# 1.数据分析部分

## **1.GitHub仓库数据分析与可视化**

**一、项目背景**

为了更好地理解GitHub中开源项目beeware的活动和贡献情况，我们开发了一个Python脚本，该脚本能够利用GitHub API获取仓库的基本信息、文件列表、README内容、贡献者数据以及提交记录。通过解析这些数据，我们不仅能够了解仓库的现状，还能分析其历史发展过程，并以图表的形式直观地展示出关键指标。

**二、技术实现**

**1. 环境配置**

**库依赖**：requests, os, certifi, base64, pandas, matplotlib

**字体设置**：为确保中文标签正确显示，设置了Matplotlib全局字体为SimHei，并解决了负号显示问题。

**2. 数据获取**

**2.1 获取仓库基本信息**

使用GitHub API接口，通过提供仓库所有者（owner）和仓库名称（repo），成功获取了包括仓库描述、创建日期、编程语言、星标数量及Fork数量在内的详细信息。

**2.2 文件与README内容获取**

实现了从指定仓库中提取文件列表及其大小的功能，并特别针对README.rst文件进行了内容读取，将其解码并打印前200个字符供预览。

**2.3 贡献者信息收集**

通过API调用，收集了参与该项目的所有贡献者的用户名和他们的贡献次数，为后续的数据分析奠定了基础。

**2.4 提交记录获取**

考虑到GitHub API对结果分页返回的特点，我们设计了一套机制来遍历所有页面，确保不会遗漏任何提交记录。同时，还根据需要设定了起始和结束日期，以便精确控制数据的时间范围。

**3. 数据处理与可视化**

**贡献者贡献次数图**：基于收集到的贡献者数据，绘制了柱状图，清晰地展示了每位贡献者的贡献次数。

**提交记录时间分布图**：将所有提交按月份汇总，生成了折线图，用以观察项目的活跃度随时间的变化趋势。

**三、遇到的问题及解决方案**

1. **SSL证书验证问题**：在进行HTTPS请求时遇到了SSL证书验证失败的问题。解决办法是安装了certifi库，并通过环境变量指定了CA证书的位置。
2. **API速率限制**：为了避免触发GitHub API的速率限制，我们在代码中加入了适当的延时，并优化了请求逻辑，如采用批量获取数据的方式减少请求数量。
3. **中文显示问题**：最初尝试在图表中显示中文时遇到了乱码现象，经过调整Matplotlib的字体设置后得到了解决。

## **2.beeware/beeware 仓库提交信息分析**

**一、项目背景**

beeware/beeware 仓库是 BeeWare 项目的主仓库，它包含了 BeeWare 套件的整体架构和一些核心组件。BeeWare 套件包括多个工具和库，如 Toga、Briefcase、Rubicon ObjC 等，这些工具和库协同工作，为开发者提供了一个完整的开发环境。例如，Briefcase 是一个用于将 Python 项目打包为可分发的成品的工具，可以将应用程序打包成适用于不同平台的安装包。

为了全面了解 beeware/beeware 仓库的开发活动和贡献情况，我们对仓库的提交记录进行了详细分析。通过图表展示，我们可以更直观地理解项目的活跃度和主要贡献者。

**二、数据分析**

**1. 贡献者分布**

**图表1：GitHub仓库贡献者贡献次数**

图表, 直方图

描述已自动生成

**主要发现**：

最主要的贡献者是 oy3742，其提交次数显著高于其他贡献者，达到了500次以上。

其次是 t[bot]，提交次数约为150次。

其他贡献者的提交次数相对较少，大部分在50次以下。

**解读**：

oy3742 和 t[bot] 是最活跃的贡献者，对项目的贡献非常大。

大多数贡献者的活动较为分散，表明项目有广泛的社区参与，但核心贡献集中在少数人身上。

**2. 提交时间分布**

**图表2：GitHub提交记录的时间分布**

图表, 直方图

描述已自动生成

**主要发现**：

从2018年到2020年初，提交数量相对较少且较为稳定。

2020年中旬开始，提交数量出现显著增加，并在2020年底达到一个小高峰。

2021年至2023年期间，提交数量波动较大，但总体上保持在一个较低的水平。

2024年初，提交数量急剧上升，达到了整个时间段内的最高点，接近160次提交。

2024年下半年，提交数量有所下降，但仍高于之前的平均水平。

**解读**：

2020年和2024年是项目最活跃的时期，可能与特定的开发周期或重大功能更新有关。

2024年初的高提交量可能表明正在进行大规模的开发工作或代码重构。

**三、关键时间节点**

1. **2020年中旬**：提交数量首次显著增加。
2. **2020年底**：出现一个小高峰。
3. **2024年初**：提交数量达到峰值。
4. **2024年下半年**：提交数量有所回落，但仍保持较高水平。

**四、建议与展望**

**项目活跃度分析**：

2020年和2024年是项目最活跃的时期，可能与特定的开发周期或重大功能更新有关。

2024年初的高提交量可能表明正在进行大规模的开发工作或代码重构。

**资源分配**：

在项目活跃期（如2024年初），确保有足够的资源和支持，以应对高频率的代码变更和潜在的问题。

在低活跃期，可以考虑进行代码审查、文档更新等维护性工作。

**社区参与**：

高峰期的大量提交可能意味着有更多的开发者参与进来。可以利用这段时间加强社区互动，鼓励更多贡献者参与。

**自动化工具**：

考虑使用自动化工具来管理频繁的提交，减少手动操作带来的错误和效率问题。

**五、总结**

通过对 beeware/beeware 仓库的提交信息进行分析，我们发现项目在2020年和2024年经历了显著的活跃期，特别是2024年初的高提交量表明了大规模的开发活动。主要贡献者 oy3742 和 t[bot] 对项目的贡献非常大，而其他贡献者的活动较为分散。未来可以进一步优化资源分配，加强社区互动，并利用自动化工具提高开发效率。

## **3.beeware/toga 仓库提交信息分析**

1. **项目背景**

beeware/toga 仓库专注于 Toga 这个组件。Toga 是 BeeWare 套件中的一个跨平台的 widget（小部件，控件）工具包。它的设计理念是“Write once, run everywhere”，即用 Python 编写一次代码，便可以在多个平台上运行。Toga 支持多个操作系统，包括 Windows、macOS、Linux、iOS 和 Android。它能够生成针对不同操作系统优化的用户界面，而无需开发者为每个平台编写特定的代码。例如，开发者可以使用 Toga 创建包含按钮、文本框等控件的图形用户界面，并在不同平台上保持一致的外观和操作体验。

为了全面了解 beeware/toga 仓库的开发活动和贡献情况，我们对仓库的提交记录进行了详细分析。通过图表展示，我们可以更直观地理解项目的活跃度和主要贡献者。

**二、数据分析**

**1. 贡献者分布**

**图表1：GitHub仓库贡献者贡献次数**

图片包含 图表

描述已自动生成

**主要发现**：

最主要的贡献者是 oy3742，其提交次数显著高于其他贡献者，达到了4000次以上。

其次是 t[bot]，提交次数约为800次。

其他贡献者的提交次数相对较少，大部分在500次以下。

**解读**：

oy3742 是最活跃的贡献者，对项目的贡献非常大。

t[bot] 作为自动化工具，也对项目有显著贡献。

大多数贡献者的活动较为分散，表明项目有广泛的社区参与，但核心贡献集中在少数人身上。

**2. 提交时间分布**

**图表2：GitHub提交记录的时间分布**

图表, 直方图

描述已自动生成

**主要发现**：

从2019年到2025年，提交数量波动较大，但总体上保持在一个较高的水平。

2019年至2020年初，提交数量出现显著增加，并在2020年底达到一个小高峰。

2021年至2023年期间，提交数量波动较大，但总体上保持在一个较高的水平。

2024年和2025年初，提交数量继续波动，但整体趋势仍然较高。

**解读**：

项目在整个时间段内都保持了较高的活跃度，特别是在2020年和2024年出现了明显的高峰。

高峰期可能与特定的开发周期或重大功能更新有关。

**三、关键时间节点**

1. **2019年至2020年初**：提交数量首次显著增加。
2. **2020年底**：出现一个小高峰。
3. **2021年至2023年**：提交数量波动较大，但总体保持较高水平。
4. **2024年和2025年初**：提交数量继续波动，但整体趋势仍然较高。

**四、建议与展望**

**项目活跃度分析**：

项目在整个时间段内都保持了较高的活跃度，特别是2020年和2024年出现了明显的高峰。

这些高峰期可能与特定的开发周期或重大功能更新有关。

**资源分配**：

在项目活跃期（如2020年底和2024年初），确保有足够的资源和支持，以应对高频率的代码变更和潜在的问题。

在低活跃期，可以考虑进行代码审查、文档更新等维护性工作。

**社区参与**：

高峰期的大量提交可能意味着有更多的开发者参与进来。可以利用这段时间加强社区互动，鼓励更多贡献者参与。

**自动化工具**：

考虑使用自动化工具来管理频繁的提交，减少手动操作带来的错误和效率问题。

t[bot] 的存在表明项目中已经使用了自动化工具，可以进一步优化这些工具以更好地支持开发流程。

**五、总结**

通过对 beeware/toga 仓库的提交信息进行分析，我们发现项目在整个时间段内都保持了较高的活跃度，特别是在2020年和2024年出现了明显的高峰。主要贡献者 oy3742 和 t[bot] 对项目的贡献非常大，而其他贡献者的活动较为分散。未来可以进一步优化资源分配，加强社区互动，并利用自动化工具提高开发效率。

# 2.测试部分

对toga工具包的部分测试：

## **1.libc.py 文件分析与功能概述**

**一、项目背景**

为了支持对 Toga 小部件创建的自动分析，我们开发了一个名为 libc.py 的 Python 脚本。该脚本使用 LibCST（Liberal CST）库来解析和遍历 Python 源代码文件中的抽象语法树（AST），以识别并记录所有通过调用 toga. 开头的方法创建的小部件实例。

**二、技术实现**

**定义 TogaWidgetFinder 类**

继承自 cst.CSTVisitor，能够遍历 CST 树并访问每个节点。

METADATA\_DEPENDENCIES 属性声明此访问器需要 PositionProvider 提供的元数据。

构造函数初始化一个空列表 widget\_calls 用来存储找到的小部件创建调用。

visit\_Call 方法检查当前节点是否是一个调用，并且这个调用是否是以 toga. 开头的方法。如果是，则将其添加到 widget\_calls 列表中。

**定义 analyze\_toga\_widgets 函数**

接受一个文件路径作为参数，读取该文件的内容。

使用 MetadataWrapper 包装 CST 树，以确保访问者可以获得所需的元数据。

创建 TogaWidgetFinder 的实例并访问包装后的 CST 树。

遍历 widget\_calls 列表，打印出找到的小部件创建信息。

**执行分析**

调用 analyze\_toga\_widgets('app.py') 对名为 app.py 的文件中的 Toga 小部件创建进行分析。

**三、关键功能**

**小部件创建检测**：

能够识别并记录所有通过调用 toga. 开头的方法创建的小部件实例。

**输出结果**：

分析完成后，会打印出所有找到的小部件创建语句，方便开发者了解哪些小部件被创建以及它们是如何被初始化的。

**四、应用案例**

假设有一个包含 Toga 应用程序的 Python 文件 app.py，通过运行 libc.py 可以自动检测并列出文件中所有的 Toga 小部件创建语句。这对于维护现有代码或重构用户界面非常有用，因为它可以帮助开发者快速定位和理解小部件的使用情况。

* + 。

**五、运行结果(部分)**

文本

描述已自动生成

## **2.app.py 中 pysnooper 的使用分析**

**一、项目背景**

在 app.py 文件中，引入了 pysnooper 库，并将其用作装饰器来跟踪 HelloWorld 类中的方法。

**二、技术实现**

用 @snoop() 装饰器修饰其中的方法。每当这些方法被执行时，pysnooper 将自动记录方法内部的每一行代码执行情况，以及任何局部变量的变化。

**三、功能**

**自动化日志记录**：

当装饰的方法被执行时，pysnooper 自动生成详细的日志文件（默认情况下是标准输出），记录每次调用时的参数值、返回值及局部变量的状态变化。

**简化错误排查**：

通过查看生成的日志，可以更容易地发现潜在的问题点，比如意外的变量值或者未预见的代码路径，从而加快开发和调试的速度。

**增强代码可读性**：

日志可以帮助新成员快速上手，了解代码的工作原理，提高整体项目的可维护性。

**四、应用**

在 app.py 文件中，pysnooper 被用于跟踪两个关键方法：

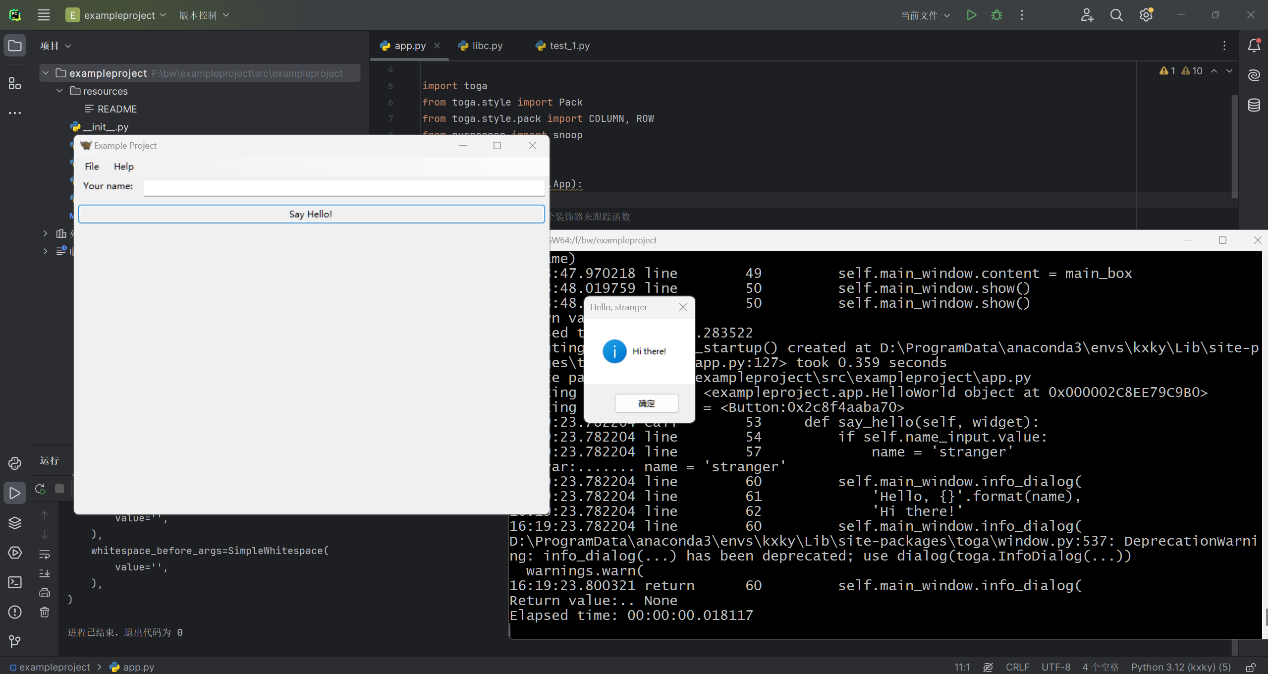
**startup 方法**：

此方法负责设置应用程序的初始界面，包括创建窗口、布局等。使用 pysnooper 可以帮助确保所有组件都按预期初始化，并且可以监控可能影响启动过程的任何异常行为。

**say\_hello 方法**：

用户交互的主要入口点之一，当用户点击按钮时触发。pysnooper 日志可以帮助确认输入处理是否正确，以及对话框是否如预期弹出。

**五、运行结果（部分）**

****

**电脑萤幕的截图

描述已自动生成**

**电脑萤幕的截图

描述已自动生成**

## **3.test\_1.py 文件**

**一、概述**

test\_1.py 是一个测试文件，使用 Python 的 unittest 框架和 unittest.mock 模块来验证 app.py 中定义的 Toga 应用程序 HelloWorld 的行为。该文件通过模拟对象和方法来确保可以在没有图形界面的情况下运行测试，并且能够准确地验证应用程序逻辑。

**二、主要功能**

**模拟主窗口 (DummyMainWindow 类)**：

创建了一个继承自 toga.MainWindow 的类，重写了 info\_dialog 方法以记录对话框调用而不是实际显示它们。这允许在单元测试中检查对话框是否按预期被调用。

**初始化测试环境 (setUp 方法)**：

在每个测试开始前设置好测试环境，包括创建 HelloWorld 应用实例，并将自定义的 DummyMainWindow 设置为主窗口。

**测试用户交互**：

使用 MagicMock 模拟用户输入的名字为空或非空两种情况，进而测试不同情况下 say\_hello 方法的行为。

**验证函数输出**：

使用 assertEqual 断言检查 DummyMainWindow 的 \_info\_dialog\_calls 列表中的最后一条记录是否符合预期，确保 say\_hello 方法按照预期调用了 info\_dialog 并传递了正确的参数。

**三、测试案例**

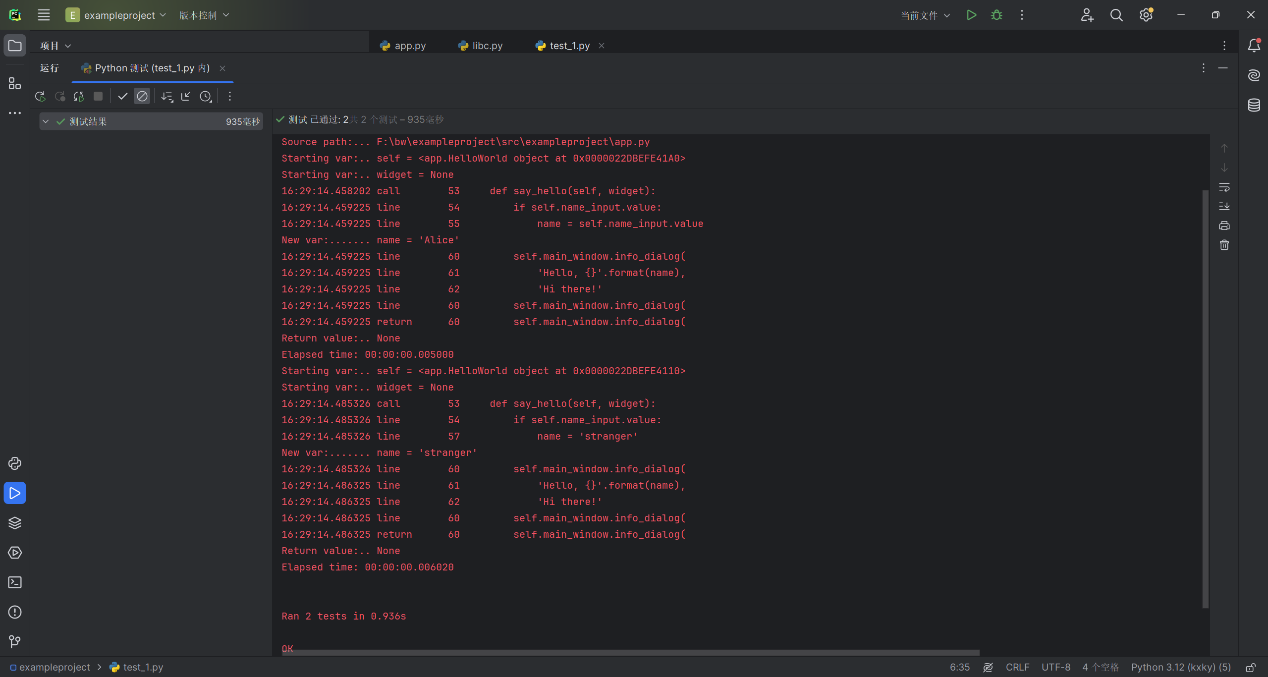
**test\_say\_hello\_with\_name**：

验证当用户输入名字（如 "Alice"）时，点击按钮后正确显示问候信息：“Hello, Alice”。

**test\_say\_hello\_without\_name**：

验证当用户未输入名字时，点击按钮后正确显示默认问候信息：“Hello, stranger”。

**四、结果**

****

## **4.总结**

以上测试结果显示beeware的toga工具包核心功能按预期工作，并未发现bug