个人记账系统

一、 项目介绍

1.1 项目概况

1.1.1 项目背景

本项目旨在开发一款基于Python的桌面个人记账系统，满足用户日常财务管理需求，提升生活和工作中的财务规划能力。

1.1.2 功能概览

主要功能包括记录每日支出、按类别统计、生成图表，并支持上传到GitHub云端备份及分享，同时具备图形化界面，方便用户操作。



二、 技术架构与关键模块

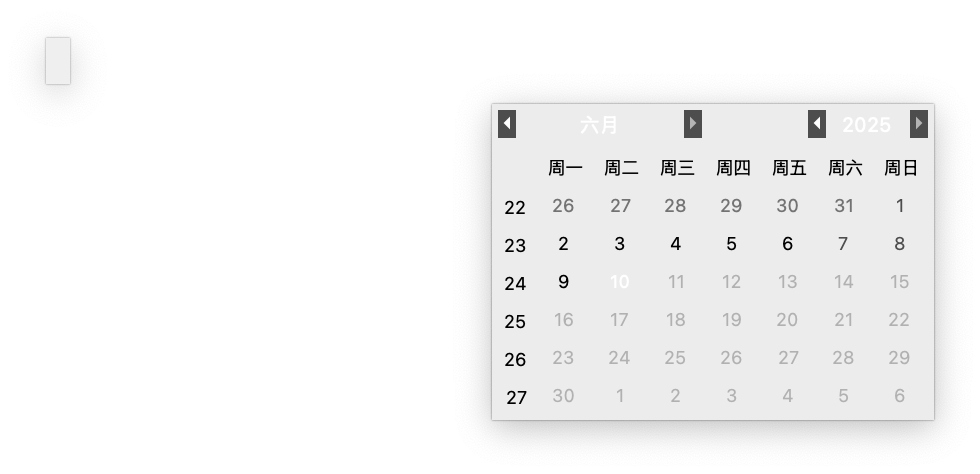
2.1 前端界面

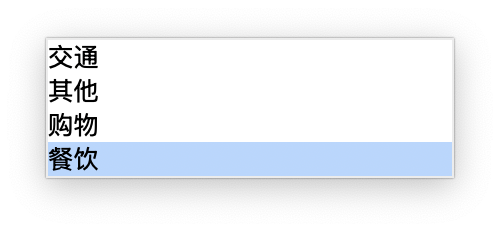
2.1.1 界面技术选型

使用Tkinter、ttk和tkcalendar构建前端界面，包含日期选择器、输入框、下拉菜单、按钮和表格，实现简洁易用的交互体验。

2.1.2 界面布局与功能

界面布局合理，各功能模块分区明确，用户可通过日期选择器快速定位，输入框和下拉菜单方便录入数据，表格清晰展示支出记录。

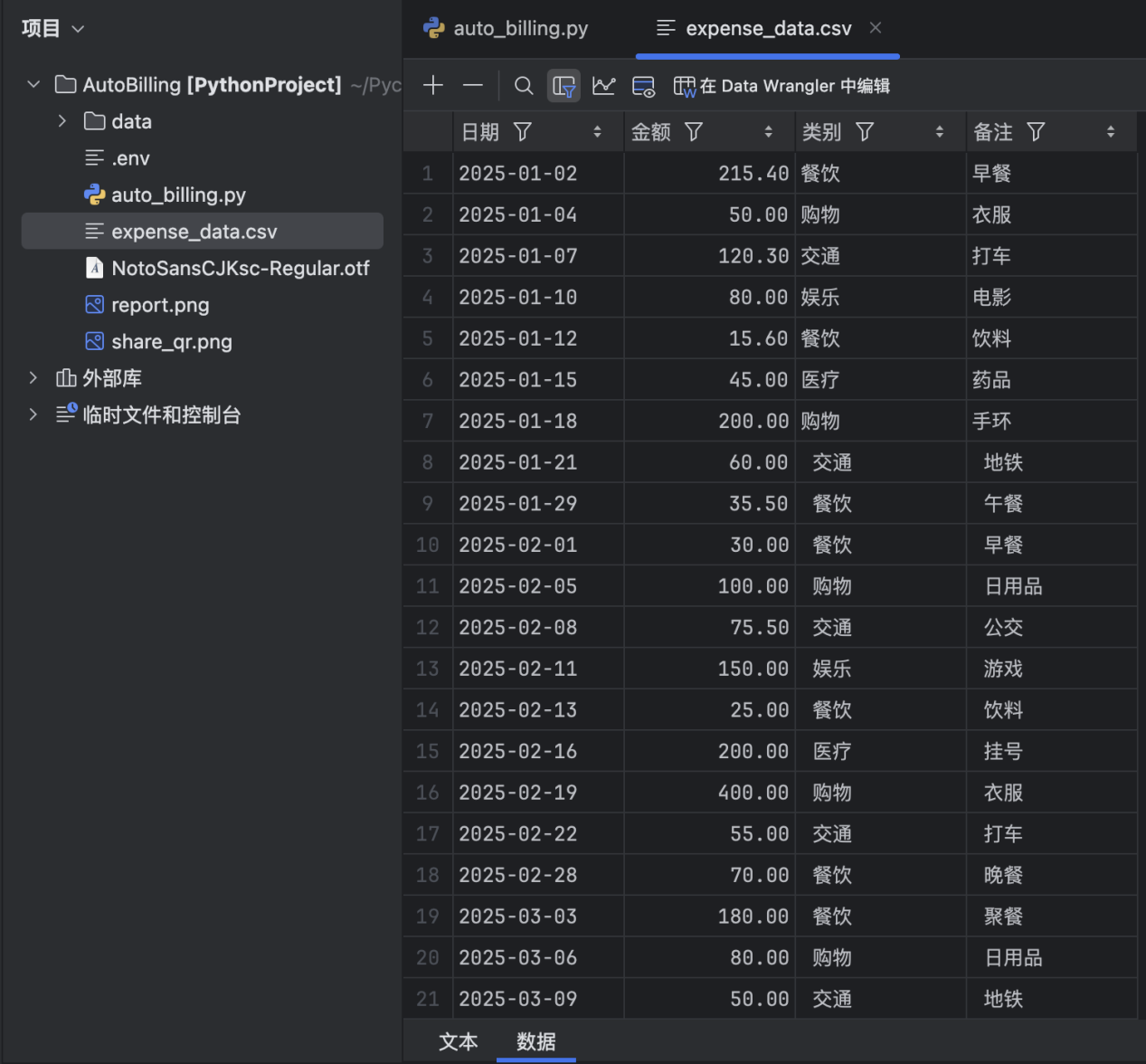




2.2 数据存储

2.2.1 存储方案

采用本地CSV文件存储数据，结构简单，易于读写和维护，适合单用户本地使用场景，保障数据存储的高效性和稳定性。



2.2.2 数据读写流程

数据读取时逐行解析CSV文件，写入时按既定格式追加记录，确保数据完整性和一致性，支持快速读写操作。

2.3 数据处理与统计

2.3.1 处理技术

借助Python标准库和collections.defaultdict进行数据处理与统计，实现支出数据的高效分类和汇总计算。

2.3.2 统计功能实现

按类别统计支出金额，计算总支出，为生成图表提供准确数据，支持灵活的统计维度扩展。

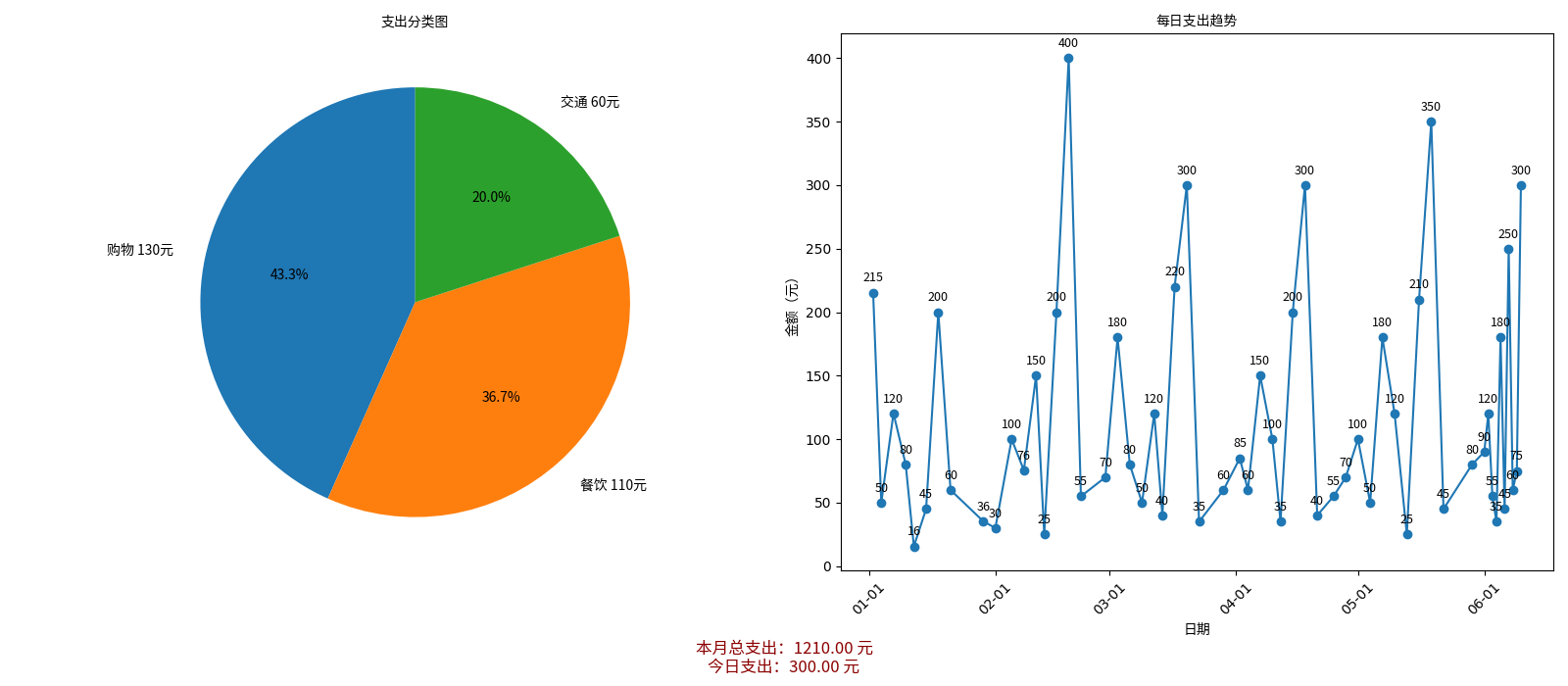
2.4 图表绘制

2.4.1 绘图工具

使用matplotlib绘制饼图和折线图，直观展示支出类别比例和每日支出趋势，帮助用户快速了解财务状况。

2.4.2 图表样式与交互

图表具备清晰的标题、图例和坐标轴标签，采用默认样式，满足基本可视化需求，暂不支持交互式操作。



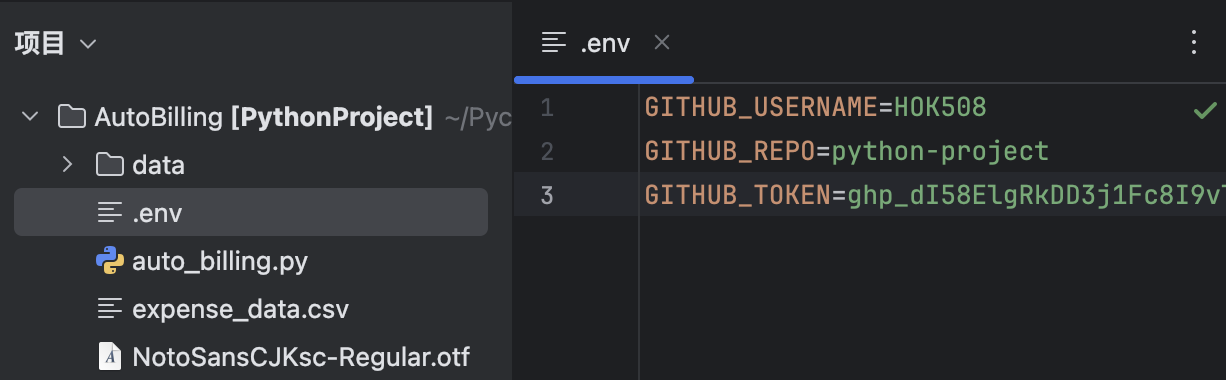
2.5 云端同步

2.5.1 同步技术

利用GitHub内容API和requests实现云端同步，通过Base64编码上传文件，支持更新已存在文件，保障数据备份和分享功能。

2.5.2 同步流程与安全

同步时先检查文件是否存在，获取SHA值判断是否更新，使用环境变量保护Token，确保同步过程安全可靠。



2.6 二维码生成

2.6.1 生成技术

使用qrcode库生成二维码，将图表访问链接转换为二维码图片，方便用户手机扫码查看云端图表，拓展系统功能。

2.6.2 二维码应用

二维码生成后可直接显示在界面上，用户点击即可保存图片，实现快速分享和便捷访问。



三、 核心功能演示

3.1 支出录入

3.1.1 录入流程

用户通过日期选择器选择日期，输入金额和类别，支持自定义类别，点击“添加”按钮完成记录，系统自动保存数据。

3.1.2 类别管理

系统预设常见类别，用户可随时添加新类别，新类别实时添加到下拉菜单中，保证录入灵活性。



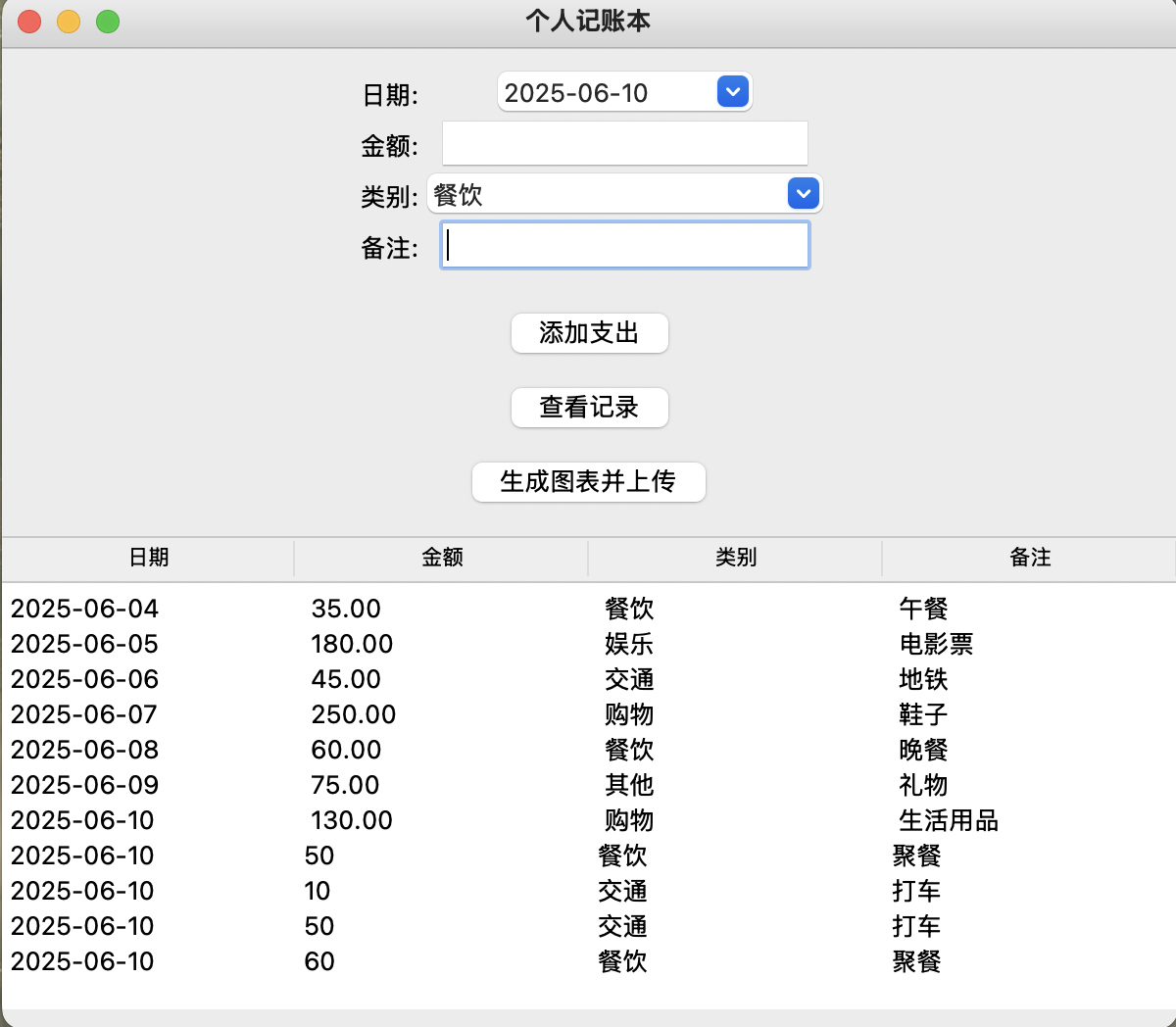
3.2 历史数据查看

3.2.1 数据展示

使用文本展示所有支出记录，包含日期、金额、类别等信息

3.2.2 分页功能

若数据量较大，表格支持分页显示，每页显示固定数量记录，用户可通过翻页按钮浏览不同页面数据。



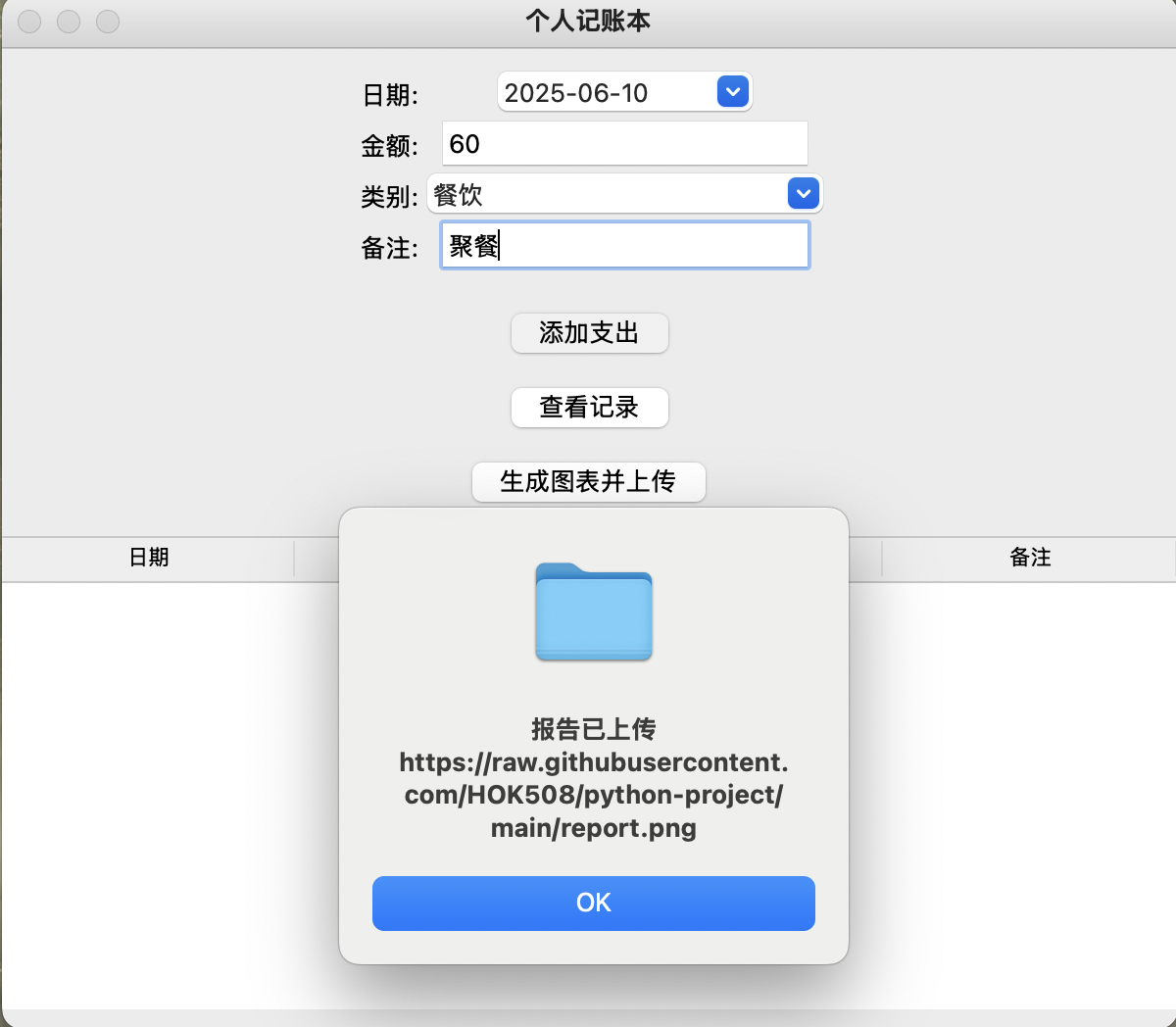
3.3 图表生成与上传

3.3.1 图表生成

点击“生成图表”按钮，系统自动根据历史数据生成饼图和折线图，饼图展示支出类别比例，折线图展示每日支出趋势。

3.3.2 图表上传

图表生成后自动上传至GitHub，系统自动生成访问链接及二维码，用户可点击链接或扫描二维码查看云端图表。



3.4 二维码分享

3.4.1 分享流程

生成的二维码图片可直接显示在项目中，系统自动保存图片到本地，方便用户分享给他人。

3.4.2 二维码应用

二维码支持PNG格式保存，满足用户分享需求。

四、实现难点及解决方案

4.1 中文字体显示问题

4.1.1 问题描述

在使用matplotlib绘制图表时，发现中文字体无法正常显示，导致图表标题和图例等中文内容显示为乱码，影响用户体验。

4.1.2 解决方案

使用matplotlib.font\_manager.FontProperties指定中文字体文件，加载系统中已安装的中文字体，确保图表中文内容正常显示。

4.2 GitHub文件上传与更新

4.2.1 问题描述

当需要将本地文件上传至GitHub时，若文件已存在，直接上传会导致重复文件产生，浪费存储空间且不利于版本管理。

4.2.2 解决方案

通过GitHub API获取文件SHA值，判断文件是否存在及是否需要更新，若文件存在且内容有变化，则先删除原文件，再上传新文件，避免重复上传。

4.3 自定义类别动态管理

4.3.1 问题描述

系统需要支持用户自定义类别，且新类别要实时添加到下拉菜单中，以满足不同用户的个性化需求，这对类别管理的实时性和灵活性提出了挑战。

4.3.2 解决方案

在用户添加新类别时，系统实时更新类别列表，并将其添加到下拉菜单中，同时将新类别保存到本地配置文件，确保下次启动时依然可用。

4.4 日期控件限制

4.4.1 问题描述

日期选择器默认允许用户选择任意日期，包括未来日期，这可能导致用户输入错误的日期信息，影响数据的准确性和可靠性。

4.4.2 解决方案

利用tkcalendar.DateEntry的属性，限制日期选择器的最大日期为当前日期，防止用户选择未来日期，从而避免日期输入错误。

五、项目优点

5.1 界面简洁友好

5.1.1 界面设计

界面采用简洁风格设计，布局合理，色彩搭配协调，图标清晰易懂，符合用户日常使用习惯，降低学习成本。

5.1.2 用户体验

操作流程简单直观，用户无需复杂培训即可快速上手，完成支出记录、查询和图表查看等操作，提升使用体验。

5.2 功能全面实用

5.2.1 功能覆盖

功能涵盖支出记录、统计、可视化及云端同步等核心需求，满足用户日常财务管理的各个方面，提升财务管理效率。

5.2.2 功能协同

各功能模块相互配合，如支出记录为统计和图表提供数据，云端同步保障数据安全，形成完整功能闭环，发挥最大价值。

5.3 安全性保障

5.3.1 敏感信息保护

使用环境变量保存GitHub Token等敏感信息，避免将敏感信息直接写入代码，降低信息泄露风险，保障用户数据安全。

5.3.2 异常处理

对文件读写和网络请求等关键操作进行异常捕获和处理，当出现错误时及时提示用户，确保程序稳定运行，防止数据丢失。

5.4 代码质量高

5.4.1 结构清晰

代码采用模块化设计，各模块职责明确，相互独立，便于理解和维护，支持后续功能扩展和优化。

5.4.2 注释完整

代码中添加了详细的注释，对关键变量、函数和逻辑进行解释说明，方便其他开发者阅读和理解代码，降低维护成本。

六、不足与改进方向

6.1 数据存储局限性

6.1.1 现状分析

当前使用CSV文件存储数据，在数据安全性和并发访问能力方面存在不足，如文件损坏可能导致数据丢失，多用户同时操作时易出现冲突。

6.1.2 改进方向

考虑迁移到关系型数据库如SQLite，提升数据存储的安全性、完整性和并发处理能力，支持更复杂的数据查询和分析。

6.2 界面现代化不足

6.2.1 现状分析

界面基于Tkinter构建，整体风格较为传统，与现代桌面应用的视觉效果存在一定差距，可能影响用户体验。

6.2.2 改进方向

可尝试引入更现代化的界面框架如PyQt，优化界面布局、控件样式和交互效果，打造更具吸引力和专业感的用户界面。

6.3 功能拓展性不足

6.3.1 现状分析

系统目前主要聚焦于支出记录和统计，缺少收入管理、预算提醒等重要财务管理功能，功能拓展性有待提升。

6.3.2 改进方向

增加收入管理模块，支持记录收入来源和金额；添加预算提醒功能，用户可设置预算限额，系统自动提醒超支情况，完善财务管理功能。

6.4 图表交互性不足

6.4.1 现状分析

图表为静态图片，用户无法进行交互操作，如缩放、筛选、查看详情等，限制了用户对数据的深入分析和探索。

6.4.2 改进方向

采用支持交互的图表库如Plotly，使图表具备动态交互功能，用户可通过鼠标操作获取更多数据细节，提升数据可视化效果。