IF category not in transactions
          CREATE empty list in transactions with the category as key
        IF transaction not in transactions[category]
          ADD transaction to transactions[category]
          INITIALIZE content_added to True

    IF content_added is True
      PRINT success message
      CALL save_transactions()
    ELSE
      PRINT content already exists
```

```
  EXCEPT FileNotFoundError
    PRINT file not found

FUNCTION add_transaction()
  DECLARE transactions
  REPEAT UNTIL TRUE:
    TRY
      GET input from the user for amount
      CONVERT amount to a number
      IF amount is less than or equal to zero
        PRINT error message
      ELSE
        EXIT the loop
    EXCEPT ValueError
      PRINT error message
  GET input from the user for category
  REPEAT UNTIL TRUE
    GET input from the user for date
    IF date is in "YYYY-MM-DD" format
      TRY
        VALIDATE that year, month, and day are numbers
        EXIT the loop
      EXCEPT ValueError
        PRINT error message
    ELSE
      PRINT error message

  CREATE transaction dictionary with 'amount' and 'date'
  IF category is not in transactions
    CREATE new empty list within transactions with the category as key
  ADD transaction to transactions[category]
  PRINT transaction added message

FUNCTION view_transactions()
  DECLARE transactions
  IF transactions is empty
```

```
    PRINT no transactions
  ELSE
    FOR EACH category, category_transactions pair in transactions
      PRINT category
      FOR EACH index, transaction in category_transactions (starting index is 1)
        GET 'amount' from transaction (or "-" if not found)
        GET 'date' from transaction (or "-" if not found)
        PRINT index, amount, and date


FUNCTION update_transaction()
  CALL view_transactions()

  REPEAT UNTIL TRUE:
    IF transactions is empty
      PRINT no transactions
      BREAK

    GET input from the user for category (convert to lowercase)
    IF category is in transactions
      GET transactions_list from transactions[category]
      IF transactions_list is not empty
        TRY
          GET input from the user for transaction index (subtract 1 for array indexing)
          IF index is valid (between 0 and length of transactions_list - 1)
            DISPLAY current transaction details

            REPEAT UNTIL TRUE
              TRY
                GET input from the user for new amount
                CONVERT amount to a number
                BRESK
              EXCEPT ValueError
                PRINT error message

            REPEAT UNTIL TRUE:
              GET input from the user for new date
```

```
            IF date is in "YYYY-MM-DD" format
              TRY
                IF that year, month, and day are numbers
                BREAK
              EXCEPT ValueError
                PRINT error message
            ELSE
              PRINT error message

          GET input from the user for update confirmation
          IF confirmation is 'yes'
            UPDATE the transaction in transactions_list
            CALL save_transactions()
            PRINT saved successfully
            BREAK
          ELSE
            PRINT cancel
        ELSE
          PRINT invalid index
      EXCEPT ValueError
        PRINT invalid input
    ELSE
      PRINT no transactions message for the category
  ELSE
    PRINT category not found

FUNCTION delete_transaction()
  DECLARE transactions

  REPEAT UNTIL TRUE:
    IF transactions is empty
      PRINT no transactions message
      BREAK

    CALL view_transactions()
    GET input from the user for category (convert to lowercase)
    IF category is in transactions
```

```
        GET transactions_list from transactions[category]
        IF transactions_list is not empty
          TRY
            GET input from the user for transaction index (subtract 1 for array indexing)
            IF index is valid (between 0 and length of transactions_list - 1)
              GET input from the user for delete confirmation (convert to lowercase)
              IF confirmation is 'yes'
                DELETE the transaction from transactions_list
                CALL save_transactions()
                PRINT successfully saved
                BREAK
              ELSE
                PRINT cancellation message
                BREAK
            ELSE
              PRINT invalid index
          EXCEPT ValueError
            PRINT invalid input
        ELSE
          PRINT no transactions message for the category
      ELSE
        PRINT category not found



FUNCTION display_summary()
  CREATE an empty dictionary called category_expenses

  FOR EACH category, transactions_list pair in transactions
    INITIALIZE total_expense to 0

    FOR EACH transaction in transactions_list
      GET 'amount' from transaction
      IF amount is greater than 0
        ADD amount to total_expense

    STORE total_expense in category_expenses with the category as key
```

```
  PRINT expenses summary header
  FOR EACH category, expense pair in category_expenses
    PRINT category and expense with formatting

FUNCTION main_menu()
  CALL load_transactions()
  CLEAR transactions dictionary

  REPEAT UNTIL TRUE:
    PRINT Personal Finance Tracker
    PRINT("Personal Finance Tracker")
        PRINT("Main Menu")
         PRINT("1. Add a Transaction")
        PRINT("2. View all Transactions")
        PRINT("3. Update a Transaction")
        PRINT("4. Delete a Transaction")
        PRINT("5. Display the expenses Summary")
        PRINT("6. Get Transactions from the text file")
        PRINT("7. Exit")

    TRY
      GET input from the user for choice
      CONVERT choice to a number
    EXCEPT ValueError
      PRINT error message
      CONTINUE

    IF choice is 1
      CALL add_transaction()
    ELSE IF choice is 2
      CALL view_transactions()
    ELSE IF choice is 3
      CALL update_transaction()
    ELSE IF choice is 4
      CALL delete_transaction()
    ELSE IF choice is 5
      CALL display_summary()
```

```
    ELSE IF choice is 6
      CALL read_bulk_transactions_from_file()
    ELSE IF choice is 7
      CALL save_transactions()
      PRINT saving confirmation
      PRINT exiting from the program
      EXIT
    ELSE
      PRINT invalid choice message


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

PYTHON CODE

```
import json

# Global dictionary to store all the transactions
transactions = {}


#Function to load transactions
def load_transactions():
  global transactions
  try:
    with open('Your_transactions.json', 'r') as file:
      file_content = file.read()
      if file_content.strip():
        transactions = json.loads(file_content)
        #Reads all the content of JSON file and parses it to a Python dictionary and stores it in the transactions variable
      else:
        transactions = {}
        #Initialize a empty dictionary if the file is empty
```

```
  except (FileNotFoundError):
    transactions = {}
    #Initialize as empty dictionary on error


#Function to save the user enter data and the text file data
def save_transactions():
  global transactions
  try:
    #Convert transactions to the correct format before saving to JSON file
    formatted_transactions = {}
    for category, category_transactions in transactions.items():
      #Loops through every key value pair in the transactions dictionary
      formatted_transactions[category] = [
        {"amount": transaction.get("amount", "-"), "date": transaction.get("date", "-")}
        #Gets the "amount" if it exists in the transaction dictionary,otherwise assigns "-" to that
        for transaction in category_transactions
      ]

    #Save the formatted transactions to the JSON file in the correct format
    with open('Your_transactions.json', 'w') as file:
      file.write("{\n")
      for index, (category, transactions_list) in enumerate(formatted_transactions.items()):
        file.write(f'  "{category}": [\n')

       #Loops over each transaction in the transactions_list which relevant with the category
        for i, transaction in enumerate(transactions_list):
      #Write each transaction with JSON formatting and a newline
          file.write(f'   {json.dumps(transaction, separators=(", ", ": "))}')
                  #json.dumps converts python dictionary transaction into a Json formatted
string

          #Add a comma and a newline after every transaction except the last
          if i != len(transactions_list) - 1:
            file.write(",\n")
          else:
            file.write("\n")
```

```
        #If index is not the last category,it then adds a comma and a new line to separate from the next category
            #If not add a new line only
            if index != len(transactions) - 1:
                file.write("  ],\n")
            else:
                file.write("  ]\n")
        file.write("}\n")
        #Writes a "}" to closing the file

    except KeyError:
        print("Key 'amount' not found in transaction.")
        #I/O error, might occurs if there's a problem when opening,writing or closing the JSON file.
        #"as e" stores the actual error object in a variable called e.Then the code can print the error message with the type of error.
    except (IOError, json.JSONDecodeError) as e:
        print(f"Error saving transactions: {e}")



#Reads transactions from text file and adds to the data.
def read_bulk_transactions_from_file():
    try:
        with open('Your_transactions.txt', 'r') as text_file:
            content_added = False
            #To check if new content is added or not to transactions after checking the file
            for line in text_file:
                line = line.strip()
                 #line.strip removes whitespace characters in that line
                if line:
                    category, amount, date = line.split(',')
                    amount = float(amount)
                    transaction = {"amount": amount, "date": date}

                    if category not in transactions:
                        transactions[category] = []
                    if transaction not in transactions[category]:
                        transactions[category].append(transaction)
```

```
                content_added = True
                #If the content_added is True after checking all lines, it means new content is added from the file to transactions

    if content_added:
      print()
      print("Text file content added successfully!")
      save_transactions()
      # Save transactionss after updating
    else:
      #If the content_added remains False after checking all lines, it means no new content is added from the file to transactions
      print()
      print("Text file content already exists.")

  except FileNotFoundError:
    print("File 'Your_transactions.txt' not found.")
    pass



#Function to Add a transaction(user input)
def add_transaction():
  global transactions
  while True:
    try:
      trans_amount = float(input("Enter the amount: "))
      if trans_amount <= 0:
        print("Amount must be greater than zero. Please try again!")
        continue
      break
    except ValueError:
      print("Invalid amount. Please try again!")

  trans_category = input("Enter category: ")
  while True:
    trans_date = input("Enter date (YYYY-MM-DD): ")
    if len(trans_date) == 10 and trans_date[4] == '-' and trans_date[7] == '-':
```

```
        try:
            # Checking if the year, month, date is a number with the index and converts it to an
integer
            int(trans_date[:4])
            int(trans_date[5:7])
            int(trans_date[8:])
            break
        except ValueError:
            print("Invalid date. Please enter a valid date in this format YYYY-MM-DD")
    else:
        print("Invalid date. Please enter a valid date in this format YYYY-MM-DD")


  transaction = {"amount": trans_amount, "date": trans_date}
  if trans_category not in transactions:
    transactions[trans_category] = []
    '''
  Checking if the category of the transaction already exists in the transactions
 dictionary.If it is not there it creates a empty list using new category and appends the content to
that
     '''
  transactions[trans_category].append(transaction)
  print("Transaction added successfully.")



#Function to view the existing transactions
def view_transactions():
  global transactions
  if not transactions:
    # Checking if the transactions dictionary is empty, if its empty it displays a message
    print()
    print("No transactions available. Please add transactions first")
  else:
    for category, category_transactions in transactions.items():
      print(f"Category: {category}")
      for index, transaction in enumerate(category_transactions, start=1):
        # enumerate function iterates over the list of the transactions
        #and gives pairs of items in the list and starts from index 1 not 0
```

```
            # and prints transaction with the index for our reference

            amount = transaction.get("amount", "-")
            date = transaction.get("date", "-")
            # used get method to get the wanted key from the dictionary and print it

            print(f"{index}. amount: {amount}, date: {date}")
            print()



#Function to update an existing transaction
def update_transaction():
  view_transactions()

  while True:
    # Loops until user enters a valid transaction details to updated or user choose to cancel
    if not transactions:
      print("No transactions available.")
      return

    category = input("Enter category of the transaction to update: ").lower()
    if category in transactions:
      transactions_list = transactions[category]
      if transactions_list:
        try:
          index = int(input("Enter the transaction index you want to update: ")) - 1
          if 0 <= index < len(transactions_list):
            # Display selected transaction details for user
            transaction = transactions_list[index]
            print(f"Current transaction: Amount: {transaction['amount']}, Date:
{transaction['date']}")

            while True:
              try:
                trans_amount = float(input("Enter the new amount: "))
                break
```

```
                except ValueError:
                    print("Invalid amount. Please try again!")

                while True:
                    trans_date = input("Enter the new date (YYYY-MM-DD): ")
                    if len(trans_date) == 10 and trans_date[4] == '-' and trans_date[7] == '-':
                        try:
                            int(trans_date[:4])
                            int(trans_date[5:7])
                            int(trans_date[8:])
                            break
                        except ValueError:
                            print("Invalid date. Please enter a valid date in this format YYYY-MM-DD")
                    else:
                        print("Invalid date. Please enter a valid date in this format YYYY-MM-DD")

                confirm_update = input("Are you sure you want to update? (yes/no): ").lower()
                if confirm_update == 'yes':
                    transactions_list[index] = {"amount": trans_amount, "date": trans_date}
                    save_transactions()
                    print("Transaction updated successfully!")
                    break
                # Exit the loop after updating the transaction
                else:
                    print("Update cancelled successfully!")
            else:
                print("Invalid transaction index.")
        except ValueError:
            print("Invalid input. Please enter a valid number!")
    else:
        print("No transactions available for this category.")
else:
    print("Category not found.")


# function to delete a transaction
def delete_transaction():
```

```
global transactions

while True:
  # Loops until a user enter correct transaction details and confirms to delete or cancel
  if not transactions:
    print("No transactions available. Please add transactions first.")
    return
  view_transactions()

  category = input("Enter category of the transaction to delete: ").lower()
  if category in transactions:
    transactions_list = transactions[category]
    #checks if the category exist in the dictionary as a key and get the details if exists
    if transactions_list:
      try:
        index = int(input("Enter the transaction index you want to delete: ")) - 1
        if 0 <= index < len(transactions_list):
          transaction = transactions_list[index]
          #checks the user input index is valid and get the details of that transaction to delete

          confirm_del = input("Are you sure you want to delete? (yes/no): ").lower()
          if confirm_del == 'yes':
            del transactions_list[index]
            save_transactions()
            print("Transaction deleted successfully!!")
            return
          # Exit and go back if delete is successful
          else:
            print("Cancelled successfully!!")
            return
        else:
          print("Invalid transaction index.")
      except ValueError:
        print("Invalid input. Please enter a valid number.")
    else:
      print("No transactions available for this category.")
  else:
```

```
        print("Category not found.")



#Function to display a summary of expenses
def display_summary():
    category_expenses = {}
    # A dictionary to store expenses by categories

    for category, transactions_list in transactions.items():
        total_expense = 0
        for transaction in transactions_list:
            amount = transaction.get("amount")
            if amount > 0:
                total_expense += amount

            ''' check each transactions in the list and check the transactions
                of the particular category and adds all the amounts to display'''

        category_expenses[category] = total_expense
    print()
    print("Your Expenses Summary:")
    for category, expense in category_expenses.items():
        print()
        print(f"You have spent a total amount of  Rs.{expense} for the {category}  ")



def main_menu():
    load_transactions()
    transactions.clear()
    #Clears content in the JSON file so when user re run it the old data will be cleared
    while True:
        print()
        print("  Personal Finance Tracker")
        print("       Main Menu")
        print()
        print("1. Add a Transaction")
```

```
    print("2. View all Transactions")
    print("3. Update a Transaction")
    print("4. Delete a Transaction")
    print("5. Display the expenses Summary")
    print("6. Get Transactions from the text file")
    print("7. Exit")
    try:
      print()
      choice = int(input("Enter your choice (1-7): "))
    except ValueError:
      print("Invalid choice. Please enter a number.")
      #If a user inserts anything other than a number it displays a message and ask user to enter again
      continue
    if choice == 1:
      add_transaction()
    elif choice == 2:
      view_transactions()
    elif choice == 3:
      update_transaction()
    elif choice == 4:
      delete_transaction()
    elif choice == 5:
      display_summary()
    elif choice == 6:
      read_bulk_transactions_from_file()
    elif choice == 7:
      save_transactions()
      print("All Transactions are saved to JSON file.")
      print("Exiting from the program...")
      break
    else:
      print("Invalid choice. Please choose a number from 1 to 7.")
    #If a user inserts anything other than a number between 1-7 it displays a message and ask user to enter again

if __name__ == "__main__":
```

main_menu()