**Openfga仓库数据挖掘与可视化**

**提交、PR、Bug与贡献者活动的多维分析**

姓名： 马钰童 班级：软件2206学号： 20222241121

姓名： 郭思萌 班级：软件2201学号： 20222241468

姓名： 洪 蕙 班级：软件2206学号： 20222241342

姓名： 常 达 班级：软件2206学号： 20222241308

大连理工大学

Dalian University of Technology

目录

[一.需求分析 3](#_Toc188870877)

[1.1需求背景 3](#_Toc188870878)

[1.2功能需求 3](#_Toc188870879)

[1.3数据收集和处理 3](#_Toc188870880)

[1.4统计指标 4](#_Toc188870881)

[1.5数据可视化 4](#_Toc188870882)

[二、设计 4](#_Toc188870883)

[2.1 数据收集与处理 4](#_Toc188870884)

[2.2 数据分析与统计 4](#_Toc188870885)

[2.3 数据可视化 5](#_Toc188870886)

[三、实现 5](#_Toc188870887)

[3.1分析项目的提交历史 5](#_Toc188870888)

[3.1.1爬取数据 5](#_Toc188870889)

[3.1.2可视化数据 6](#_Toc188870890)

[3.2 分析仓库的Bug报告 8](#_Toc188870891)

[3.2.1爬取数据 8](#_Toc188870892)

[3.2.2可视化数据 8](#_Toc188870893)

[3.3 Contributor报告分析 10](#_Toc188870894)

[3.3.1爬取数据介绍 10](#_Toc188870895)

[3.3.2 可视化介绍 11](#_Toc188870896)

[3.4 Request报告分析 13](#_Toc188870897)

[3.4.1爬取数据介绍 13](#_Toc188870898)

[3.4.2 可视化介绍 13](#_Toc188870899)

[3.5 Code Review 报告分析 17](#_Toc188870900)

[3.5.1 爬取数据介绍 17](#_Toc188870901)

[3.5.2 可视化介绍 18](#_Toc188870902)

[3.6 代码质量分析报告 20](#_Toc188870903)

[3.6.1 数据生成与分析 20](#_Toc188870904)

[3.6.2 数据分析与可视化 20](#_Toc188870905)

[3.7 GitHub Issue 评论分析报告 22](#_Toc188870906)

[3.7.1 数据爬取与分析 22](#_Toc188870907)

[3.7.2 数据分析与可视化 23](#_Toc188870908)

[四、工作总结 24](#_Toc188870909)

# 

# 一.需求分析

如题目所述，我们小组准备对开源软件仓库GitHub上的openFGA项目进行多维度的数据挖掘与可视化分析。我们将重点分析提交（Commits）、拉取请求（PRs）、Bug报告和贡献者的活动数据，通过对这些数据的统计分析和可视化展示，为开发团队提供关于项目进展、问题管理、开发效率和贡献者活跃度的有价值洞察。

## **1.1需求背景**

在开源项目的开发过程中，提交记录、拉取请求、Bug报告和贡献者活动数据是反映项目健康状态、开发进度和社区活跃度的重要指标。这些数据能够帮助开发团队识别项目中的瓶颈，评估开发效率，优化开发资源的分配，并促进社区的可持续发展。通过对openFGA项目中的这些数据进行挖掘与可视化分析，能够深入了解项目的动态变化，为决策提供支持，进一步提升项目的管理效率与软件质量。

## **1.2功能需求**

本次分析将围绕OpenFGA项目在GitHub上的多维数据展开，重点对以下几个方面进行统计与分析：

**提交（Commits）**：统计OpenFGA项目每月的提交数量，并分析不同时间段内的提交活跃度，识别活跃和低谷期。

**拉取请求（PRs）**：分析每个拉取请求的提交时间、合并时间，计算PR的生命周期，评估贡献者的提交频率和PR合并效率。

**Bug报告**：对提交的Bug进行数量统计，分析Bug解决的时间跨度，识别常见的Bug类型及其对项目开发的影响。

**贡献者活动**：分析贡献者的提交频率、PR提交量和Bug修复数量，评估各个贡献者的活跃度，并通过可视化方式呈现贡献者的活动分布。

## **1.3数据收集和处理**

为完成数据收集工作，我们将使用GitHub API对OpenFGA项目的数据进行爬取。数据包括但不限于提交记录、PR记录、Bug报告及相关的贡献者活动。爬取的数据将包括：提交记录（包括每次提交的时间和提交者）、PR记录（包括PR的创建时间、合并时间、状态）、Bug记录（包括Bug报告的时间、修复时间和状态）、贡献者信息（包括贡献者的GitHub用户名、贡献数量等）等。

爬取的数据将经过清洗与预处理，去除无效或重复信息，并将有效数据存储起来，方便后续的分析。

## **1.4统计指标**

通过数据统计和分析，我们将提取每月提交数量、每月PR数量及其生命周期（从创建到合并）、每月Bug报告数量及解决时长、贡献者活动统计（贡献频次、贡献类型等）等指标。

## **1.5数据可视化**

为了帮助更直观地理解分析结果，我们将采用以下可视化方法：

**柱状图和饼图**：使用matplotlib进行数据的柱状图和饼状图展示，呈现每月提交、PR和Bug的数量，分析其各类标签占比及时间分布。

**贡献者活跃度图**：展示各个贡献者的活动频次和贡献量，帮助评估团队成员的贡献情况和项目协作状态。

# 二、设计

我们小组根据项目需求，将代码撰写分为几个主要模块，包括数据爬取、数据处理与统计分析，以及数据可视化展示。

## **2.1 数据收集与处理**

我们利用GitHub API对OpenFGA项目的数据进行了抓取，主要涉及以下几类数据：

提交记录（Commits）：包括每次提交的时间、提交者信息等。

拉取请求（PRs）：包括PR的创建时间、合并时间、状态等。

Bug报告：包括Bug报告的时间、修复状态等。

贡献者活动数据：统计每个贡献者的提交次数、PR数量等。

通过编写爬虫脚本，我们从OpenFGA项目的GitHub页面中提取了这些数据，并对数据进行了清洗，剔除了无效或重复的条目。最终，将清洗后的数据存储为CSV文件，便于后续分析和可视化。

## **2.2 数据分析与统计**

在完成数据收集后，我们进行了如下分析：

提交记录统计：每月提交次数的分析，识别提交的高峰期和低谷期。

PR记录统计：分析PR的数量和生命周期，评估PR的合并效率。

Bug报告统计：对Bug报告数量和解决时长进行统计，评估Bug的严重性及其对开发进度的影响。

贡献者活动统计：对每个贡献者的提交次数、PR提交量、Bug修复等活动进行统计，评估其活跃度和对项目的贡献。

## **2.3 数据可视化**

为了更直观地呈现数据，我们在数据可视化部分采用了多种方式，包括柱状图、饼状图、折线图和词云等。以贡献者活动数据可视化为例，下面是我们设计的图表和流程：

贡献者贡献分布（饼状图）：展示各个贡献者的贡献分布，帮助我们了解哪些贡献者对项目的贡献最大。代码中使用了plt.pie()来生成饼图，并使用了不同的颜色来区分不同贡献者。

贡献者贡献排名（柱状图）：通过柱状图展示贡献最多的前10个贡献者，以便分析谁是项目的主要贡献者。代码中使用了sns.barplot()来绘制柱状图，并通过排序得到前10名贡献者。

贡献者活动趋势（折线图）：展示每月的贡献者活动趋势。我们将贡献者的活动按月份进行分组，并用折线图展示每月的贡献数量。这有助于我们识别项目开发的活跃期和低谷期。代码中使用了plt.plot()来生成折线图。

贡献者活动时间分布（24小时制）：展示一天内不同时间段的贡献者活动频率。通过对提交时间进行处理，统计每个小时的活动数量。使用seaborn.histplot()绘制时间分布图。

所有这些可视化图表的生成都使用了matplotlib和seaborn库，这些图表不仅有助于更好地理解数据，还能为决策者提供清晰、直观的视觉支持。可视化结果会保存在visualization\_results目录下，生成的图表包括：

contributor\_distribution.png（贡献者贡献分布）

top\_contributors.png（贡献者排名）

monthly\_activity\_trend.png（每月活动趋势）

contributor\_time\_distribution.png（24小时活动时间分布）

# 三、实现

## **3.1分析项目的提交历史**

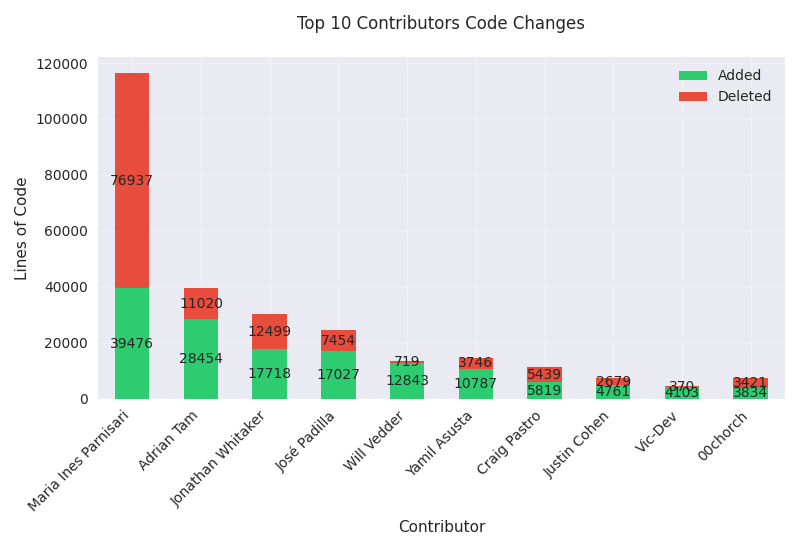
### 3.1.1爬取数据

把原始项目拉到本地D盘test目录之后，通过GitPython库的Repo类获取本地Git仓库信息，然后获取默认分支的所有提交记录。使用pandas处理时间数据和统计信息，将提交信息转换为DataFrame格式，并使用Counter统计作者提交次数，遍历每个提交提取时间、作者、消息等关键信息，同时统计代码变更，即文件数、增删行数信息，并分析提交类型，Bug修复、新功能、文档等，最后将分析结果（包括提交详情、月度统计、作者统计等）保存为CSV文件。最终得到总提交数、作者分布、时间分布等提交统计，文件修改数、代码行增删等代码变更统计，Bug修复、新功能、文档等提交类型统计，以及月度提交趋势、提交时间分布等时间趋势分析，结果也被保存在CSV文件中以便观察分析和后续的可视化。

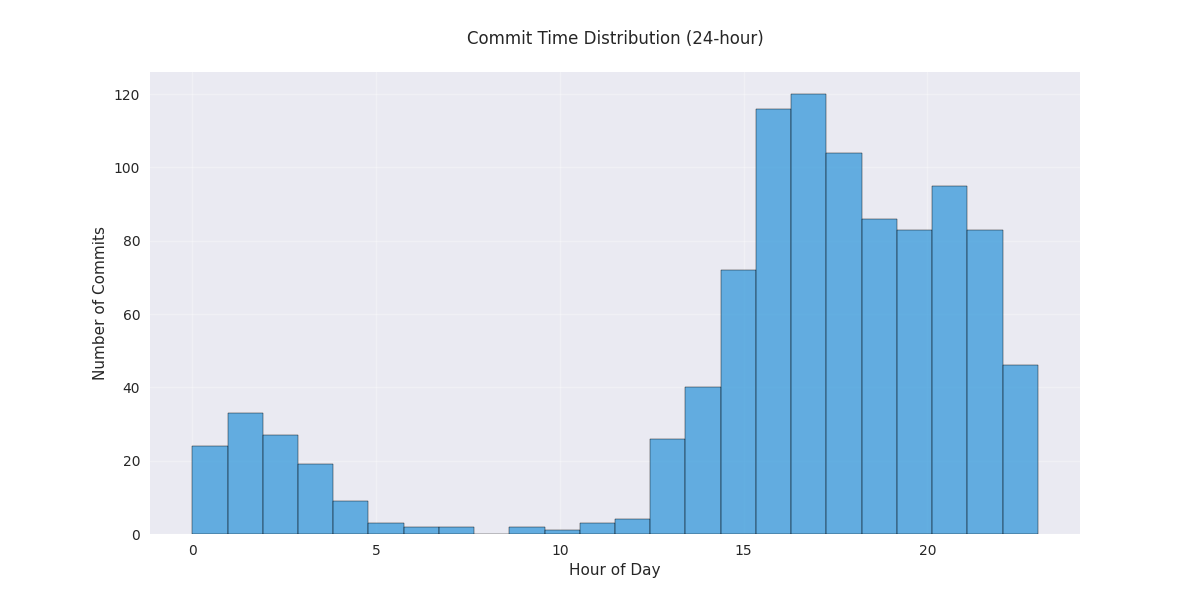
### 3.1.2可视化数据

使用pandas读取上一步的CSV文件数据，通过translate\_chinese\_to\_english函数将中文文本转换为英文，利用matplotlib和seaborn库创建八种不同的可视化图表：贡献者分布饼图、提交类型分布饼图、月度提交趋势线图、提交时间分布直方图、代码变更统计条形图、前10名贡献者活动热力图、前10名贡献者代码变更堆叠柱状图以及月度代码变更趋势线图。通过plt.style.use('seaborn')设置统一的图表样式，使用自定义颜色方案提升可视化效果，并为每个图表添加标题、标签和图例。

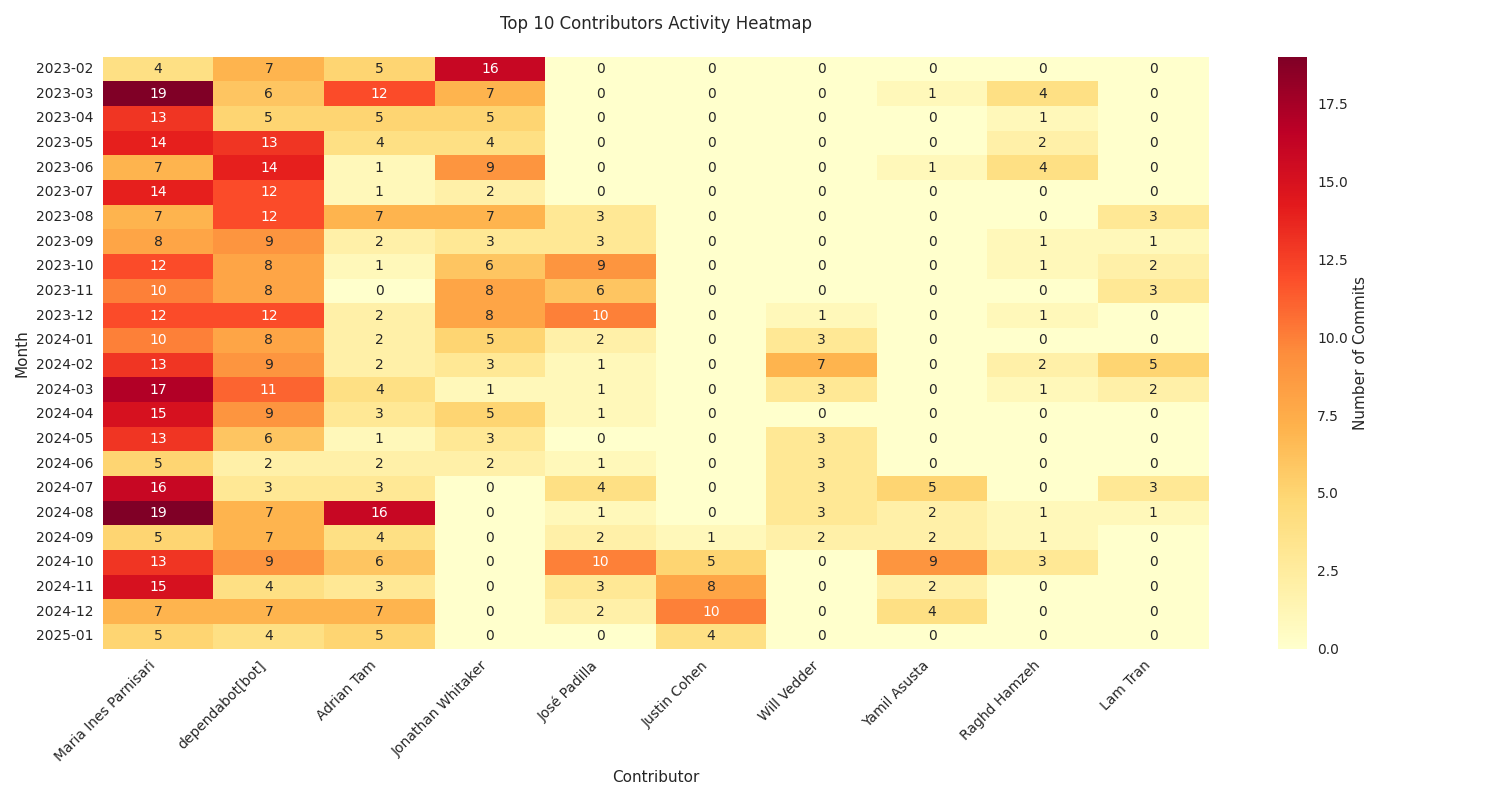
部分结果分析：



从"Top 10贡献者代码变更"堆叠柱状图可以看出，项目代码变更量呈现显著的差异分布：最活跃的贡献者"Maria Ines Parnisari"的代码变更量远超其他人，新增了39,476行代码，删除了76,937行，总变更量超过11万行，显示了大规模的代码重构或优化工作；第二和第三位的"Adrian Tam"和"Jonathan Whitaker"的变更量相对较大，分别有约4万和3万行的总变更量，其中"Adrian Tam"新增28,454行，删除11,020行，"Jonathan Whitaker"新增17,718行，删除12,499行；其余贡献者如"José Padilla"、"Will Vedder"等的代码变更量相对较小，大多在2万行以下，这种分布模式表明项目存在明显的核心开发者，他们承担了大部分的代码改进工作。



从"提交时间分布(24小时)"直方图可以看出，项目代码提交活动呈现明显的双峰分布特征：主要提交高峰出现在下午14-16点，峰值达到约120次提交，显示这是开发者最活跃的工作时段；凌晨0-4点出现一个较小的活动高峰，峰值约30次提交，可能反映了不同时区开发者的参与；而在早上5-13点期间的提交活动明显较少，基本维持在10次以下，这可能是大多数开发者的休息时间；从16点后提交频率逐渐下降，但仍保持相对较高水平，直到深夜23点仍有约40次提交，这种分布模式反映了项目团队跨时区协作的特点，以及开发者倾向于在下午和晚间工作的习惯。



从"Top 10贡献者活动热力图"可以看出，项目贡献者的活动呈现出明显的时间和个人特征：Maria Ines Parnisari作为最活跃的贡献者，在整个时间段内保持稳定的高频贡献，特别是在2023-03和2024-08期间达到峰值（19次提交）；dependabot[bot]作为自动化工具，也显示出较为稳定的活动频率，通常每月有6-14次提交；Adrian Tam的贡献集中在特定时期，如2024-08有16次提交的高峰；Jonathan Whitaker在项目早期（2023-02）较为活跃，有16次提交；而其他贡献者如José Padilla、Justin Cohen等的活动相对分散，呈现出间歇性的参与模式。整体来看，热力图反映了项目既有持续活跃的核心贡献者，也有定期参与的维护者，形成了良好的协作模式。

## **3.2 分析仓库的Bug报告**

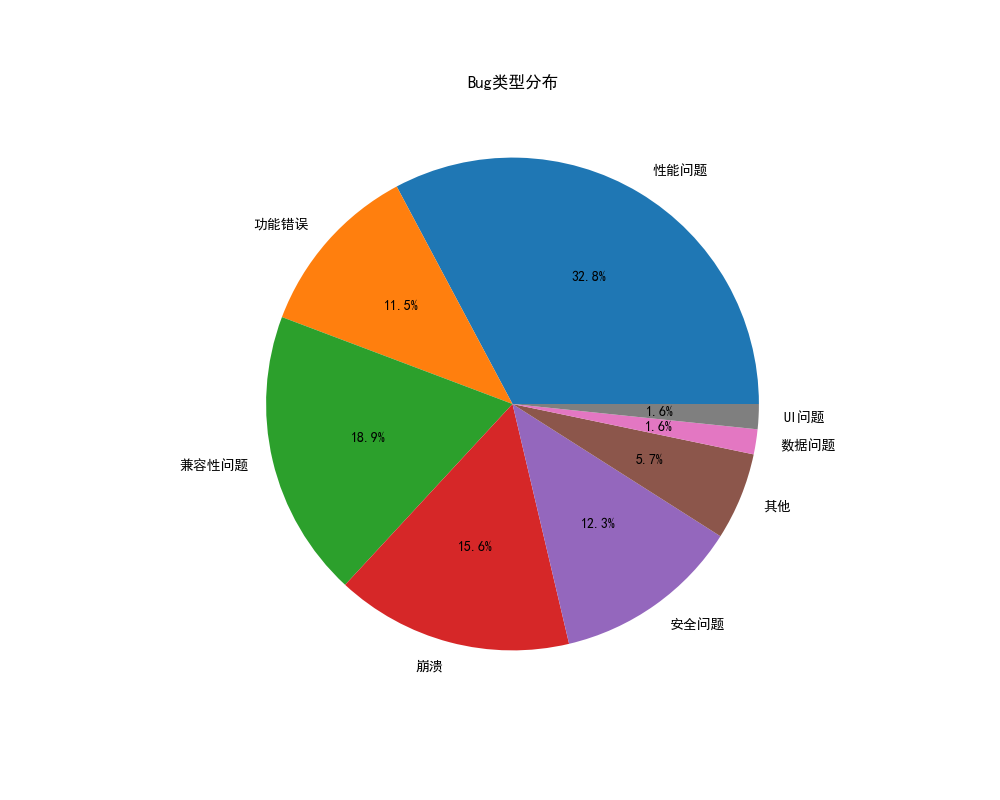
### 3.2.1爬取数据

使用GitHub API获取和分析仓库的bug报告：通过requests库发送API请求获取issues，使用pandas处理数据，对每个bug issue进行详细解析，提取标题、描述、创建时间、状态、标签等信息，通过关键词匹配对bug进行分类（功能错误、性能问题、兼容性问题等）、识别受影响组件（前端、后端、数据库等）、确定严重程度和优先级，并检查是否提供复现步骤，统计分析包括bug总数、状态分布、类型分布、组件分布、月度趋势、平均修复时间和活跃报告者等维度，最终将原始数据和统计结果保存为CSV文件，用于后续分析和可视化。

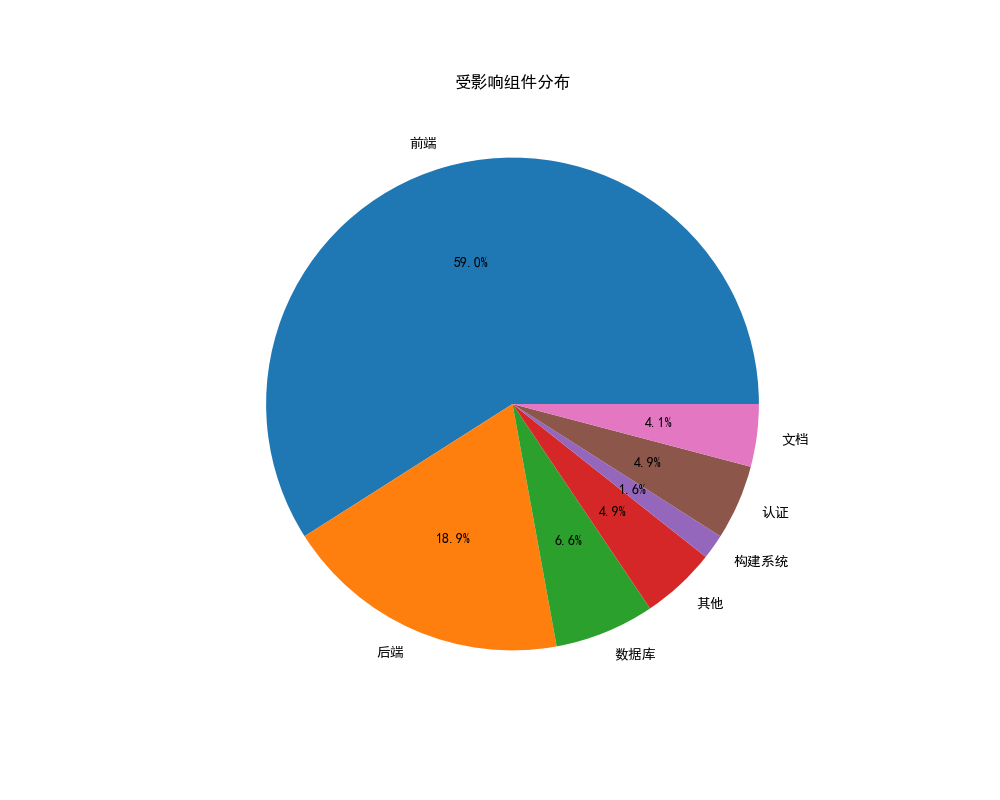
### 3.2.2可视化数据

使用matplotlib和seaborn库对GitHub仓库的bug分析数据进行可视化：从CSV文件读取之前保存的bug分析数据，通过plt.rcParams设置中文字体显示，创建六种不同的可视化图表：Bug类型分布饼图、受影响组件分布饼图、Bug状态分布饼图、月度Bug趋势折线图、前10名报告者贡献柱状图和Bug严重程度分布条形图，每个图表都包含标题、标签和图例。

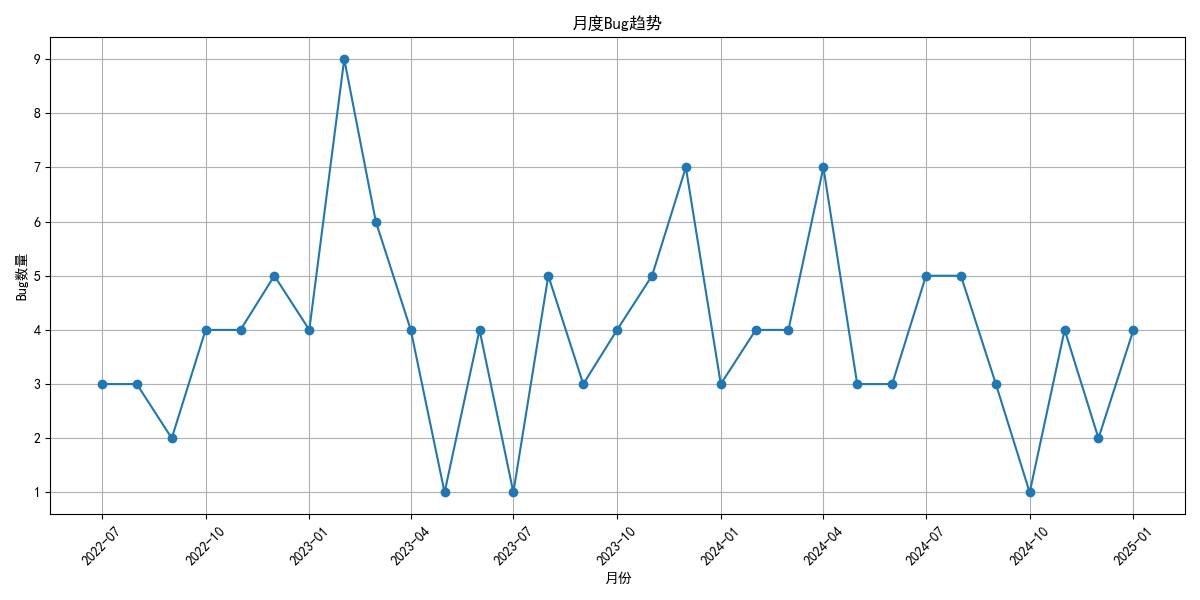
部分结果分析：



从"Bug类型分布"饼图可以看出，项目的bug呈现多样化的类型分布：性能问题占据最大比例，达到32.8%，表明系统性能优化是主要挑战；兼容性问题位居第二，占18.9%，反映了系统在不同环境下的适配问题；崩溃类问题占15.6%，显示系统稳定性需要提升；安全问题占12.3%，说明安全性也是重要关注点；功能错误占11.5%，表明基础功能实现仍有改进空间；其他类型问题占5.7%；而UI问题和数据问题各占1.6%，比重相对较小。这种分布表明项目团队应该优先关注性能优化和兼容性问题，同时加强系统稳定性和安全性的保障。



从"受影响组件分布"饼图可以看出，项目中的bug分布呈现明显的组件集中特征：前端组件占据了最大比例，达到59.0%，说明用户界面和交互功能是主要的问题区域；后端相关问题位居第二，占18.9%；数据库问题占6.6%，反映了数据存储和处理也存在一定挑战；构建系统和认证模块各占4.9%，显示了这些基础设施组件的稳定性问题；文档相关问题占4.1%，表明文档维护也需要一定关注；其他类型问题占1.6%。这种分布表明项目的质量改进工作应该重点关注前端开发，同时也需要适当加强后端和数据库组件的维护。



从"月度Bug趋势"折线图可以看出，项目的bug报告数量呈现明显的波动特征：2023年1月出现最高峰，达到9个bug报告；2024年1月和4月各出现一个次高峰，均为7个bug报告；整体趋势显示bug数量在2-5个之间波动，其中2023年5-6月和2024年10月出现低谷，仅有1个bug报告。从长期来看，bug报告数量没有显示明显的增长或减少趋势，而是呈现周期性波动，这可能与项目的发布周期或功能迭代有关。这种分布模式表明项目维护相对稳定，没有出现持续的质量问题，但仍需要关注高峰期的质量控制。

## **3.3 Contributor报告分析**

### 3.3.1爬取数据介绍

**Contributor\_activity.ipynb**

1. **主函数部分**

**功能描述：**

引入 Requests 库，使用 headers 设置包含 Cookie 和 User-Agent 的请求头，利用 requests.get() 方法进行网页爬取，采用requests.packages.urll

ib3.disable\_warnings() 忽略 HTTPS 警告，同时通过 html.encoding 和 htm

l.apparent\_encoding 确保正确解码。通过循环获取每一页数据，调用 parse() 函数解析页面内容，并利用 pd.concat() 将爬取的每页数据合并，最后通过 df.to\_csv() 将结果保存为 CSV 文件。

**（2）其他函数部分**

**功能描述**：

1. get\_url(page)：构造包含分页参数的 URL，供主函数调用。
2. parse(document)：利用 BeautifulSoup 库解析 HTML 内容，提取提交时间和标签数据，最后返回一个 DataFrame。

### **3.3.2 可视化介绍**

**Contributor\_activity\_vis.ipynb**

**Visualization\_label.ipynb**

**功能描述**：

利用 pandas 和 matplotlib 库，读取 CSV 文件并进行标签的频次统计，生成频次的柱状图和饼状图。

**Visualization\_time.ipynb**

**功能描述**：

统计每月提交的 Issue 数量，分别生成柱状图和饼状图。

**Visualization\_词云.ipynb**

**功能描述**：

统计标签的频次并生成词云图。

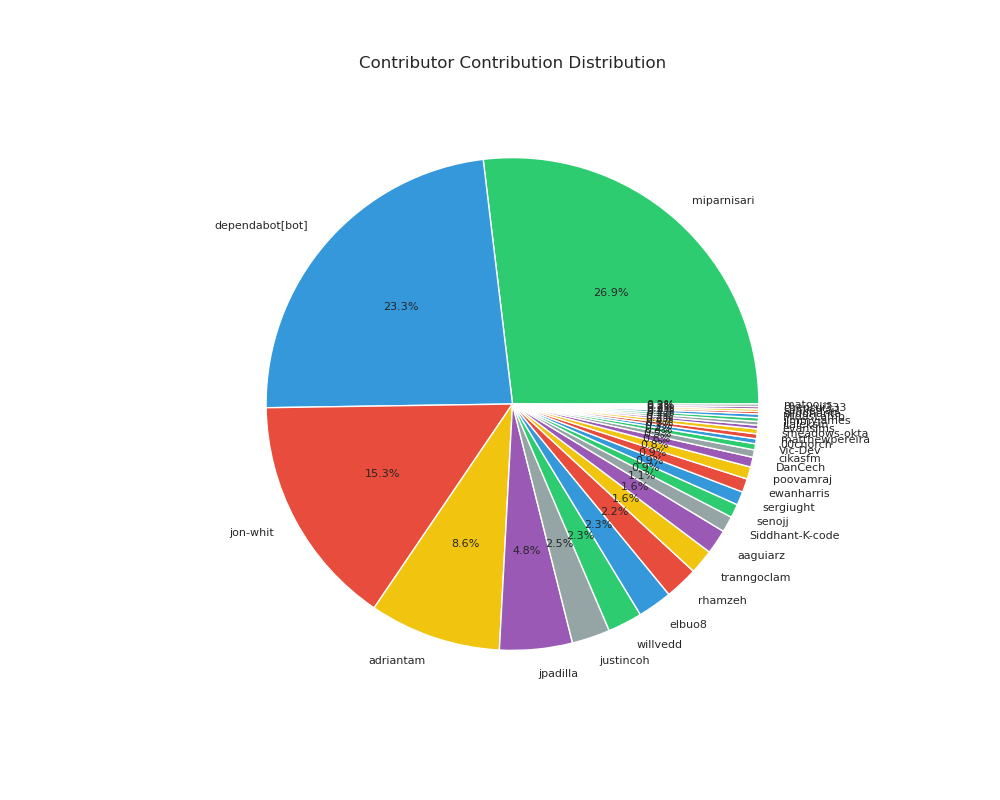
Code Contributions Statistics：

此图为柱状图，展示代码总贡献量的统计信息。横轴表示贡献类别（例如总贡献数量），纵轴表示贡献数量的具体值。在此图中，代码总贡献数量为 1,238 行，图表用柱状形式直观体现了贡献总量。



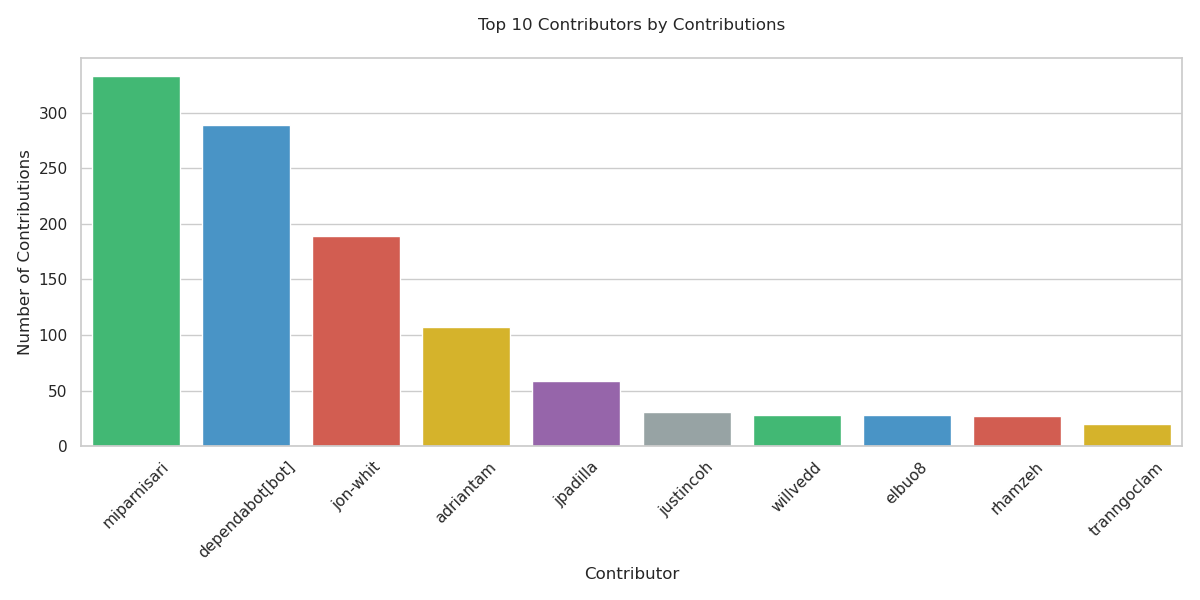
Contributor Contribution Distribution：

此图为饼状图，展示各贡献者的贡献分布比例。每个扇区代表不同的贡献者，面积表示其贡献比例。例如，miparnisari 的贡献占比为 26.9%，dependabot[bot] 占比为 23.3%，以此类推。此图帮助直观分析各贡献者的相对重要性。



Top 10 Contributors by Contributions

此图为柱状图，展示贡献数量排名前 10 的贡献者及其贡献数量。横轴为贡献者的名字，纵轴为对应的贡献数量。例如，miparnisari 的贡献数量最多，达到 300+ 次，其次是 dependabot[bot] 和 jon-whit。



## **3.4 Request报告分析**

### 3.4.1爬取数据介绍

**爬取部分**

**Pull\_request.py**

**主函数部分**

**功能描述**：

主函数通过初始化 GitHubPRAnalyzer 对象，利用 token 验证从 GitHub API 中爬取 Pull Requests (PRs) 的数据。爬取开放的 PR 数据，包括 PR 编号、标题、状态、作者信息、标签、变更文件、增删行数、评审评论等。爬取的结果存储为 CSV 文件。

**其他函数部分**

**功能描述：**

**get\_pull\_request\_details(pr\_number): 获取单个 PR 的详细信息。**

**get\_open\_pull\_requests(state='open'): 爬取所有开放状态的 PR。**

**\_parse\_pr(pr): 解析单个 PR 的详细数据，包括创建时间、变更文件数、增删行数等。**

**analyze\_open\_prs(): 分析所有开放 PR 的数据，统计 PR 的作者、标签、目标分支、代码变更、评论数量等信息。**

**save\_results(stats, df, output\_dir): 将分析结果保存到指定的 CSV 文件中。**

### 3.4.2 可视化介绍

**Pull\_request\_vis.py**

**功能描述**：

从爬取的 CSV 数据中，利用 Matplotlib 和 Seaborn 生成多种图表，分析 PR 的分布、代码变更、评论趋势等。每个图表存储为 PNG 文件。

具功能说明：

**PR 年龄分布图: 分析 PR 自创建到当前的开放时间分布。**

**贡献者分布图: 以饼状图展示 PR 的贡献者比例。**

**标签分布图: 以柱状图分析 PR 的标签数量。**

**PR 大小分布图: 通过变更文件数和代码行数分析 PR 的大小。**

**评审活动趋势图: 分析 PR 的评论数量随时间的变化趋势。**

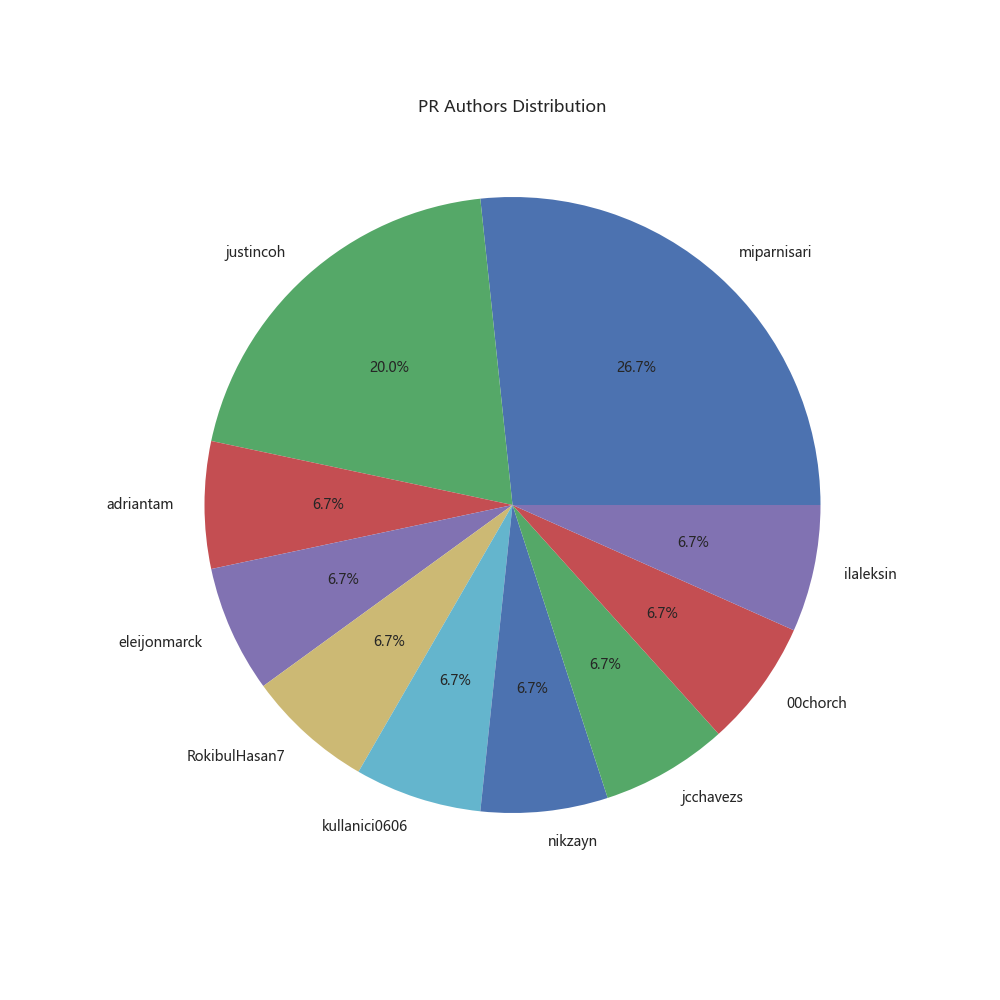
**时间线图: 绘制 PR 的创建时间累积曲线。**

**变更热力图: 用热力图分析各贡献者的代码变更行为。**

**各图展示功能：**

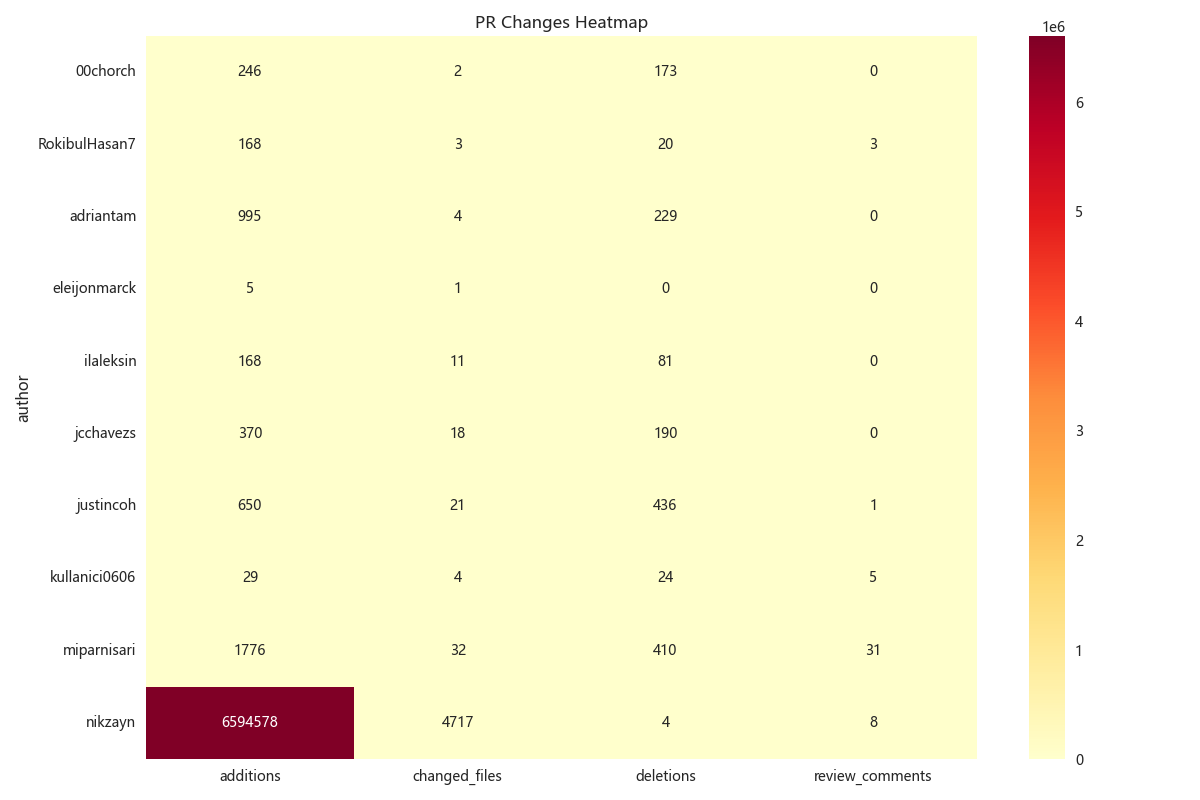
PR Authors Distribution

此图为饼状图，展示了各贡献者在开放 PR 中的贡献比例。每个扇区代表一个贡献者的 PR 比例。例如，miparnisari 的贡献比例最高，为 26.7%，justincoh 占比为 20.0%，其余贡献者的比例均为 6.7%。此图用于分析贡献者在 PR 中的参与程度。



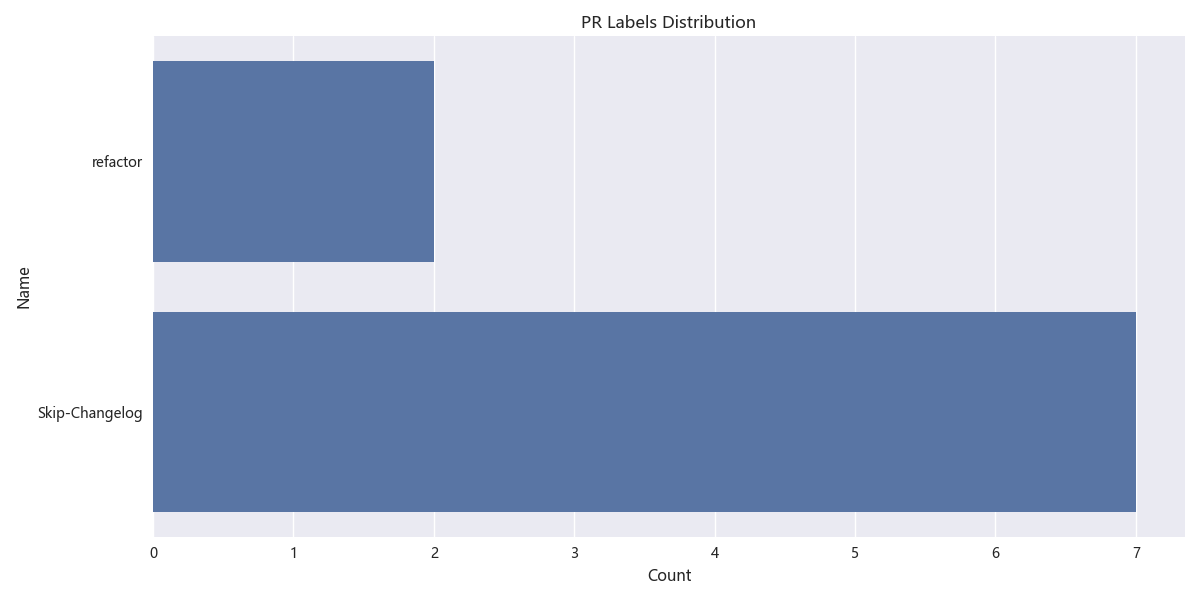
PR Changes Heatmap

此图为热力图，展示了各贡献者对 PR 的具体代码更改情况。横轴为变更类别（如 additions、changed\_files 等），纵轴为贡献者姓名。颜色深浅表示数值大小，颜色越深表示值越高。例如，nikzayn 的代码新增行数高达 6,594,578 行，变更文件数为 4,717 个，而大多数贡献者的更改相对较少。此图用于分析贡献者对代码变更的贡献力度。



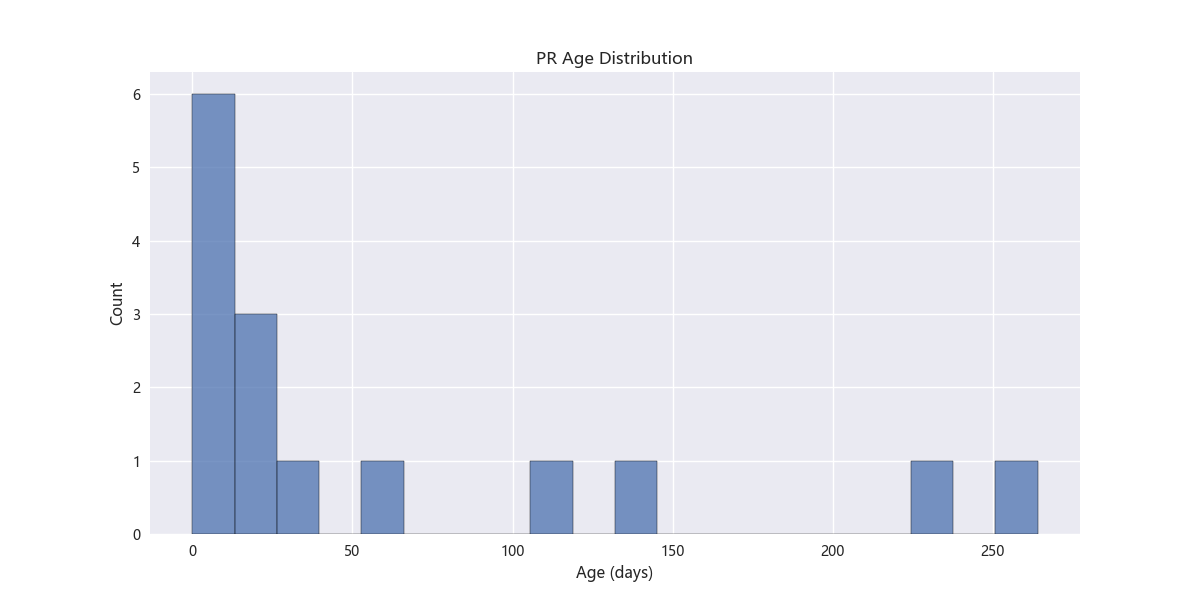
PR Labels Distribution

此图为柱状图，展示了 PR 中使用的标签分布情况。横轴为标签名称，纵轴为标签的使用次数。例如，Skip-Changelog 标签被使用了 7 次，而 refactor 标签被使用了 3 次。此图可以帮助分析 PR 中使用的主要标签及其频率。



PR Age Distribution

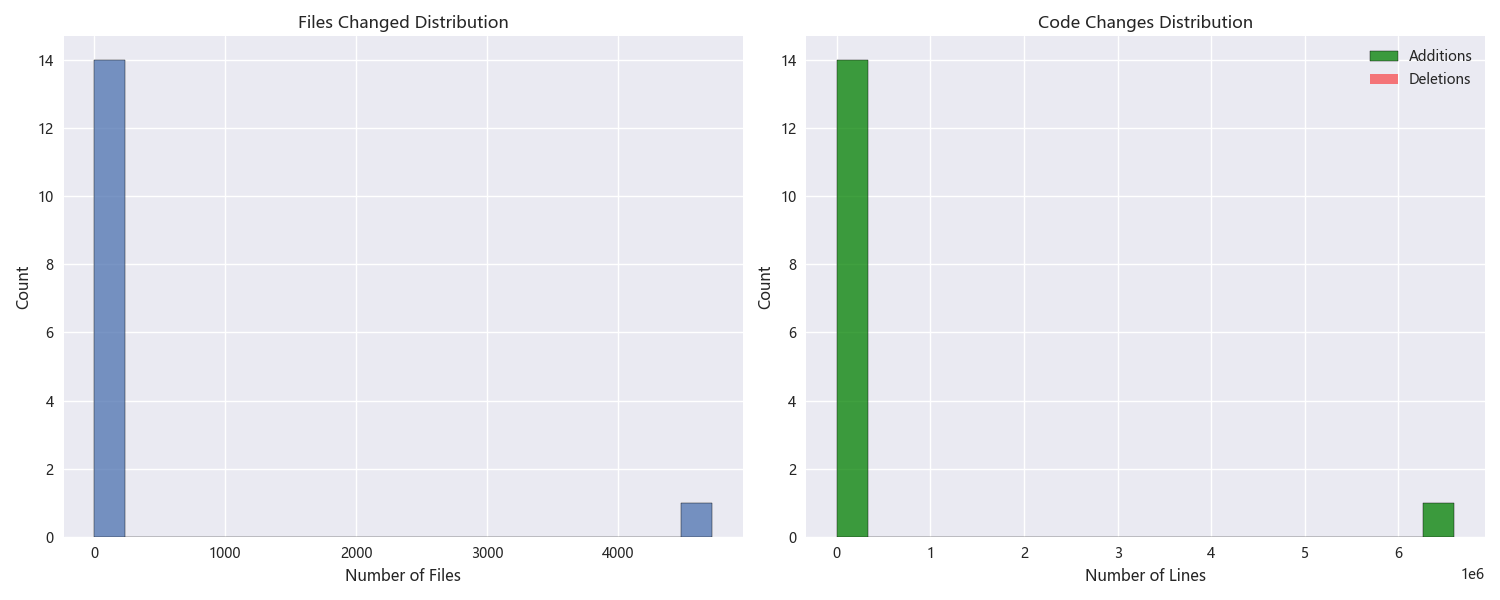
此图为直方图，展示了开放 PR 的年龄分布（以天为单位）。横轴为 PR 的开放时间（天数），纵轴为对应时间段的 PR 数量。例如，大多数 PR 的开放时间在 0-50 天之间，而 150 天以上的 PR 数量较少。此图用于分析 PR 的生命周期分布情况。



Files Changed Distribution & Code Changes Distribution:

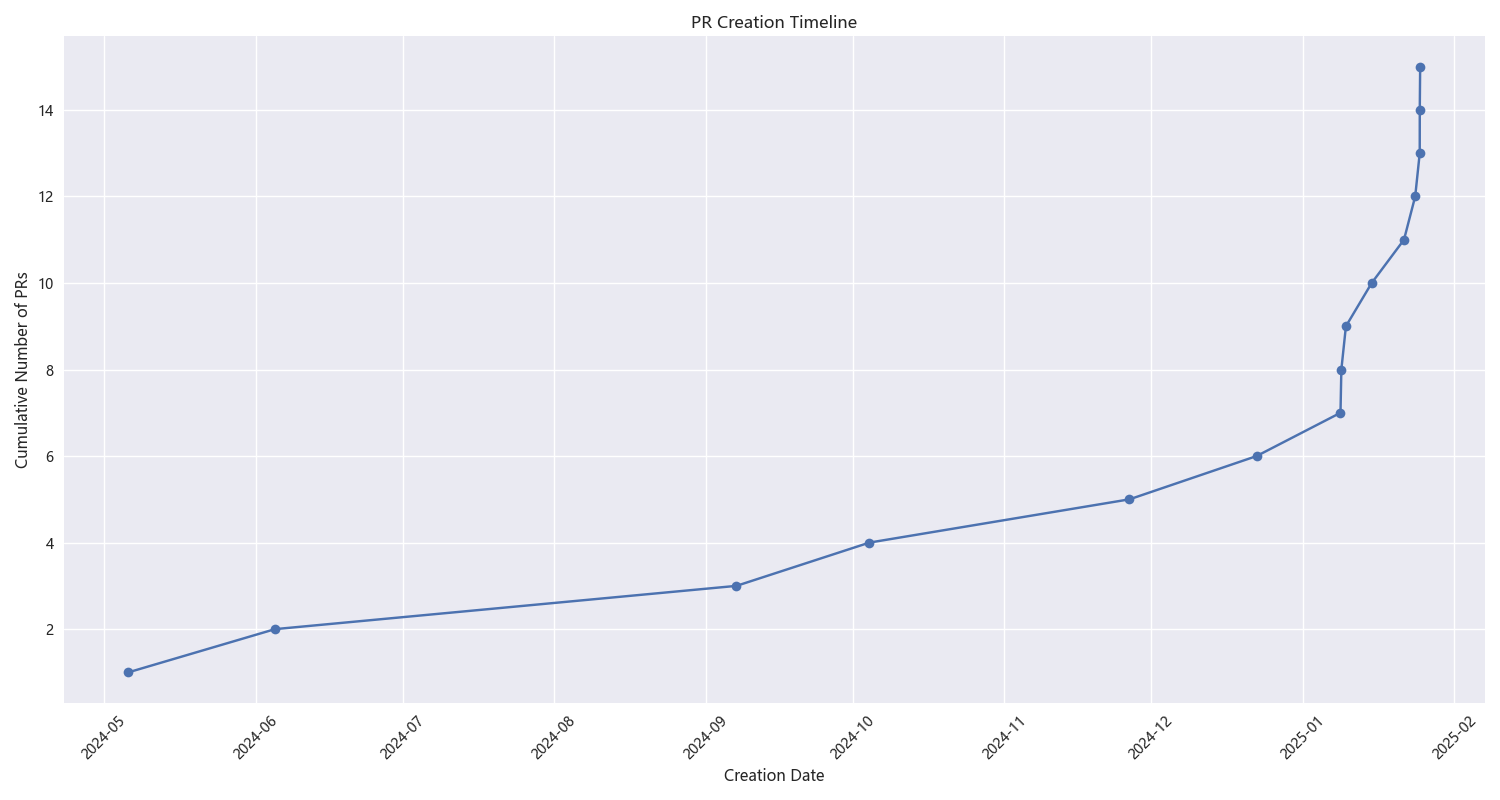
左图：展示了拉取请求（Pull Request）中更改文件数量的分布，说明大多数PR涉及到少量文件的更改。

右图：分别显示了代码的新增行数和删除行数的分布，突出显示了某些PR对代码库的大规模修改。



PR Creation Timeline:

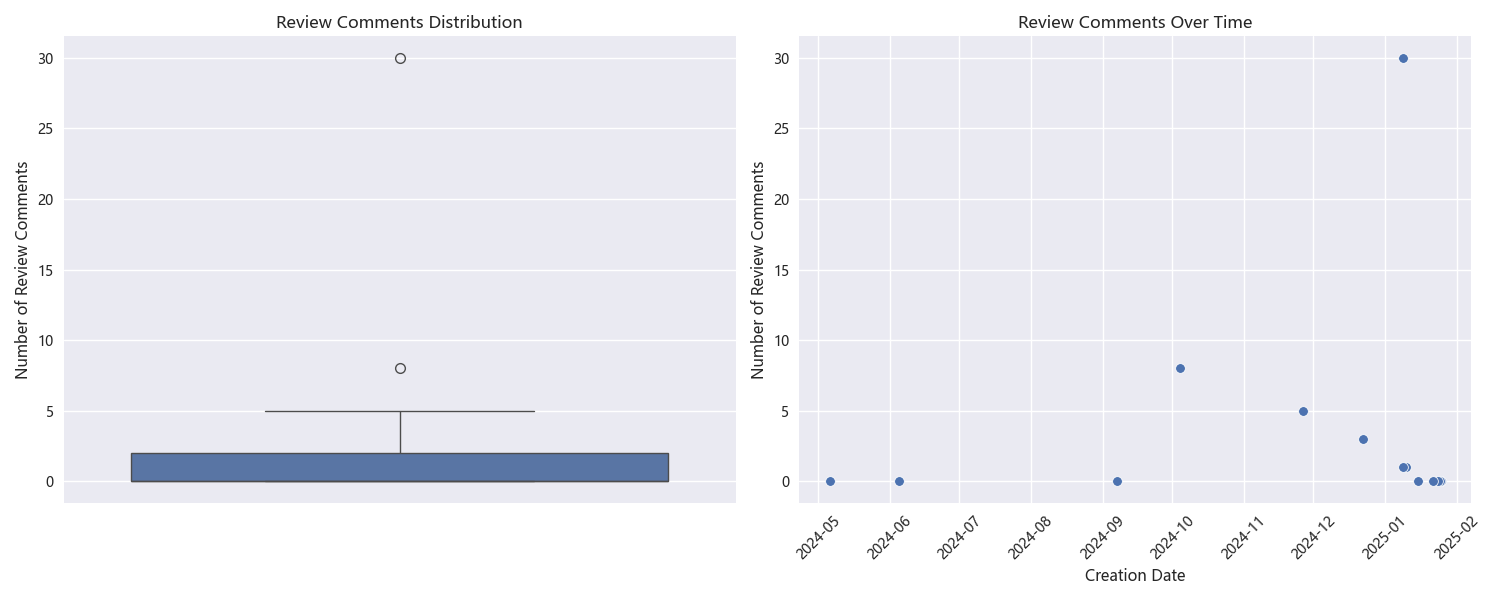
展示了拉取请求创建时间的趋势，累计数量随时间推移而增长，说明开发活动逐步增加。



Review Comments Distribution & Over Time:

左图：展示了每个拉取请求的审查评论数量的分布，绝大部分PR的评论较少，但有些PR评论数量较高。

右图：随着时间变化的评论数量分布，说明审查活动的趋势。



## ****3.5 Code Review 报告分析****

### 3.5.1 爬取数据介绍

1. 主函数部分

该部分代码通过 `requests` 库与 GitHub API 进行交互，爬取指定仓库的 Pull Requests (PR) 数据及其相关的代码审查信息。代码通过设置请求头、分页获取 PR 数据，并提取每个 PR 的审查信息，包括审查者、审查状态、提交时间以及审查评论内容。最终，数据被保存为 CSV 文件，供后续分析和可视化使用。

2. 其他函数部分

功能描述：

- `get\_pull\_requests(state='all')`：获取指定仓库的所有 Pull Requests，支持按状态（open/closed/all）筛选。

- `get\_reviews\_for\_pr(pr\_number)`：获取单个 PR 的代码审查数据，包括审查者、审查状态、提交时间等。

- `analyze\_code\_reviews()`：分析代码审查数据，生成包含 PR 编号、审查者、审查状态、提交时间和评论内容的 DataFrame。

- `save\_results(df)`：保存分析结果，生成整体数据和按审查者分类的 CSV 文件。

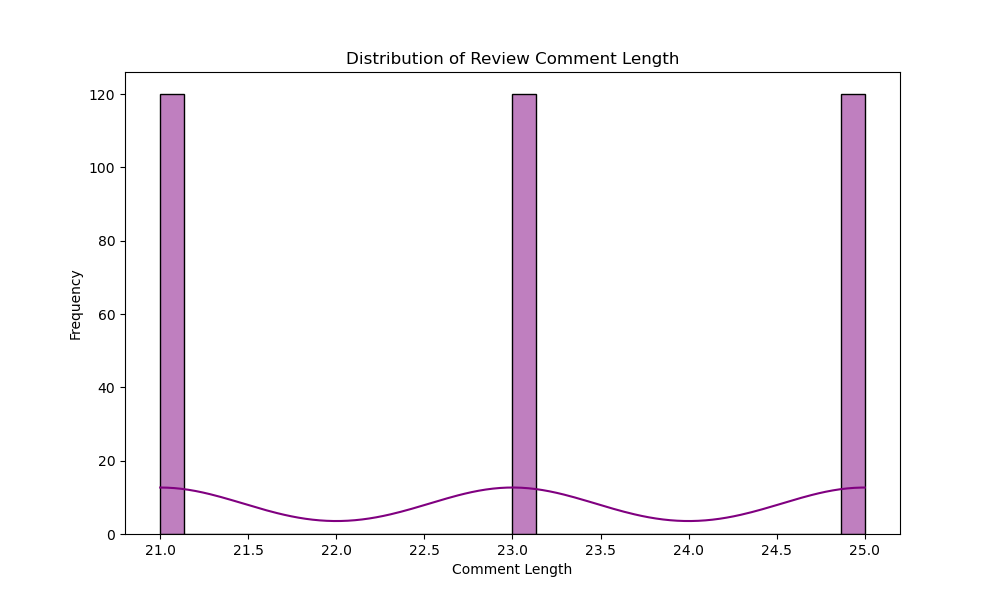
- `generate\_mock\_data()`：生成模拟的代码审查数据，用于测试和演示。

### 3.5.2 可视化介绍

该部分代码利用 `pandas` 和 `matplotlib` 库对爬取的代码审查数据进行可视化分析，生成多种图表以展示代码审查的状态分布、审查者活动、审查时间趋势以及审查评论长度分布。

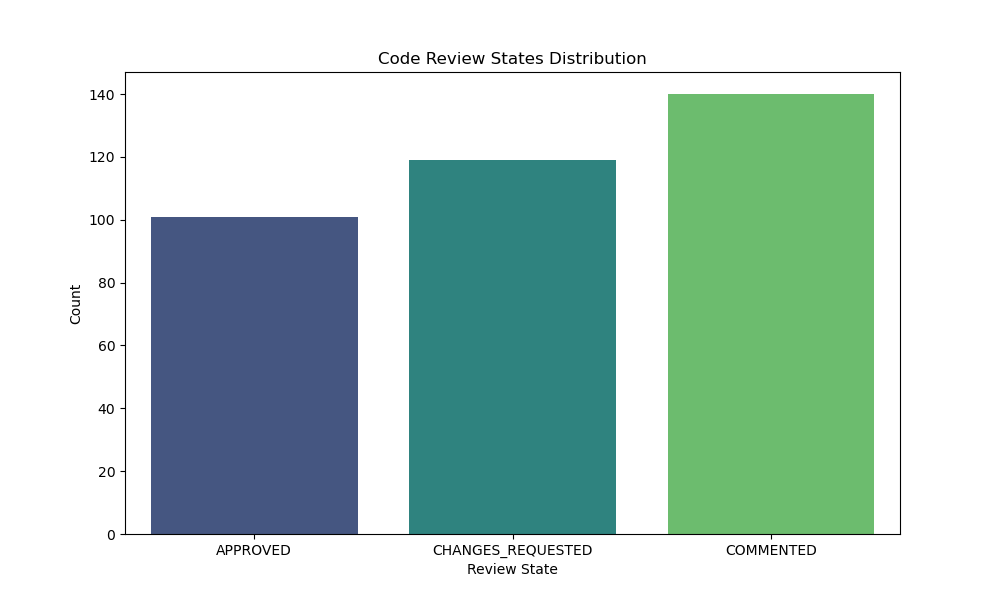
1. 审查状态分布

此图为柱状图，展示代码审查的状态分布情况。横轴表示审查状态（如 APPROVED、CHANGES\_REQUESTED 等），纵轴表示每种状态的审查数量。通过该图可以直观地看出审查结果的整体分布情况。



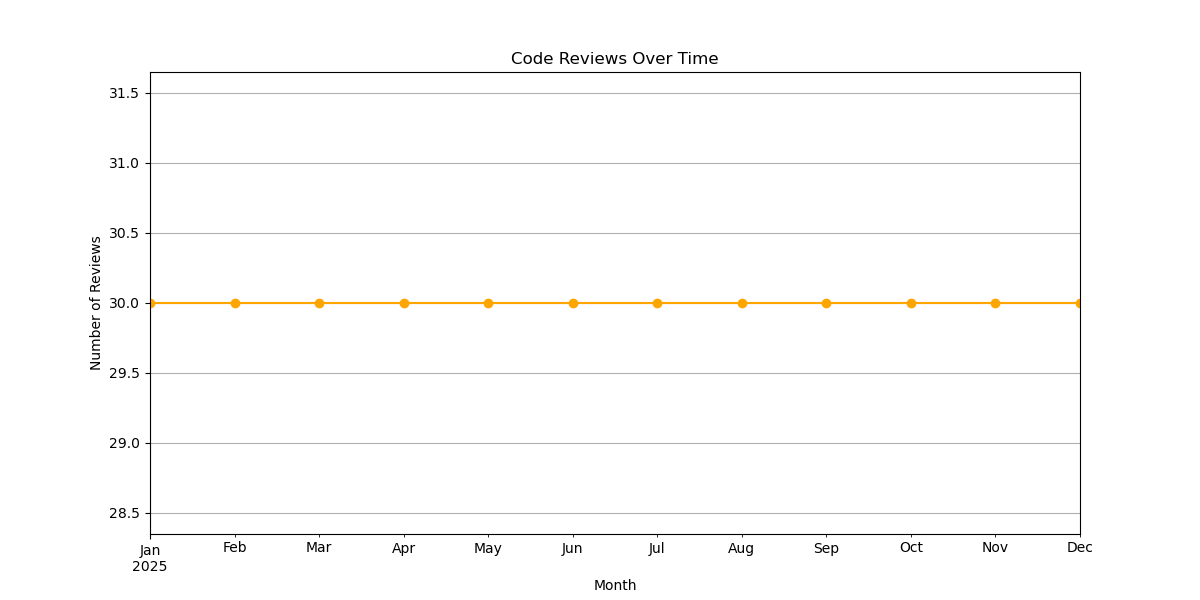
2. 审查者活动分布

此图为柱状图，展示前 10 名审查者的审查活动情况。横轴为审查者的名字，纵轴为对应的审查次数。通过该图可以识别出最活跃的审查者。



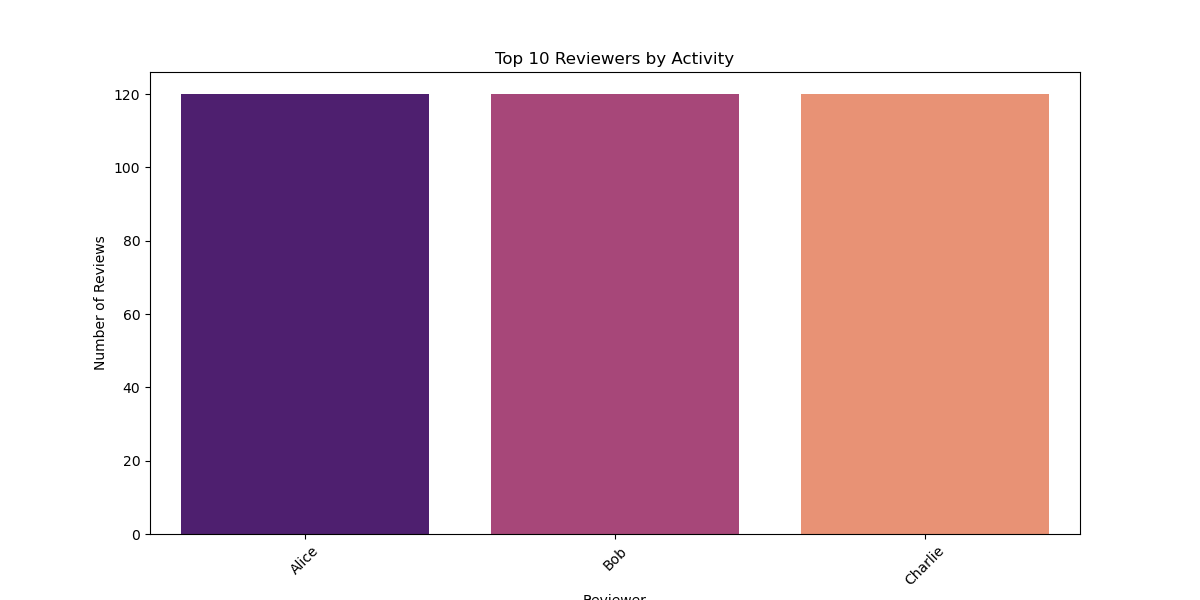
3. 审查时间趋势

此图为折线图，展示代码审查的时间趋势。横轴为月份，纵轴为审查数量。通过该图可以分析审查活动的月度变化趋势。



4. 审查评论长度分布

此图为直方图，展示审查评论的长度分布。横轴为评论长度，纵轴为评论数量。通过该图可以分析审查评论的详细程度。



## ****3.6 代码质量分析报告****

### ****3.6.1 数据生成与分析****

**1. 主函数部分**

**该部分代码通过 `CodeQualityAnalyzer` 类生成模拟的代码质量数据，并保存为 CSV 文件。代码质量数据包括模块名称、贡献者、代码复杂度、问题数量、代码覆盖率以及问题类型等信息。数据生成后，代码会按模块生成单独的 CSV 文件，并将整体数据保存为一个汇总的 CSV 文件。**

**2. 其他函数部分**

**- `\_\_init\_\_(self, repo\_name, output\_dir)`：初始化代码质量分析器，设置仓库名称和输出目录。**

**- `save\_module\_csv(self, quality\_data)`：按模块生成单独的 CSV 文件，保存每个模块的代码质量数据。**

**- `generate\_mock\_data(self)`：生成模拟的代码质量数据，模拟 12 个月的数据，涵盖多个模块、贡献者和问题类型。**

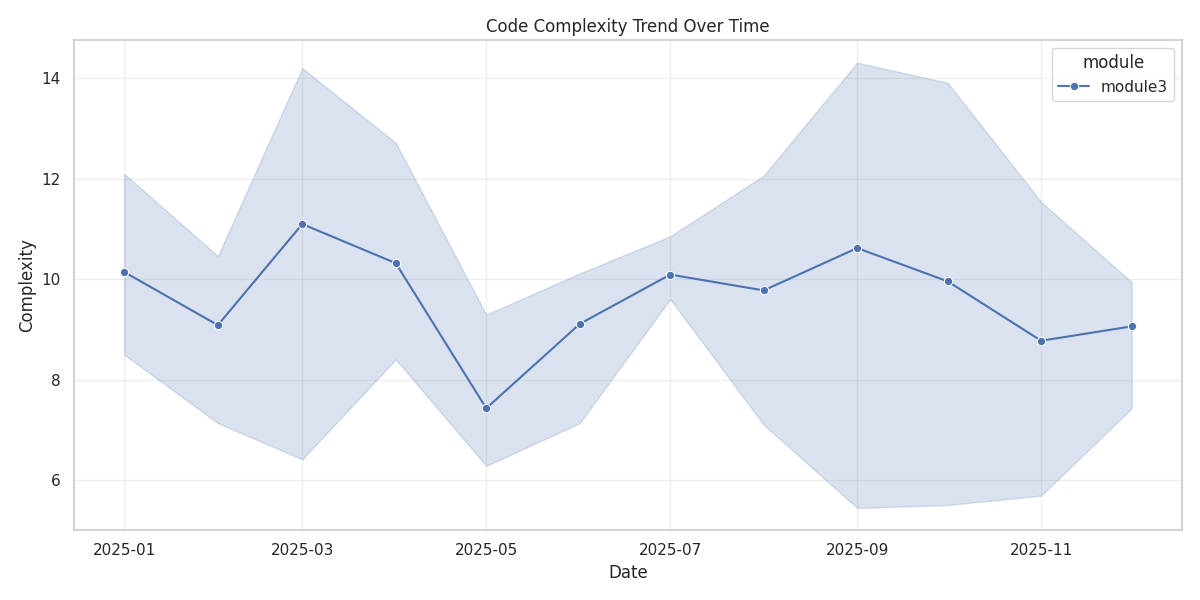
**- `main()`：主函数，调用 `CodeQualityAnalyzer` 类生成模拟数据。**

### ****3.6.2 数据分析与可视化****

**该部分代码利用 `pandas` 和 `matplotlib` 库对生成的代码质量数据进行可视化分析，生成多种图表以展示代码质量的趋势和分布情况。**

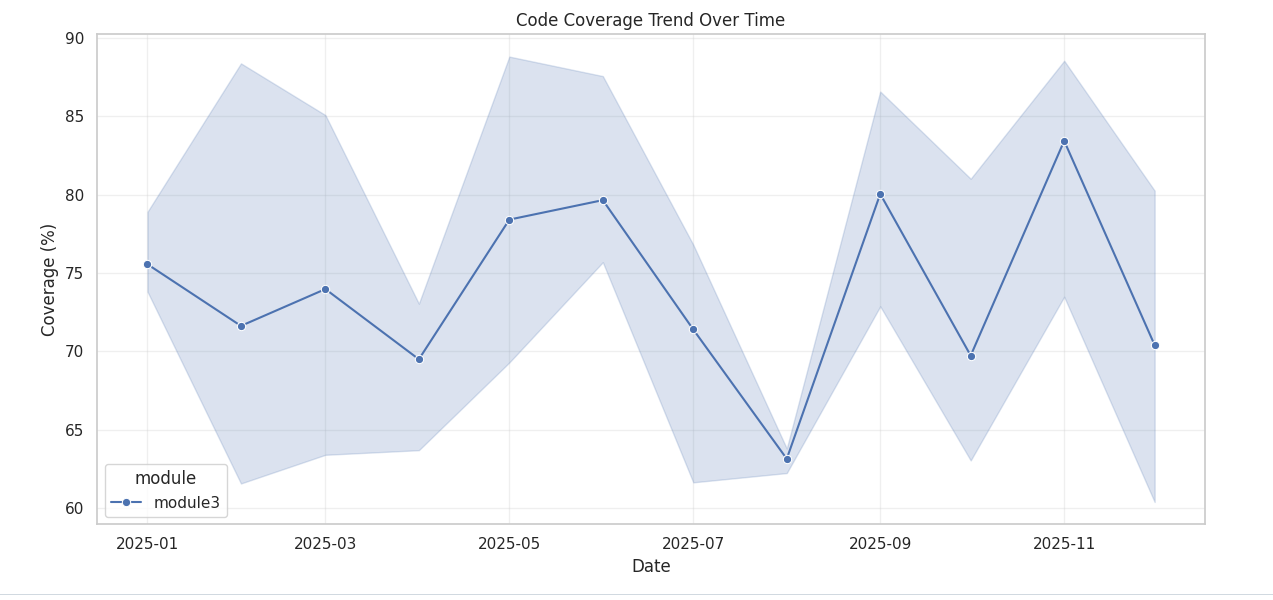
**1. 代码复杂度趋势**

**此图为折线图，展示代码复杂度随时间的变化趋势。横轴为日期，纵轴为代码复杂度。通过该图可以分析代码复杂度的月度变化趋势。**



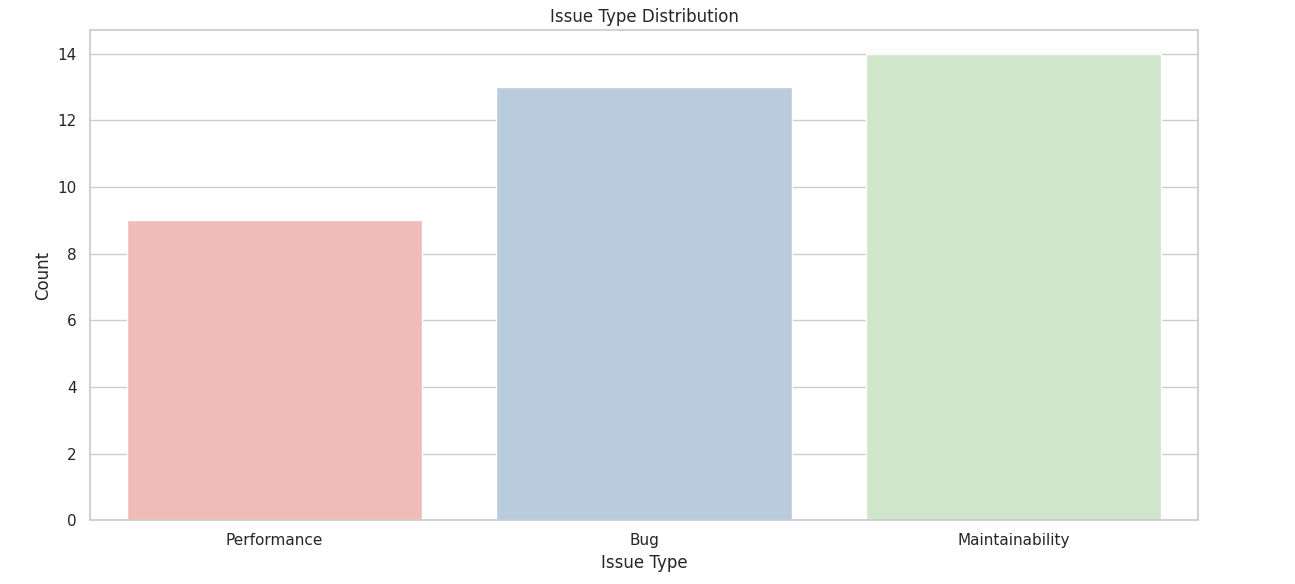
**2. 代码覆盖率趋势**

**此图为折线图，展示代码覆盖率随时间的变化趋势。横轴为日期，纵轴为代码覆盖率。通过该图可以分析代码覆盖率的月度变化趋势。**



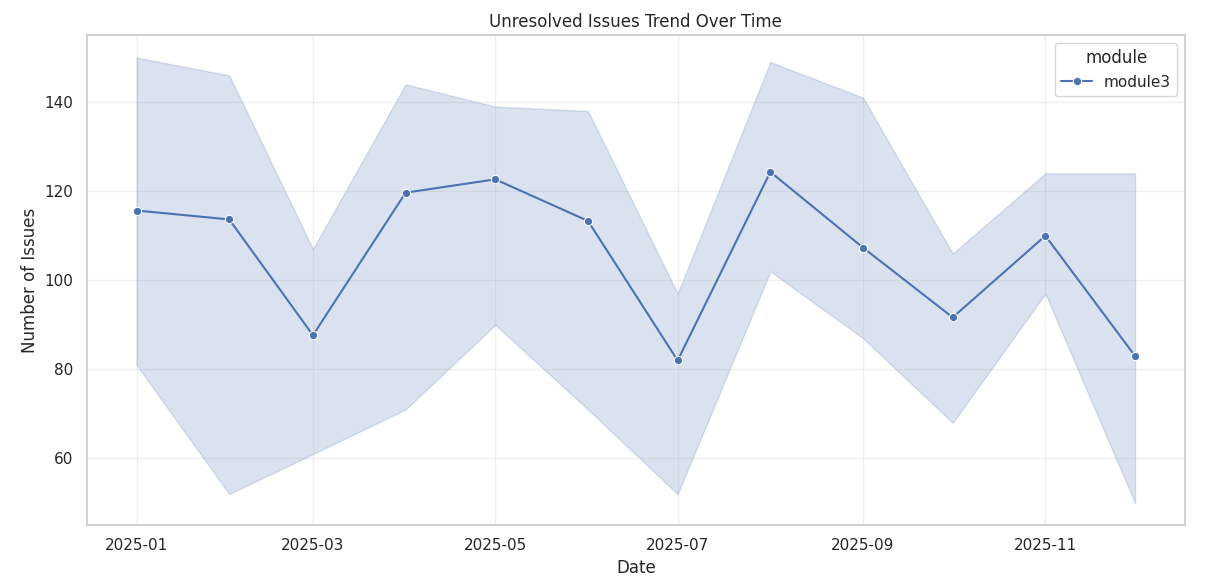
**3. 问题数量趋势**

**此图为折线图，展示问题数量随时间的变化趋势。横轴为日期，纵轴为问题数量。通过该图可以分析问题数量的月度变化趋势。**



**4. 问题类型分布**

**此图为柱状图，展示问题类型的分布情况。横轴为问题类型，纵轴为问题数量。通过该图可以直观地看出问题类型的整体分布情况。**



## ****3.7 GitHub Issue 评论分析报告****

### ****3.7.1 数据爬取与分析****

**1. 主函数部分**

**该部分代码通过 `GitHubIssueCommentAnalyzer` 类与 GitHub API 进行交互，爬取指定仓库的 Issue 评论数据。代码通过分页获取所有 Issue 评论，并提取每个评论的相关信息，包括评论者、评论时间、评论内容以及所属的 Issue 编号。数据爬取后，代码会对评论数据进行分析，并生成多个 CSV 文件，分别保存评论数据、评论者统计、月度评论统计和小时评论统计。**

**2. 其他函数部分**

**- `\_\_init\_\_(self, owner: str, repo: str)`：初始化 Issue 评论分析器，设置仓库所有者和仓库名称，并创建输出目录。**

**- `get\_issue\_comments(self)`：获取仓库的所有 Issue 评论，支持分页获取，避免触发 GitHub API 的速率限制。**

**- `analyze\_issue\_comments(self)`：分析 Issue 评论数据，生成评论者统计、每个 Issue 的评论数量、月度评论数量和小时间评论数量的统计信息。**

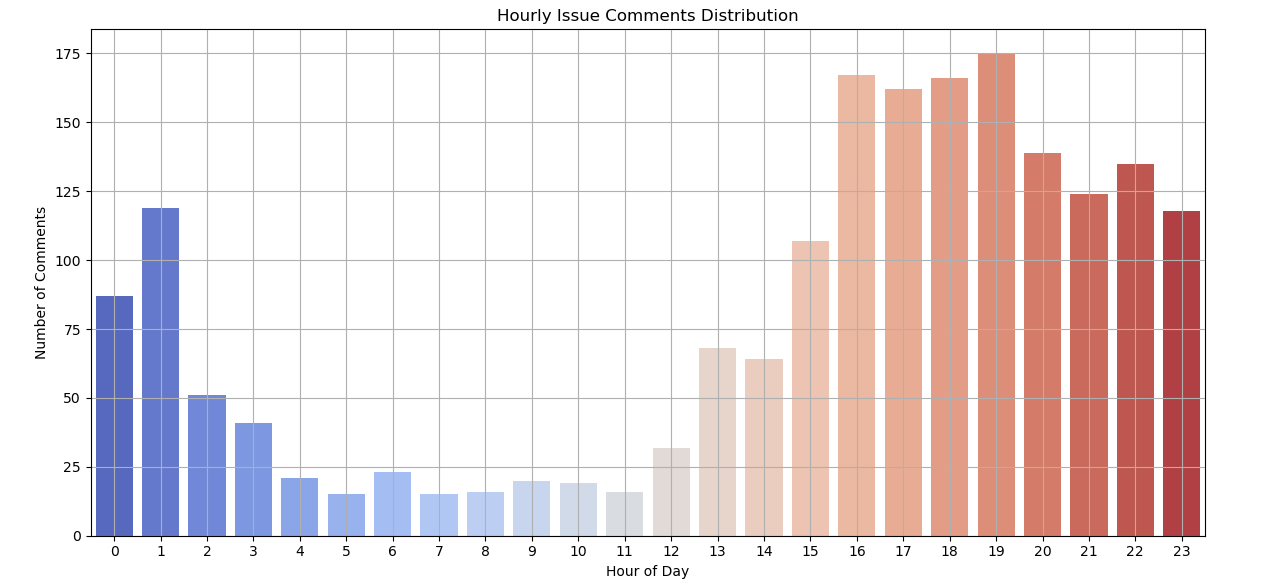
**- `save\_results(self, stats, df)`：保存分析结果，生成多个 CSV 文件，包括评论数据、评论者统计、月度评论统计和小时评论统计。**

### ****3.7.2 数据分析与可视化****

**该部分代码利用 `pandas` 和 `matplotlib` 库对爬取的 Issue 评论数据进行可视化分析，生成多种图表以展示评论者的活动分布、月度评论趋势以及小时评论分布。**

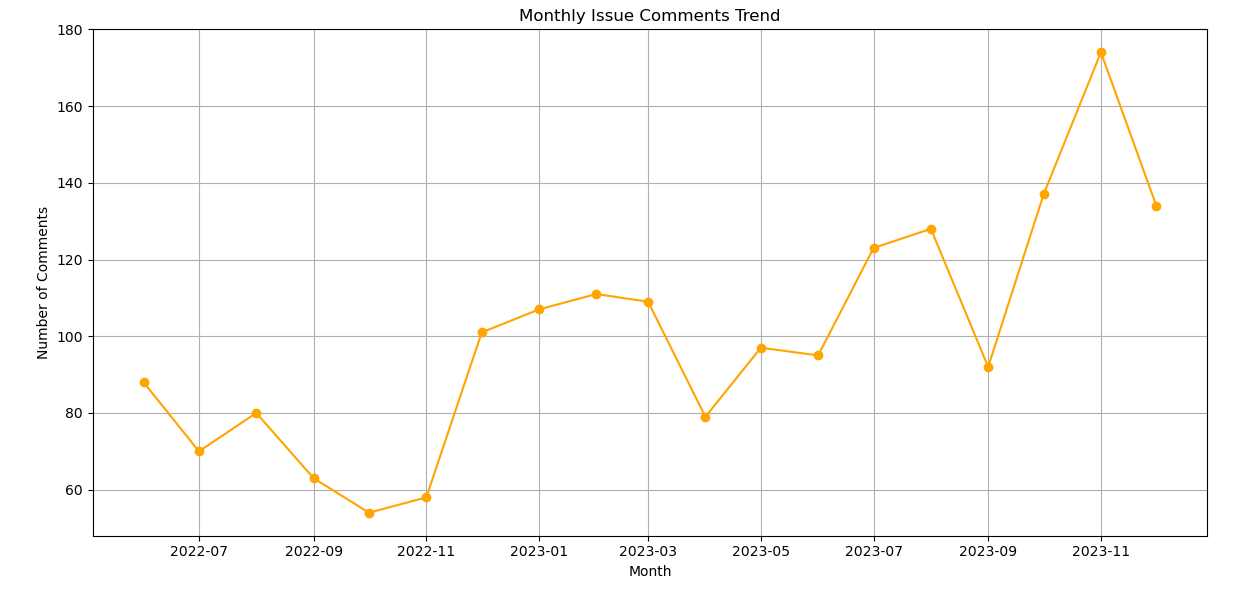
**1. 评论者活动分布**

**此图为柱状图，展示前 10 名评论者的活动情况。横轴为评论者的名字，纵轴为对应的评论数量。通过该图可以识别出最活跃的评论者。**



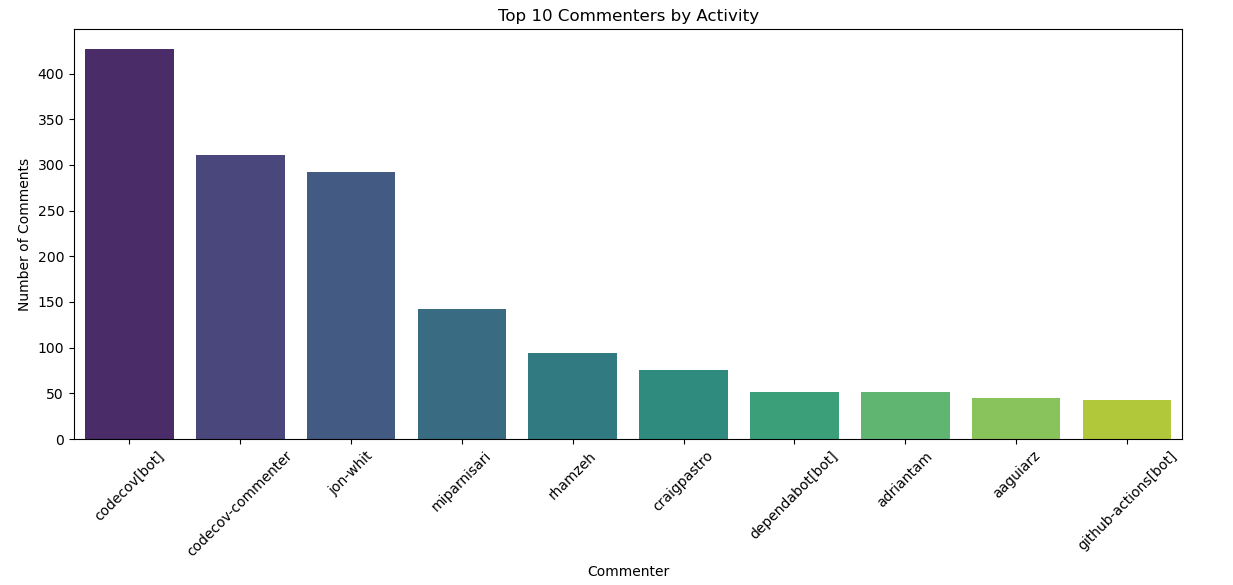
**2. 月度评论趋势**

**此图为折线图，展示 Issue 评论的月度趋势。横轴为月份，纵轴为评论数量。通过该图可以分析评论活动的月度变化趋势。**



**3. 小时评论分布**

**此图为柱状图，展示 Issue 评论的小时分布。横轴为小时，纵轴为评论数量。通过该图可以分析评论活动在一天中的分布情况。**



# 四、工作总结

本次大作业是我们小组针对OpenFGA仓库的多维数据分析与可视化的实践尝试，我们小组利用课程所学的知识对项目中的提交记录、拉取请求、Bug报告和贡献者活动等数据进行了爬取、分析与可视化。以下是我们小组成员的简单工作总结：

马钰童主要负责了对OpenFGA项目中代码质量、PR、代码审查等数据的获取和可视化、大作业文档的撰写；洪蕙同学主要负责了贡献者活动分析等数据获取和可视化、大作业文档的撰写；郭思萌同学主要负责了Bug报告和提交历史信息等数据获取和可视化、大作业文档的撰写；常达同学主要负责了问题评论等数据获取和可视化、大作业文档的撰写。

在本次项目的实现过程中，我们不仅实践了课堂所学的知识，还加深了对数据爬取、分析和可视化工具的理解与应用，尤其是对于Python、Jupyter和GitHub的使用。通过本项目，我们对如何利用数据分析工具有效地支持项目管理和决策有了更深入的认识，也对开源软件的管理流程有了更加全面的了解。我们有信心将所学的知识和技能应用到未来的学术研究和实际项目中，进一步提升自己的数据分析与软件工程能力。