tracker/expense_tracker.py

```python
# functions for adding, viewing, budgeting, saving, and loading expenses

import csv
from datetime import datetime
import os
import re

expenses = []  # List to store expenses
budget = 0  # Variable to store budget


# Function to get the path to the data file
def get_data_file_path():
    current_dir = os.path.dirname(__file__)
    return os.path.join(current_dir, "data", "expenses.csv")


# Function to add an expense
# This function will be called to add an expense to the list.
# Each expense is represented as a dictionary with keys: date, category, amount,
and description.
# It will validate the input using the validate_expense_data function.
# If the input is valid, it will append the expense to the expenses list.
# If the input is invalid, it will raise a ValueError.
# The function will also handle the case where the date is not provided by using
the current date.
# It will format the date to YYYY-MM-DD format.
def add_expense(amount, category, description, date=None):
    if date is None or date == "":
        date = datetime.now().strftime("%Y-%m-%d")

    # Validate the expense data
    if validate_expense_data(amount, category, description, date):
        expenses.append(
            {
                "date": date,
                "category": category,
                "amount": amount,
                "description": description,
            }
        )
        print(
            f"Added expense: {amount:.2f} in {category} on {date} with description
'{description}'"
        )
    return
```

```
# Function to view all expenses
# This function will be called to display all the expenses in the list.
# It will sort the expenses by date in descending order (most recent first).
# It will iterate through the expenses list and print each expense in a formatted
manner.
# It will also validate the expense data before displaying it.
# If the data is invalid, it will print an error message and skip that entry.
def view_expenses():
    if not expenses:
        print("No expenses found.")
        return
    print("Expenses:")

    # Validate the expense data
    for expense in expenses:
        if not validate_expense_data(
            expense["amount"],
            expense["category"],
            expense["description"],
            expense["date"],
            entry=False,
        ):
            if len(expenses) == 1:
                print("Invalid expense data found. Please check your entries.")
                return
            print("Invalid expense entry found. Skipping invalid entry.")
            return

        print(
            f"-{expense['amount']: .2f} in {expense['category']} on
{expense['date']} with description '{expense['description']}'"
        )


# Function to validate the expense data
# This function will be called to validate the expense data before adding it to
the list and before displaying it
# Depending on the entry parameter, it will either raise a ValueError or return
False if the data is invalid.
def validate_expense_data(amount, category, description, date, entry=True):
    # Date validation (YYYY-MM-DD)
    if not re.match(r"^\d{4}-\d{2}-\d{2}$", date):
        if entry:
            raise ValueError("Date must be in YYYY-MM-DD format.")
        return False

    # Amount should be a positive number
    if not isinstance(amount, (int, float)) or amount <= 0:
        if entry:
            raise ValueError("Amount must be a positive number.")
        return False

    # Category should not be empty
    if not isinstance(category, str) or not category.strip():
        if entry:
            raise ValueError("Category must be a non-empty string.")
        return False

    # Description should not be empty
    if not isinstance(description, str) or not description.strip():
        if entry:
            raise ValueError("Description must be a non-empty string.")
        return False

    return True


# Function to set monthly budget
# This function will be called to set the monthly budget.
# It will validate the input to ensure it is a positive number.
# If the input is invalid, it will raise a ValueError.
# If the input is valid, it will set the budget variable to the specified amount.
def set_budget(amount):
    global budget
    try:
        amount = float(amount)
    except ValueError:
        budget = 0
        raise ValueError("Budget must be a positive number.")
    if not isinstance(amount, (int, float)) or amount <= 0:
        raise ValueError("Budget must be a positive number.")
    budget = amount
    print(f"Monthly budget set to: {budget:.2f}")


# Function to compare expenses with monthly budget
# This function will be called to check if the expenses exceed the budget.
# It will calculate the total expenses for the current month and compare it with
the budget.
# If the expenses exceed the budget, it will print a message indicating the excess
amount.
# If the expenses are within the budget, it will print a message indicating the
remaining budget.
def compare_budget():
    total_expenses = get_current_month_expenses()
    if budget == 0:
        print("No budget set.")
        return
    if total_expenses > budget:
        print(f"Expenses exceeded the budget by: {total_expenses - budget:.2f}")
    else:
        print(
            f"Expenses are within the budget. Remaining budget for the month:
{budget - total_expenses:.2f}"
        )


# Function to get total expenses for the current month
# This function will be used to calculate the total expenses for the current
month.
def get_current_month_expenses():
    current_month = datetime.now().month
    current_year = datetime.now().year
    return get_expenses_for_month(current_month, current_year)


# Function to get the total expenses for a specific month and year
# This function will be used to calculate the total expenses for a specific month
and year.
# It will filter the expenses list based on the month and year provided.
# It will return the total amount spent in that month.
def get_expenses_for_month(month, year):
    month_expenses = [
        expense
        for expense in expenses
        if datetime.strptime(expense["date"], "%Y-%m-%d").month == month
        and datetime.strptime(expense["date"], "%Y-%m-%d").year == year
    ]
    total = sum(expense["amount"] for expense in month_expenses)
    return total


# Function to load expenses from a CSV file
# This function will be called at the start of the program to load any previously
saved expenses.
# It will also clear the existing expenses list to avoid duplicates.
# Gracefully handle the case where the data is corrupt by using a try-except
block.
# If the file is not found, it will print a message and continue without raising
an error.
def load_expenses_from_csv():
    print("Loading expenses...")
    expenses.clear()  # Clear the existing expenses list
    try:
        with open(get_data_file_path(), mode="r") as file:
            reader = csv.DictReader(file)
            for row in reader:
                try:
                    expenses.append(
                        {
                            "date": row["date"],
                            "category": row["category"],
                            "amount": float(row["amount"]),
                            "description": row["description"],
                        }
                    )
                except ValueError as e:
                    print(f"Error loading expense: {e}")
    except FileNotFoundError:
        print("No previous expenses found. Starting fresh.")
        pass
```

```python
# Function to save expenses to a CSV file
# This function will be called when the user chooses to save expenses.
# It will overwrite the existing file with the current expenses.
def save_expenses_to_csv():
    with open(get_data_file_path(), mode="w", newline="") as file:
        fieldnames = ["date", "category", "amount", "description"]
        writer = csv.DictWriter(file, fieldnames=fieldnames)

        writer.writeheader()
        for expense in expenses:
            writer.writerow(expense)
    print("Expenses saved.")
```