**复旦大学计算机科学技术学院**

**2024-2025学年第一学期期末论文课程评分表**

**课程名称： 计算机可视化 课程代码：COMP130174.01**

**开课院系： 计算机科学技术学院**

**学生姓名：洪成勋 学号：22300240021 专业：计算机科学与技术**

**论文名称：Github关注量Top200用户数据的可视化分析及实现**

|  |
| --- |
| **论文评语（教师填写）：**  **任课教师签名：**  **日 期：** |

**成绩：**

**Github关注量Top200用户数据的可视化分析及实现**

洪成勋 22300240021

【摘要】

随着开源文化在全球范围内蓬勃发展，Github 作为最大的开源代码托管平台之一，其用户数据蕴含着丰富的信息。本设计聚焦于 Github 关注量 Top200 用户，旨在深入挖掘与剖析这些高关注度用户群体的特征及行为模式。通过自主编写的网络爬虫工具精准地从 Github 官网抓取关键个人信息，涵盖用户名、地理位置、粉丝数、贡献数等多元维度，并基于数据清洗与整理流程，运用 Python 数据处理库、AI代码助手Cursor及专业可视化工具，精心绘制一系列丰富且极具洞察性的图表，如全球视野下的用户分布地图、多变量对比的柱状与折线图、关联关系映射的桑基图和旭日图等。设计成果不仅直观呈现高关注用户的静态画像，更动态揭示其在年份、语言使用及组织参与等维度的演变趋势，可为开源社区生态研究、开发者行为分析及平台优化策略制定提供全新视角与量化依据。

（本文相关代码仓库：https://github.com/hcxxxxxx/Visualization）

【正文】

1. 相关技术与理论基础
2. 数据获取机制

该部分旨在精准获取 Github 关注量前 200 用户的详细个人信息，为后续深入的可视化分析奠定坚实的数据基础。数据获取过程综合运用了多种技术手段，包括 requests 库、BeautifulSoup 库以及 Selenium 库，巧妙协同以克服 Github 页面动态加载及数据分散性难题，确保信息抓取的完整性与准确性。

数据抓取过程集成在fetch.py文件中，抓取步骤大致分为：目标URL构建与请求，页面解析与用户信息初筛，动态页面加载与关键数据抓取，以及补充信息采集与数据整合。抓取的数据放入文件person\_data\_top200.json中。

1. 可视化技术选型与原理

本设计对比了主流可视化库（Matplotlib、Seaborn、Plotly 等）的优劣势，最终依据不同图表类型（地理、关系、分布等）选定了Streamlit作为主框架，并结合了Pandas、Plotly绘制可视化图像。

可视化类型选择方面，本设计针对各类数据信息的关系和特性，选择了不同的图表结构和可视化形式。例如，采用柱形图、折线图阐述粉丝数与贡献量关联模式、不同年份用户流入流出趋势等；采用桑基图展示组织-语言资源流动原理、旭日图嵌套结构呈现国家-语言层次关系、世界地图展示热点用户分布；采用3D坐标图布局用户多属性向量，对用户数据进行多维观察和分析。

1. AI代码助手

本设计中全程借助AI代码助手Cursor进行智能代码推荐、AI问答以及代码调试和修改。部分关键对话与代码段落将在后文中呈现。

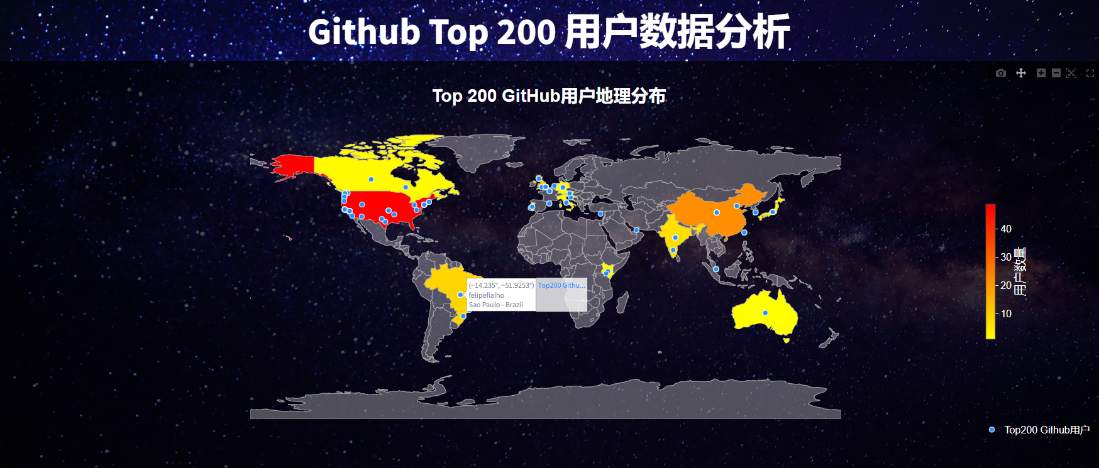
1. 数据收集与预处理
2. 数据来源

本研究的数据来源于对Github平台的爬取，获取了关注量Top200用户的多方面信息，包括但不限于用户名称、关注者数量、贡献数、仓库数量、编程语言、加入年份、地理位置等。

1. 数据清洗
2. 缺失值处理：对数据集中存在的缺失值进行识别和处理。对于部分关键信息缺失严重的记录，考虑予以删除；对于部分可补充或不影响整体分析的缺失值，采用合理的填充策略，如使用平均值、中位数或特定的默认值进行填充。
3. 异常值处理：通过数据分布分析和业务规则判断，识别并处理异常值。例如，对于明显偏离正常范围的关注者数量或贡献数，检查数据的准确性，如确为异常则进行修正或删除。
4. 数据格式转换：将日期格式的数据转换为统一的格式，以便后续按时间维度进行分析；对数值型数据进行类型转换，确保数据的一致性和准确性，如将字符串类型的数字转换为合适的数值类型。
5. 数据转换
6. 地理信息处理：将用户的地理位置信息转换为经纬度坐标，以便在地图可视化中准确展示用户分布。同时，对国家和地区信息进行标准化处理，建立统一的映射关系，便于统计和分析用户的国家分布情况。
7. 数值转换：对关注者数量、贡献数等数值型数据进行单位统一和量级转换，如将以“k”“m”为单位的数值转换为具体数字，以便进行准确的计算和比较。
8. 可视化方法与实现
9. 可视化库的选择

本项目选用了Plotly和Streamlit等Python可视化库。Plotly提供了丰富的交互式图表类型，如地图、柱状图、饼图、散点图、桑基图、旭日图等，能够满足多维度数据可视化的需求，并且生成的图表具有良好的交互性和美观性。Streamlit则用于构建Web应用，方便将可视化结果以直观的界面形式展示出来，同时提供了简洁的开发方式，便于快速迭代和部署。

1. 具体可视化图表及实现
2. 世界地图（chart\_01）

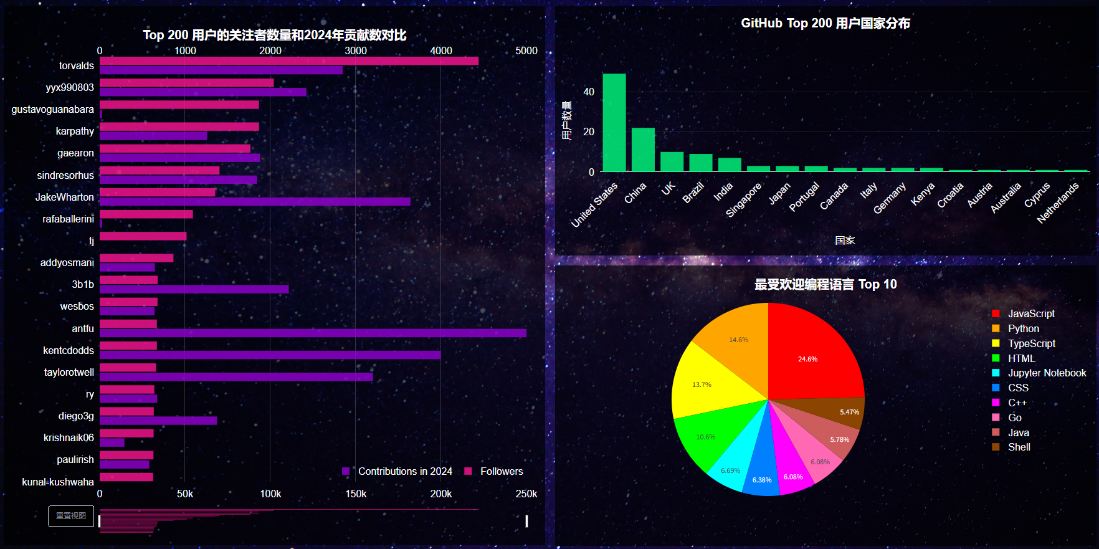
·功能：展示Top200 Github用户在全球的地理分布，通过热力图直观呈现不同国家和地区的用户数量分布情况，同时在地图上标记用户的具体位置。·实现方式：使用Plotly的Choropleth地图类型，根据用户所在国家的统计数据绘制热力图，颜色深浅表示用户数量的多少。通过自定义颜色比例尺，从黄色到红色渐变，清晰展示用户数量的相对差异。对于用户位置标记，使用Scattergeo类型，以蓝色标记点显示用户所在经纬度位置，并在悬停时显示用户姓名和具体位置信息。

·分析：

(chart1：世界地图)

1. 关注者数量和贡献数对比（chart\_02）

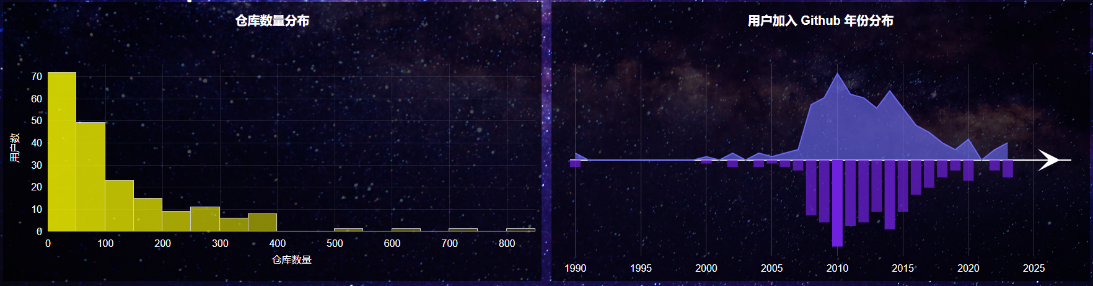
·功能：以水平双向柱状图对比Top200用户的关注者数量和2024年贡献数，帮助分析用户在社交影响力和近期贡献活跃度方面的关系。·实现方式：使用Plotly的Bar图表类型，分别绘制关注者数量和贡献数的柱状图，通过设置不同的颜色和x轴范围，清晰区分两者。为了增强交互性，添加了底部滑块，方便用户缩放查看数据范围；同时设置了重置按钮，使用户能够快速恢复初始视图。

1. 用户国家分布（chart\_03）·功能：以柱状图展示Github Top200用户的国家分布情况，直观呈现不同国家的用户数量占比。·实现方式：统计每个国家的用户数量，使用Plotly的Bar图表绘制柱状图，x轴为国家名称，y轴为用户数量。通过设置x轴标签倾斜角度，避免标签重叠；添加网格线，增强图表可读性。
2. 最受欢迎语言Top10（chart\_04）·功能：以饼图展示Top200用户中最受欢迎的前10种编程语言及其占比，反映用户的技术偏好。·实现方式：对用户使用的编程语言进行统计，选取使用频率最高的前10种语言，使用Plotly的Pie图表绘制饼图，每种语言使用不同的鲜艳颜色区分，使图表更加直观。

（chart2：关注者数量和贡献数对比

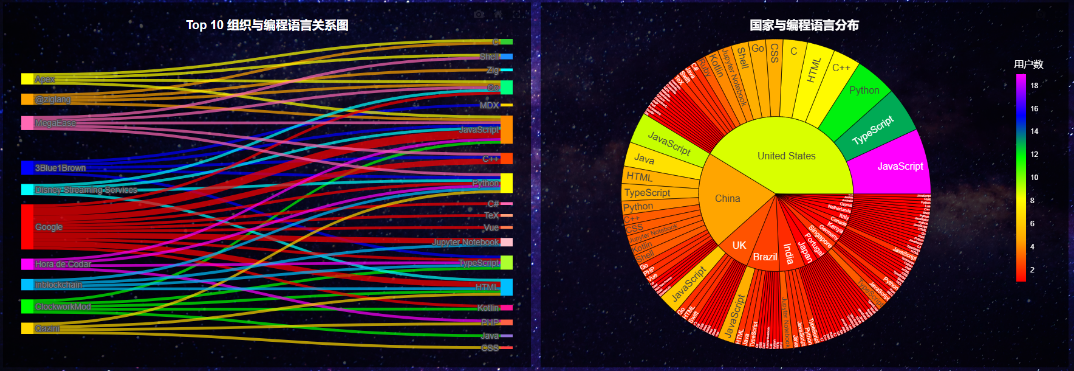
chart3：用户国家分布

chart4：最受欢迎语言Top10）

1. 仓库数量分布（chart\_05）·功能：以直方图展示Top200用户的仓库数量分布情况，帮助了解用户在项目创建方面的活跃度差异。·实现方式：使用Plotly的Histogram图表类型，将仓库数量数据划分为多个区间（bins），统计每个区间内的用户数量，以不同透明度的黄色标记直方图的柱子，展示仓库数量的分布趋势。
2. 加入Github年份分布（chart\_06）·功能：通过折线图和矩形图组合的方式展示用户加入Github的年份分布，包括每年加入的用户数量以及总体趋势，同时通过颜色深浅表示用户数量的相对多少。·实现方式：统计每年加入Github的用户数量，使用Plotly的Scatter和Scatter（filled）图表类型，绘制折线图表示趋势，以不同颜色和透明度的矩形填充表示每年的用户数量，颜色越深表示用户数量越多。添加箭头指示趋势方向，增强图表的可视化效果。

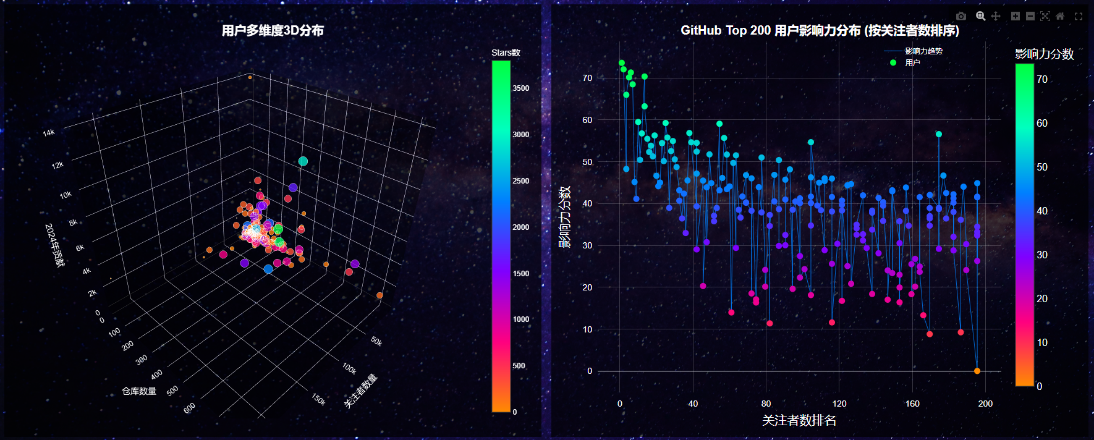
（chart5：仓库数量分布

chart6：加入Github年份分布）

1. Top10组织与编程语言关系（chart\_07）·功能：以桑基图展示Top10组织与编程语言之间的关系，包括组织使用的编程语言种类以及编程语言在不同组织中的应用情况，同时通过颜色表示节点的连接数多少。·实现方式：统计Top10组织使用的编程语言及相关数量，构建桑基图的节点和链接数据。根据节点的连接数计算颜色深浅，使用Plotly的Sankey图表类型绘制桑基图，组织节点和语言节点使用不同的颜色系列区分，链接使用半透明颜色显示，增强图表的层次感和可读性。
2. 国家与编程语言分布（chart\_08）·功能：以旭日图展示不同国家的用户使用编程语言的分布情况，呈现国家与编程语言之间的层次关系和占比。·实现方式：统计每个国家用户使用的编程语言及其数量，使用Plotly的Sunburst图表类型绘制旭日图，根据用户数量设置颜色渐变，从红色到紫色，展示不同国家和编程语言组合的占比情况。通过自定义悬停信息样式，在鼠标悬停时显示详细信息。

（chart7：Top10组织与编程语言关系

chart8：国家与编程语言分布）

1. 用户多维度3D分布（chart\_09）·功能：以3D散点图展示用户在关注者数量、仓库数量、2024年贡献数和Stars数四个维度上的分布情况，帮助分析用户在多个方面的综合特征。·实现方式：将相关数据转换为数值型，使用Plotly的Scatter3d图表类型绘制3D散点图，每个点的位置表示用户在三个维度上的数值，点的大小根据Stars数进行对数缩放，颜色根据Stars数设置渐变，从橙色到绿色，同时添加颜色条方便用户理解颜色与Stars数的对应关系。通过设置相机视角，提供最佳的可视化效果。
2. 用户影响力分布（chart\_10）·功能：通过计算用户的综合影响力分数（考虑关注者数量、贡献数、仓库数量和注册年限等因素），以趋势图展示用户影响力随关注者排名的变化情况，同时以散点图展示每个用户的影响力分数和排名。·实现方式：定义函数计算影响力分数，对数据进行归一化处理，计算加权总分得到影响力分数。使用Plotly的Scatter图表类型绘制趋势线和散点图，趋势线展示整体影响力趋势，散点颜色根据影响力分数设置渐变，同时添加颜色条显示分数范围，悬停时显示用户详细信息。

（chart9：用户多维度3D分布

chart10：用户影响力分布）

1. 可视化界面设计与交互功能
2. 界面布局：使用Streamlit构建Web应用，通过创建多列布局（如两列或多列布局），合理安排各个可视化图表的展示位置，使页面布局简洁美观，便于用户查看和比较不同图表。
3. 交互功能·悬停提示：在图表中设置悬停提示功能，当用户鼠标悬停在数据点、柱状图、地图标记等元素上时，显示详细的信息，如用户名称、具体数值、地理位置等，帮助用户更好地理解数据。·缩放和平移：对于部分图表（如地图、折线图等），启用缩放和平移功能，用户可以通过鼠标滚轮或手势操作，放大或缩小图表，查看局部细节或整体趋势；平移图表可以查看不同区域的数据。·筛选和排序：提供筛选和排序功能，用户可以根据自己的需求选择特定的数据范围或对数据进行排序，例如按照关注者数量、贡献数等进行排序，以便更深入地分析数据。
4. 可视化结果分析与图表解读
5. 地理分布特征

从世界地图可视化结果可以看出，Github关注量Top200用户主要分布在少数几个国家，如美国、中国、英国、德国等科技发达地区，这些地区的用户数量较多，在地图上呈现明显的热力集中区域。同时，通过用户位置标记可以发现，部分城市（如旧金山、纽约、北京、上海等）聚集了较多的高关注量用户，可能与当地的科技产业发展、教育资源丰富以及开源文化氛围浓厚等因素有关。

1. 关注者与贡献数关系

关注者数量和贡献数对比图表显示，部分用户关注者数量众多，但2024年贡献数相对较少，可能是一些知名开发者或项目负责人，其影响力主要体现在社交方面；而另一些用户虽然关注者数量较少，但贡献数较高，可能是活跃的开发者，专注于实际代码贡献。整体趋势上，关注者数量和贡献数之间存在一定的正相关关系，但并非完全线性相关，说明在Github社区中，用户的影响力来源具有多样性。

1. 用户国家分布情况

用户国家分布柱状图清晰展示了不同国家在Top200用户中的占比情况。美国用户占比较大，反映了其在全球开源领域的主导地位；中国紧随其后，显示中国开发者在Github上的活跃度和影响力不断提升。其他国家如英国、德国、印度等也有一定比例的用户，体现了全球范围内开源社区的多元性。

1. 编程语言偏好

最受欢迎语言Top10饼图揭示了当前Github用户群体中流行的编程语言。例如，某些语言在多个领域广泛应用，其在Top200用户中的使用频率较高；而一些新兴语言可能在特定领域或小众群体中受到青睐，也在图表中有所体现。通过分析编程语言偏好，有助于开发者了解行业技术趋势，选择合适的编程语言进行学习和项目开发。

1. 仓库数量分布

仓库数量分布直方图展示了用户在项目创建方面的活跃度差异。大部分用户的仓库数量集中在一定范围内，少数用户拥有较多的仓库，可能是经验丰富、项目众多的开发者或开源项目团队。该分布情况有助于了解Github平台上项目创建的整体规模和活跃度层次结构。

1. 加入Github年份趋势

加入Github年份分布图表呈现了用户加入平台的历史趋势。早期加入的用户数量相对较少，随着时间推移，用户数量逐渐增加，近年来增速有所变化，可能与开源文化的普及、行业发展趋势以及Github平台自身的推广和优化等因素相关。通过分析加入年份趋势，平台管理者可以评估不同阶段的用户增长情况，制定相应的发展策略。

1. 组织与编程语言关系

Top10组织与编程语言关系的桑基图展示了组织与编程语言之间的复杂关系。一些大型组织使用多种编程语言，体现了其在多技术领域的布局和项目需求多样性；同时，某些编程语言在多个组织中广泛应用，表明其通用性和重要性。该可视化结果有助于组织了解自身技术栈与行业趋势的匹配度，以及不同组织之间在技术选择上的共性和差异。

1. 国家与编程语言分布

国家与编程语言分布旭日图呈现了不同国家用户在编程语言使用上的偏好差异。一些国家在特定编程语言上具有较高的使用比例，可能与该国的教育体系、产业结构以及技术发展方向有关。例如，某些国家在新兴技术领域的编程语言应用较为领先，而另一些国家则在传统领域的编程语言使用上更为集中。通过分析该图表，能够深入了解全球范围内不同国家在开源技术领域的特点和优势。

1. 用户多维度综合特征

用户多维度3D分布散点图展示了用户在关注者数量、仓库数量、2024年贡献数和Stars数四个维度上的综合分布情况。可以观察到不同用户在各个维度上的表现差异，一些用户在多个维度上都表现出色，位于图表的特定区域；而另一些用户可能在某个维度上具有突出优势，但在其他维度上相对较弱。通过该可视化，能够更全面地评估用户的综合影响力和活跃度，为识别核心用户、发现潜在优秀开发者提供依据。

1. 用户影响力分布

用户影响力分布趋势图和散点图结合，清晰展示了用户影响力随关注者排名的变化情况。影响力分数较高的用户通常位于趋势线的上方，他们在关注者数量、贡献数、仓库数量和注册年限等方面综合表现优秀；而影响力分数较低的用户则分布在趋势线下方，可能需要在某些方面进一步提升。通过分析该图表，能够深入了解Github社区中用户影响力的分布结构，为平台制定用户激励机制、促进社区发展提供参考。

1. 关键AI对话内容

（注：下文“对话”中“A”表示Author，“C”表示Cursor Chat）

1. 初始设计

对话：

A：同一目录下有一个文件@person\_data\_top200.json，内容是Github关注量排名top200的个人用户信息，请据此在test.py中编写代码，实现Github关注度Top200个人用户信息数据大屏。数据大屏应包含以下信息（图表）：  
Chart1: 水平双向柱状图，followers & contributions（展示关注量排名和贡献情况，降序，可滑动）  
Chart2: 世界地图，loc（展示Top200用户地区分布，坐标和热点图）  
Chart3: 饼状图，languages（展示最热门的语言分布和比例）  
Chart4: 饼状图，is\_pro（展示Github Pro升级情况）  
Chart5: 柱状图，repo\_num（展示Top200用户仓库数量）  
Chart6: 柱状图，join\_date（展示Top200用户加入Github的年限情况）

C：以下是一个示例代码，展示如何在 test.py 中实现您所要求的 GitHub 关注度 Top 200 个人用户信息数据大屏。请确保您已经安装了所需的库，例如 pandas、matplotlib 和 plotly。

# test.py

import json

import pandas as pd

import matplotlib.pyplot as plt

import plotly.express as px

# 读取数据

with open('person\_data\_top200.json', 'r', encoding='utf-8') as f:

data = json.load(f)

# 将数据转换为 DataFrame

df = pd.DataFrame(data)

# Chart 1: 水平双向柱状图，followers & contributions

def chart1():

df['followers'] = df['followers'].str.replace('k', '').astype(float) \* 1000

df['contributions\_in\_2024'] = df['contributions\_in\_2024'].astype(int)

df\_sorted = df.sort\_values(by=['followers', 'contributions\_in\_2024'], ascending=False)

plt.figure(figsize=(10, 6))

plt.barh(df\_sorted['name'], df\_sorted['followers'], color='blue', label='Followers')

plt.barh(df\_sorted['name'], df\_sorted['contributions\_in\_2024'], color='orange', label='Contributions', alpha=0.5)

plt.xlabel('Count')

plt.title('Top 200 GitHub Users: Followers & Contributions')

plt.legend()

plt.show()

# Chart 2: 世界地图，loc

def chart2():

fig = px.choropleth(df, locations="loc", locationmode='country names',

title='Top 200 Users Location Distribution')

fig.show()

# Chart 3: 饼状图，languages

def chart3():

languages = df['languages'].explode().value\_counts()

plt.figure(figsize=(8, 8))

plt.pie(languages, labels=languages.index, autopct='%1.1f%%', startangle=140)

plt.title('Most Popular Languages Distribution')

plt.axis('equal')

plt.show()

# Chart 4: 饼状图，is\_pro

def chart4():

is\_pro\_counts = df['is\_pro'].value\_counts()

plt.figure(figsize=(8, 8))

plt.pie(is\_pro\_counts, labels=is\_pro\_counts.index, autopct='%1.1f%%', startangle=140)

plt.title('GitHub Pro Upgrade Status')

plt.axis('equal')

plt.show()

# Chart 5: 柱状图，repo\_num

def chart5():

df['repo\_num'] = df['repo\_num'].astype(int)

plt.figure(figsize=(10, 6))

plt.bar(df['name'], df['repo\_num'], color='green')

plt.xlabel('User')

plt.ylabel('Number of Repositories')

plt.title('Top 200 Users Repository Count')

plt.xticks(rotation=90)

plt.show()

# Chart 6: 柱状图，join\_date

def chart6():

df['join\_date'] = df['join\_date'].astype(int)

join\_counts = df['join\_date'].value\_counts().sort\_index()

plt.figure(figsize=(10, 6))

plt.bar(join\_counts.index, join\_counts.values, color='purple')

plt.xlabel('Join Year')

plt.ylabel('Number of Users')

plt.title('Top 200 Users Join Year Distribution')

plt.show()

# 调用所有图表函数

chart1()

chart2()

chart3()

chart4()

chart5()

chart6()

代码说明：

数据读取：从 person\_data\_top200.json 文件中读取数据并转换为 Pandas DataFrame。

图表绘制：

Chart 1：使用 Matplotlib 绘制水平双向柱状图，展示关注量和贡献情况。

Chart 2：使用 Plotly 绘制世界地图，展示用户地区分布。

Chart 3：绘制饼状图，展示最热门的编程语言分布。

Chart 4：绘制饼状图，展示 GitHub Pro 升级情况。

Chart 5：绘制柱状图，展示用户的仓库数量。

Chart 6：绘制柱状图，展示用户加入 GitHub 的年限情况。

请根据您的需求调整图表的样式和参数。确保在运行代码之前安装所需的库。

A：改善以下问题：1.Github Pro比例饼图，将文本“true”和“false”更改为“Github Pro用户”和“非Github Pro用户”（该部分图表在后续修改中已经删去）；2.仓库数量分布和用户加入Github年份分布，柱状图的柱形分别用多样不同的颜色标注，以区分开相邻的列。

C：好的，我来修改这些图表的显示效果。

# ... 前面的代码保持不变 ...

with col4:

# Chart4: Github Pro比例饼图

pro\_counts = df['is\_pro'].value\_counts()

pro\_counts.index = ['Github Pro用户' if x else '非Github Pro用户' for x in pro\_counts.index]

fig4 = px.pie(values=pro\_counts.values,

names=pro\_counts.index,

title='Github Pro 用户比例')

st.plotly\_chart(fig4, use\_container\_width=True)

# 创建最后两列布局

col5, col6 = st.columns(2)

with col5:

# Chart5: 仓库数量分布

fig5 = px.histogram(df,

x='repo\_num',

title='仓库数量分布',

nbins=30,

color\_discrete\_sequence=px.colors.qualitative.Set3) # 使用Set3配色方案

fig5.update\_traces(marker\_line\_width=1,

marker\_line\_color="white") # 添加白色边框分隔

st.plotly\_chart(fig5, use\_container\_width=True)

with col6:

# Chart6: 加入年份分布

fig6 = px.histogram(df,

x='join\_year',

title='用户加入 Github 年份分布',

color\_discrete\_sequence=px.colors.qualitative.Set1) # 使用Set1配色方案

fig6.update\_traces(marker\_line\_width=1,

marker\_line\_color="white") # 添加白色边框分隔

st.plotly\_chart(fig6, use\_container\_width=True)

# ... 其他代码保持不变 ...

A：在第一张图“Top 20用户的关注者数量和2024年贡献数对比”中，请将参与绘图的用户数由20增加至200，图片区域比例不变，而是可以放大缩小，上下拖动。同时，使用关注者数量（followers）作为key将用户进行降序排列。你也可以不必重新排列，因为在@person\_data\_top200.json文件中用户数据就是用followers来降序排列的。同时我注意到你没有翻译followers的数据，例如，json文件中记录followers数量为88.3k，你需要将其转换为883000来确定柱状图的长度。

C：好的，我来修改代码以处理这些问题。主要需要处理 followers 数据的单位转换，并调整图表的显示方式。

# ... 前面的代码保持不变 ...

@st.cache\_data

def load\_data():

with open('person\_data\_top200.json', 'r', encoding='utf-8') as f:

data = json.load(f)

df = pd.DataFrame(data)

# 处理 followers 数据

def convert\_followers(value):

if isinstance(value, str):

if 'k' in value.lower():

return float(value.lower().replace('k', '')) \* 1000

elif 'm' in value.lower():

return float(value.lower().replace('m', '')) \* 1000000

return float(value)

df['followers\_count'] = df['followers'].apply(convert\_followers)

# 处理日期

df['join\_date'] = pd.to\_datetime(df['join\_date'])

df['join\_year'] = df['join\_date'].dt.year

# 处理地理位置

def get\_country\_code(location):

if pd.isna(location):

return None

for country in pycountry.countries:

if country.name.lower() in location.lower():

return country.alpha\_3

return None

df['country\_code'] = df['loc'].apply(get\_country\_code)

return df

df = load\_data()

# 创建两列布局

col1, col2 = st.columns(2)

with col1:

# Chart1: 水平双向柱状图

fig1 = go.Figure()

fig1.add\_trace(go.Bar(

y=df['name'], # 使用所有200个用户

x=df['followers\_count'], # 使用转换后的followers数值

name='Followers',

orientation='h'

))

fig1.add\_trace(go.Bar(

y=df['name'], # 使用所有200个用户

x=df['contributions\_in\_2024'],

name='Contributions in 2024',

orientation='h'

))

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

barmode='group',

height=800, # 增加默认高度

yaxis={'tickfont': {'size': 10}}, # 调整y轴标签字体大小

showlegend=True,

legend={'orientation': 'h', 'yanchor': 'bottom', 'y': 1.02, 'xanchor': 'right', 'x': 1},

margin={'l': 20, 'r': 20, 't': 40, 'b': 20}

)

# 添加滚动条

fig1.update\_layout(

yaxis=dict(

rangeslider=dict(visible=True),

type='category' # 使用category类型以保持标签顺序

)

)

st.plotly\_chart(fig1, use\_container\_width=True)

# ... 后面的代码保持不变 ...

主要修改：

添加了 followers 数据的单位转换函数，将 "k" 和 "m" 后缀转换为实际数值

创建了新的 'followers\_count' 列存储转换后的数值

扩展图表显示所有200个用户

调整了图表布局，添加了滚动条功能

优化了图表的显示效果（标签大小、图例位置等）

这些修改后：

图表会显示所有200个用户的数据

followers 数据会正确显示实际数值

可以通过滚动条上下浏览完整的数据

保持了原有的排序（因为数据本身已经按 followers 降序排列）

A：把chart1的降序排列改为升序排列。并且让每个用户柱形能够显示的更大，相应地，图表区域内显示的柱形变少，支持纵向滚动来查看更多的用户柱形。

C：好的，我来修改代码，将排序改为升序，并调整柱形的显示大小：

with col1:

# 对数据进行升序排序

df\_sorted = df.sort\_values('followers\_count', ascending=True).reset\_index(drop=True)

# Chart1: 水平双向柱状图

fig1 = go.Figure()

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['followers\_count'],

name='Followers',

orientation='h'

))

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['contributions\_in\_2024'],

name='Contributions in 2024',

orientation='h'

))

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

barmode='group',

height=800, # 保持总高度

yaxis=dict(

tickfont=dict(size=10),

range=[0, 15], # 初始只显示15个用户

autorange=False, # 关闭自动调整范围

showticklabels=True,

type="category",

scaleanchor="x",

constrain="domain"

),

xaxis=dict(

type="log",

showgrid=True

),

showlegend=True,

legend=dict(

orientation='h',

yanchor='bottom',

y=1.02,

xanchor='right',

x=1

),

margin=dict(l=20, r=20, t=40, b=20),

dragmode='pan', # 启用拖拽模式

bargap=0.2, # 增加柱子之间的间距

bargroupgap=0.1 # 增加组内柱子的间距

)

# 添加按钮来控制缩放和平移

fig1.update\_layout(

updatemenus=[

dict(

type="buttons",

showactive=False,

buttons=[

dict(

label="重置视图",

method="relayout",

args=[{"yaxis.range": [0, 15]}] # 重置为显示15个用户

)

],

pad={"r": 10, "t": 10},

x=0.1,

y=1.1

)

]

)

st.plotly\_chart(fig1, use\_container\_width=True)

主要修改：

添加了数据升序排序

减少了初始显示的用户数量（从20个改为15个）

设置 autorange=False 以固定初始显示范围

添加了 bargap 和 bargroupgap 参数来调整柱形的间距

调整了重置按钮的显示范围

这样修改后：

数据将按照 followers\_count 升序排列

柱形会显示得更大，更清晰

初始只显示15个用户的数据

可以通过拖拽和滚轮来查看更多用户数据

点击重置按钮可以回到初始视图

A：按照以下要求修改代码：1.chart1固定x轴初始比例，使得左侧为0，右侧最远为222k（即222000）；2.固定柱形大小，使得chart1图像区域显示20个用户数据，而可以上下滚动查看更多。柱形的左右长度可以通过下方的拉杆来调整比例，可以变得更短但不能超出显示范围。

C：好的，我来修改代码以满足这些要求

with col1:

# 对数据进行升序排序

df\_sorted = df.sort\_values('followers\_count', ascending=True).reset\_index(drop=True)

# Chart1: 水平双向柱状图

fig1 = go.Figure()

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['followers\_count'],

name='Followers',

orientation='h'

))

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['contributions\_in\_2024'],

name='Contributions in 2024',

orientation='h'

))

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

barmode='group',

height=800,

yaxis=dict(

tickfont=dict(size=10),

range=[0, 20], # 固定显示20个用户

autorange=False,

showticklabels=True,

type="category",

scaleanchor="x",

constrain="domain",

fixedrange=False # 允许Y轴滚动

),

xaxis=dict(

range=[0, 222000], # 固定X轴范围

showgrid=True,

fixedrange=True, # 锁定X轴范围

rangemode='nonnegative' # 确保不会出现负值

),

showlegend=True,

legend=dict(

orientation='h',

yanchor='bottom',

y=1.02,

xanchor='right',

x=1

),

margin=dict(l=20, r=20, t=40, b=20),

dragmode='pan',

bargap=0.15, # 调整柱子间距

bargroupgap=0.1

)

# 添加底部滑块

fig1.update\_layout(

xaxis=dict(

rangeslider=dict(

visible=True,

thickness=0.05 # 调整滑块高度

)

)

)

# 添加重置按钮

fig1.update\_layout(

updatemenus=[

dict(

type="buttons",

showactive=False,

buttons=[

dict(

label="重置视图",

method="relayout",

args=[{

"yaxis.range": [0, 20],

"xaxis.range": [0, 222000]

}]

)

],

pad={"r": 10, "t": 10},

x=0.1,

y=1.1

)

]

)

st.plotly\_chart(fig1, use\_container\_width=True)

主要修改：

固定 X 轴范围为 [0, 222000]

使用 fixedrange=True 锁定 X 轴范围

固定显示 20 个用户数据

添加底部滑块用于调整数据显示比例

设置 rangemode='nonnegative' 确保不会出现负值

调整了柱形间距以获得更好的显示效果

这样修改后：

X 轴范围固定在 0-222k

始终显示 20 个用户数据

可以通过拖拽和滚轮上下查看更多用户

可以通过底部滑块调整数据显示比例

重置按钮可以恢复初始视图

A：chart1初始显示最顶部那一部分的用户，即关注数最多的用户；同时，横坐标“关注者数量”和“contributions\_in\_2024”分别使用两套横坐标，x轴处显示对应的两套横坐标，其中关注者数量的横坐标范围与之前保持一致，contributions\_in\_2024的横坐标范围为0到4000。

C：好的，我来修改代码以添加双 X 轴并调整初始显示位置：

with col1:

# 对数据进行升序排序

df\_sorted = df.sort\_values('followers\_count', ascending=True).reset\_index(drop=True)

# Chart1: 水平双向柱状图

fig1 = go.Figure()

# 添加 Followers 数据

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['followers\_count'],

name='Followers',

orientation='h',

xaxis='x' # 使用主x轴

))

# 添加 Contributions 数据

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['contributions\_in\_2024'],

name='Contributions in 2024',

orientation='h',

xaxis='x2' # 使用第二个x轴

))

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

barmode='group',

height=800,

yaxis=dict(

tickfont=dict(size=10),

range=[180, 200], # 初始显示最后20个用户（关注数最多的）

autorange=False,

showticklabels=True,

type="category",

constrain="domain",

fixedrange=False # 允许Y轴滚动

),

# 主x轴（Followers）

xaxis=dict(

range=[0, 222000],

showgrid=True,

fixedrange=True,

rangemode='nonnegative',

title='关注者数量',

side='bottom'

),

# 第二个x轴（Contributions）

xaxis2=dict(

range=[0, 4000],

showgrid=False,

fixedrange=True,

rangemode='nonnegative',

title='2024年贡献数',

side='top',

overlaying='x' # 覆盖在主x轴上

),

showlegend=True,

legend=dict(

orientation='h',

yanchor='bottom',

y=1.02,

xanchor='right',

x=1

),

margin=dict(l=20, r=20, t=40, b=20),

dragmode='pan',

bargap=0.15,

bargroupgap=0.1

)

# 添加底部滑块

fig1.update\_layout(

xaxis=dict(

rangeslider=dict(

visible=True,

thickness=0.05

)

)

)

# 添加重置按钮

fig1.update\_layout(

updatemenus=[

dict(

type="buttons",

showactive=False,

buttons=[

dict(

label="重置视图",

method="relayout",

args=[{

"yaxis.range": [180, 200], # 重置到顶部20个用户

"xaxis.range": [0, 222000],

"xaxis2.range": [0, 4000]

}]

)

],

pad={"r": 10, "t": 10},

x=0.1,

y=1.1

)

]

)

st.plotly\_chart(fig1, use\_container\_width=True)

主要修改：

添加了第二个 X 轴（xaxis2）用于显示贡献数

设置了不同的范围：

Followers: 0-222k

Contributions: 0-4k

将初始显示范围设置为 [180, 200]，这样会显示最后20个用户（关注数最多的）

调整了两个 X 轴的位置（一个在底部，一个在顶部）

更新了重置按钮的功能，使其能够同时重置所有轴的范围

这样修改后：

初始显示关注数最多的20个用户

两个指标使用不同的 X 轴和刻度

可以通过拖拽和滚轮上下查看更多用户

可以通过底部滑块调整数据显示比例

重置按钮可以恢复到初始视图

A：这段代码需要修改，目前是双x轴对应的柱形重合显示，我想要让它们并列显示，达到双x轴对照柱形图的效果。同时，在上方显示第二个x轴，对应contributions\_in\_2024，并且将横向范围设置为0到4000，调整contributions\_in\_2024柱形的比例。如何修改？

C：我明白了您的需求。我们需要让两组柱形并列显示，同时保持它们分别对应不同的 x 轴和比例。这里是修改后的代码：

with col1:

# 对数据进行升序排序

df\_sorted = df.sort\_values('followers\_count', ascending=True).reset\_index(drop=True)

# Chart1: 水平双向柱状图

fig1 = go.Figure()

# 添加 Followers 数据

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['followers\_count'],

name='Followers',

orientation='h'

))

# 添加 Contributions 数据，使用第二个x轴的比例

fig1.add\_trace(go.Bar(

y=df\_sorted['name'],

x=df\_sorted['contributions\_in\_2024'],

name='Contributions in 2024',

orientation='h',

marker\_color='darkgreen'

))

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

height=800,

yaxis=dict(

tickfont=dict(size=8),

range=[180, 200],

autorange=False,

showticklabels=True,

type="category",

constrain="domain",

fixedrange=False,

tickmode='linear',

dtick=1

),

xaxis=dict(

range=[0, 222000],

showgrid=True,

fixedrange=True,

rangemode='nonnegative',

title='关注者数量',

side='bottom'

),

xaxis2=dict(

range=[0, 4000],

showgrid=True,

fixedrange=True,

rangemode='nonnegative',

title='2024年贡献数',

side='top',

overlaying='x',

position=1,

anchor='free',

shift=0 # 调整轴的位置

),

showlegend=True,

legend=dict(

orientation='h',

yanchor='bottom',

y=1.02,

xanchor='right',

x=1

),

margin=dict(l=20, r=20, t=40, b=20),

dragmode='pan',

bargap=0.15,

bargroupgap=0.1,

barmode='group' # 确保柱形并列显示

)

# 添加底部滑块

fig1.update\_layout(

xaxis=dict(

rangeslider=dict(

visible=True,

thickness=0.05

)

)

)

# 添加重置按钮

fig1.update\_layout(

updatemenus=[

dict(

type="buttons",

showactive=False,

buttons=[

dict(

label="重置视图",

method="relayout",

args=[{

"yaxis.range": [180, 200],

"xaxis.range": [0, 222000],

"xaxis2.range": [0, 4000]

}]

)

],

pad={"r": 10, "t": 10},

x=0.1,

y=1.1

)

]

)

st.plotly\_chart(fig1, use\_container\_width=True)

主要修改：

移除了第二个 trace 中的 xaxis='x2'，让它默认使用主坐标轴

修改了 xaxis2 的设置，使用 anchor='free' 和 shift=0

保持 barmode='group' 确保柱形并列显示

保持双 x 轴的范围设置（0-222000 和 0-4000）

这样修改后应该能实现：

两组柱形并列显示

顶部显示 contributions 的刻度（0-4000）

底部显示 followers 的刻度（0-222000）

保持所有的交互功能（滚动、缩放、重置等）

A：下面修改graph2部分。person\_data\_top200中的loc字段为每个用户的地理位置，请据此在世界地图上用红色小圆点标注这200个用户的位置，loc值为null的不必标注。地图暂时不必着色。

C：好的，我们需要使用 Plotly 的 scattergeo 来在世界地图上标注用户位置。由于 loc 字段是城市或地区名称，我们需要先将其转换为经纬度坐标。这里我们可以使用 geopy 库来进行地理编码：

from geopy.geocoders import Nominatim

from geopy.exc import GeocoderTimedOut, GeocoderUnavailable

import time

with col2:

# 创建地理编码器

geolocator = Nominatim(user\_agent="my\_app")

# 获取非空地理位置的用户

users\_with\_loc = df[df['loc'].notna()]

# 存储经纬度结果

locations = []

# 获取每个位置的经纬度

for loc in users\_with\_loc['loc']:

try:

# 添加延时以避免超过API限制

time.sleep(1)

location = geolocator.geocode(loc)

if location:

locations.append({

'lat': location.latitude,

'lon': location.longitude,

'loc': loc

})

except (GeocoderTimedOut, GeocoderUnavailable):

continue

# 创建世界地图

fig2 = go.Figure(data=go.Scattergeo(

lon=[loc['lon'] for loc in locations],

lat=[loc['lat'] for loc in locations],

mode='markers',

marker=dict(

size=8,

color='red',

opacity=0.7

),

text=[loc['loc'] for loc in locations], # 添加悬停文本

hoverinfo='text'

))

# 更新布局

fig2.update\_layout(

title='GitHub Top 200用户地理分布',

geo=dict(

showframe=False, # 不显示边框

showcoastlines=True, # 显示海岸线

projection\_type='equirectangular' # 使用等矩形投影

),

height=600,

margin=dict(l=0, r=0, t=30, b=0)

)

# 显示地图

st.plotly\_chart(fig2, use\_container\_width=True)

主要功能：

使用 geopy 库将地理位置名称转换为经纬度坐标

跳过 loc 为 null 的用户

在世界地图上用红色小圆点标注用户位置

添加悬停效果显示具体位置

使用等矩形投影显示世界地图

不显示地图着色，只显示海岸线

注意事项：

需要安装 geopy 库：pip install geopy

地理编码过程可能需要一些时间，因为需要调用外部 API

添加了延时以避免超过 API 限制

处理了可能的地理编码错误

您可能需要根据实际情况调整：

圆点的大小（marker size）

圆点的透明度（opacity）

地图的投影方式（projection\_type）

地图的其他视觉效果

为了实现这个功能，我们需要先在数据处理部分添加经纬度转换的代码。请在 load\_data() 函数中添加以下代码：

@st.cache\_data

def load\_data():

# ... existing code ...

# 添加经纬度数据

from geopy.geocoders import Nominatim

from geopy.exc import GeocoderTimedOut

def get\_lat\_lon(location):

if pd.isna(location):

return None, None

try:

geolocator = Nominatim(user\_agent="github\_analysis")

location\_data = geolocator.geocode(location)

if location\_data:

return location\_data.latitude, location\_data.longitude

return None, None

except GeocoderTimedOut:

return None, None

# 获取经纬度

df['latitude'], df['longitude'] = zip(\*df['loc'].apply(get\_lat\_lon))

return df

这段代码会：

使用 geopy 库将位置字符串转换为经纬度坐标

在地图上用红色小圆点标注每个用户的位置

当鼠标悬停在点上时，显示用户名和位置信息

保持地图基础样式简洁，不对国家进行着色

注意：由于地理编码可能需要一些时间，第一次加载数据时可能会比较慢。数据会被缓存，之后的加载会更快。

A：出现以下问题

图形用户界面, 应用程序

描述已自动生成

C：这个错误是因为我们还没有成功创建经纬度数据。让我们修改一下代码，先创建一个简单的地理位置映射字典来处理常见城市/国家的经纬度：

@st.cache\_data

def load\_data():

with open('person\_data\_top200.json', 'r', encoding='utf-8') as f:

data = json.load(f)

df = pd.DataFrame(data)

# 处理 followers 数据

def convert\_followers(value):

if isinstance(value, str):

if 'k' in value.lower():

return float(value.lower().replace('k', '')) \* 1000

elif 'm' in value.lower():

return float(value.lower().replace('m', '')) \* 1000000

return float(value)

df['followers\_count'] = df['followers'].apply(convert\_followers)

# 处理日期

df['join\_date'] = pd.to\_datetime(df['join\_date'])

df['join\_year'] = df['join\_date'].dt.year

# 创建地理位置映射字典

location\_coords = {

'China': (35.8617, 104.1954),

'United States': (37.0902, -95.7129),

'San Francisco': (37.7749, -122.4194),

'New York': (40.7128, -74.0060),

'London': (51.5074, -0.1278),

'Singapore': (1.3521, 103.8198),

'India': (20.5937, 78.9629),

'Japan': (36.2048, 138.2529),

'Brazil': (-14.2350, -51.9253),

'Germany': (51.1657, 10.4515),

'Portland': (45.5155, -122.6789),

'Seattle': (47.6062, -122.3321),

'California': (36.7783, -119.4179),

'Beijing': (39.9042, 116.4074),

'Shanghai': (31.2304, 121.4737),

}

def get\_coordinates(location):

if pd.isna(location):

return None, None

# 遍历位置映射字典查找匹配

for key, coords in location\_coords.items():

if key.lower() in str(location).lower():

return coords

return None, None

# 获取经纬度

df['latitude'], df['longitude'] = zip(\*df['loc'].apply(get\_coordinates))

return df

# ... 其余代码保持不变 ...

这个修改：

创建了一个包含常见地理位置及其经纬度的字典

通过简单的字符串匹配来确定位置的经纬度

不再依赖外部地理编码服务，提高了加载速度和稳定性

这样应该可以解决 KeyError 的问题，并在地图上显示那些能够匹配到经纬度的位置点。如果某些位置无法匹配，它们将被跳过（返回 None, None）。

你可以根据需要继续扩充 location\_coords 字典，添加更多的地理位置映射。

A：根据@person\_data\_top200.json中的地点，为我添加更多的地理位置映射字典

C：我来分析 JSON 数据中的地点并扩充地理位置映射字典。以下是更新后的映射字典：

location\_coords = {

# 美国城市和地区

'United States': (37.0902, -95.7129),

'San Francisco': (37.7749, -122.4194),

'New York': (40.7128, -74.0060),

'Portland': (45.5155, -122.6789),

'Seattle': (47.6062, -122.3321),

'California': (36.7783, -119.4179),

'Mountain View': (37.3861, -122.0839),

'Pittsburgh': (40.4406, -79.9959),

'Salt Lake City': (40.7608, -111.8910),

'Little Rock': (34.7465, -92.2896),

'Austin': (30.2672, -97.7431),

'Cambridge': (42.3736, -71.1097),

'Virginia': (37.4316, -78.6569),

'Washington': (47.7511, -120.7401),

'Phoenix': (33.4484, -112.0740),

'Anaheim': (33.8366, -117.9143),

'Palo Alto': (37.4419, -122.1430),

'Southern Oregon': (42.8865, -122.8156),

'Texas': (31.9686, -99.9018),

# 中国城市和地区

'China': (35.8617, 104.1954),

'Beijing': (39.9042, 116.4074),

'Shanghai': (31.2304, 121.4737),

'Hangzhou': (30.2741, 120.1551),

'Wuxi': (31.4916, 120.3119),

'Chongqing': (29.4316, 106.9123),

'Hefei': (31.8206, 117.2272),

# 欧洲城市和国家

'London': (51.5074, -0.1278),

'Germany': (51.1657, 10.4515),

'Berlin': (52.5200, 13.4050),

'Austria': (47.5162, 14.5501),

'Portugal': (39.3999, -8.2245),

'Barcelona': (41.3851, 2.1734),

'Paris': (48.8566, 2.3522),

# 亚洲其他地区

'Singapore': (1.3521, 103.8198),

'Japan': (36.2048, 138.2529),

'India': (20.5937, 78.9629),

'Bangalore': (12.9716, 77.5946),

'Taiwan': (23.6978, 120.9605),

'Dubai': (25.2048, 55.2708),

'Korea': (35.9078, 127.7669),

# 其他国家和地区

'Brazil': (-14.2350, -51.9253),

'Rio de Janeiro': (-22.9068, -43.1729),

'Santa Catarina': (-27.2423, -50.2189),

'São Paulo': (-23.5505, -46.6333),

'Croatia': (45.1000, 15.2000),

'Australia': (-25.2744, 133.7751),

'New Zealand': (-40.9006, 174.8860),

'Canada': (56.1304, -106.3468),

'Ontario': (51.2538, -85.3232),

'Hamilton': (43.2557, -79.8711),

'Kenya': (0.0236, 37.9062),

'Nairobi': (-1.2921, 36.8219),

'Cyprus': (35.1264, 33.4299),

'Israel': (31.0461, 34.8516),

'Scotland': (56.4907, -4.2026),

'Ireland': (53.1424, -7.6921),

'Netherlands': (52.1326, 5.2913),

'Amsterdam': (52.3676, 4.9041),

'Italy': (41.8719, 12.5674),

'Catania': (37.5079, 15.0830),

'Bath': (51.3758, -2.3599),

'Lisbon': (38.7223, -9.1393),

'Oslo': (59.9139, 10.7522),

'Stockholm': (59.3293, 18.0686),

'Helsinki': (60.1699, 24.9384),

'Copenhagen': (55.6761, 12.5683),

'Brussels': (50.8503, 4.3517),

'Vienna': (48.2082, 16.3738),

'Prague': (50.0755, 14.4378),

'Warsaw': (52.2297, 21.0122),

'Budapest': (47.4979, 19.0402),

'Buenos Aires': (-34.6037, -58.3816),

'Mexico City': (19.4326, -99.1332),

'Santiago': (-33.4489, -70.6693)

}

这个扩充后的字典：

包含了更多在 JSON 数据中出现的城市和国家

按照地理区域进行了分组（美国、中国、欧洲等）

包含了主要城市和它们的精确坐标

涵盖了不同大洲的主要地区

这样应该能够匹配到更多用户的位置信息，使地图上的标注更加完整。如果你发现还有其他常见地点需要添加，可以继续扩充这个字典。

A：效果不错。现在为这个地图着色

C：我们可以为世界地图添加基础颜色。我会修改 Chart2 部分的代码，添加适当的地图底色：

with col2:

# Chart2: 世界地图

fig2 = go.Figure()

# 添加基础地图层，使用浅色调

fig2.add\_trace(go.Choropleth(

locations=['USA', 'GBR', 'CHN', 'RUS', 'BRA', 'AUS', 'IND', 'FRA', 'DEU', 'CAN'], # 国家代码

z=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], # 统一的值，用于生成统一的颜色

locationmode='ISO-3',

colorscale=[[0, '#E5ECF6'], [1, '#E5ECF6']], # 使用浅蓝灰色

showscale=False,

marker=dict(

line=dict(

color='rgb(180,180,180)',

width=0.5

)

)

))

# 添加用户位置标记

valid\_locations = df[df['loc'].notna()]

fig2.add\_trace(go.Scattergeo(

lon=valid\_locations['longitude'],

lat=valid\_locations['latitude'],

mode='markers',

marker=dict(

size=8,

color='red',

opacity=0.7,

line=dict(

color='rgb(40,40,40)',

width=0.5

)

),

hovertext=valid\_locations['name'] + '<br>' + valid\_locations['loc'],

name='GitHub用户'

))

fig2.update\_layout(

title='Top 200 GitHub用户地理分布',

showlegend=True,

geo=dict(

showframe=False,

showcoastlines=True,

projection\_type='equirectangular',

coastlinecolor='rgb(180,180,180)',

landcolor='rgb(217, 217, 217)', # 陆地颜色

oceancolor='rgb(239, 243, 255)', # 海洋颜色

showocean=True,

showcountries=True,

countrycolor='rgb(180,180,180)', # 国界线颜色

showlakes=True,

lakecolor='rgb(239, 243, 255)', # 湖泊颜色

showrivers=False

),

paper\_bgcolor='white', # 图表背景色

plot\_bgcolor='white' # 绘图区背景色

)

st.plotly\_chart(fig2, use\_container\_width=True)

这个更新：

为陆地添加了浅灰色

为海洋添加了浅蓝色

添加了国界线，使用灰色

优化了标记点的样式，添加了细微的边框

整体配色保持简洁清爽

保留了红色标记点的显著性

这样的配色方案应该能让地图看起来更加专业和美观，同时不会喧宾夺主，让用户位置标记依然清晰可见。

A：我注意到世界地图的下方有一块空白，我想在此处增加一个柱状统计图来统计这200位用户来自的国家，并且以国家作为x坐标，人数作为纵坐标，以人数降序排列。

C：我们可以在世界地图下方添加一个国家分布的柱状图。首先需要处理位置数据，将其归类到对应的国家，然后创建柱状图：

with col2:

# Chart2: 世界地图

# ... 保持原有的世界地图代码不变 ...

st.plotly\_chart(fig2, use\_container\_width=True)

# 添加国家分布统计图

# 创建国家映射字典

country\_mapping = {

'United States': ['USA', 'United States', 'California', 'San Francisco', 'New York', 'Portland',

'Seattle', 'Mountain View', 'Pittsburgh', 'Salt Lake City', 'Little Rock',

'Austin', 'Cambridge', 'Virginia', 'Washington', 'Phoenix', 'Anaheim',

'Palo Alto', 'Southern Oregon', 'Texas'],

'China': ['China', 'Beijing', 'Shanghai', 'Hangzhou', 'Wuxi', 'Chongqing', 'Hefei'],

'UK': ['United Kingdom', 'London', 'Bath', 'Scotland'],

'Germany': ['Germany', 'Berlin'],

'Brazil': ['Brazil', 'Rio de Janeiro', 'Santa Catarina', 'São Paulo'],

'India': ['India', 'Bangalore'],

'Japan': ['Japan'],

'Singapore': ['Singapore'],

'Canada': ['Canada', 'Ontario', 'Hamilton'],

'Netherlands': ['Netherlands', 'Amsterdam'],

'Portugal': ['Portugal', 'Lisbon'],

'Australia': ['Australia'],

'Israel': ['Israel'],

'Taiwan': ['Taiwan'],

'Austria': ['Austria', 'Vienna'],

'Croatia': ['Croatia'],

'Kenya': ['Kenya', 'Nairobi'],

'Cyprus': ['Cyprus'],

'Italy': ['Italy', 'Catania']

}

def get\_country(location):

if pd.isna(location):

return 'Unknown'

location = str(location).lower()

for country, cities in country\_mapping.items():

if any(city.lower() in location for city in cities):

return country

return 'Other'

# 统计国家分布

df['country'] = df['loc'].apply(get\_country)

country\_counts = df['country'].value\_counts()

# 创建柱状图

fig\_countries = go.Figure()

fig\_countries.add\_trace(go.Bar(

x=country\_counts.index,

y=country\_counts.values,

marker\_color='#3480b8' # 使用与之前图表相协调的颜色

))

fig\_countries.update\_layout(

title='GitHub Top 200 用户国家分布',

xaxis\_title='国家',

yaxis\_title='用户数量',

xaxis\_tickangle=-45, # 倾斜x轴标签以防重叠

height=400, # 控制图表高度

margin=dict(t=30, b=80), # 调整边距

paper\_bgcolor='white',

plot\_bgcolor='white',

showlegend=False

)

# 添加网格线

fig\_countries.update\_xaxes(showgrid=False)

fig\_countries.update\_yaxes(

showgrid=True,

gridwidth=1,

gridcolor='rgba(128,128,128,0.2)'

)

st.plotly\_chart(fig\_countries, use\_container\_width=True)

这段代码：

创建了一个国家映射字典，将城市和地区映射到对应的国家

定义了一个函数来处理位置信息，将其归类到对应的国家

统计每个国家的用户数量

创建了一个柱状图来展示国家分布

对图表进行了样式优化：

倾斜x轴标签避免重叠

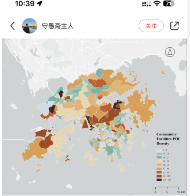
添加了网格线

使用了协调的颜色

调整了图表大小和边距

这样就能直观地展示用户的国家分布情况。如果你发现还有其他需要添加的国家映射，可以继续扩充 country\_mapping 字典。

A：请在世界地图中按照国家对应的用户数量对对应国家进行着色，并且该国家用户数越多颜色就越红热。参考此图片



C：我来修改世界地图的代码，添加基于用户数量的热力图着色效果

with col2:

# Chart2: 世界地图

fig2 = go.Figure()

# 准备国家代码映射

country\_code\_mapping = {

'United States': 'USA',

'China': 'CHN',

'UK': 'GBR',

'Germany': 'DEU',

'Brazil': 'BRA',

'India': 'IND',

'Japan': 'JPN',

'Singapore': 'SGP',

'Canada': 'CAN',

'Netherlands': 'NLD',

'Portugal': 'PRT',

'Australia': 'AUS',

'Israel': 'ISR',

'Taiwan': 'TWN',

'Austria': 'AUT',

'Croatia': 'HRV',

'Kenya': 'KEN',

'Cyprus': 'CYP',

'Italy': 'ITA'

}

# 获取国家统计数据

df['country'] = df['loc'].apply(get\_country)

country\_counts = df['country'].value\_counts()

country\_counts = country\_counts[~country\_counts.index.isin(['Other', 'Unknown'])]

# 准备热力图数据

locations = []

z\_values = []

for country, count in country\_counts.items():

if country in country\_code\_mapping:

locations.append(country\_code\_mapping[country])

z\_values.append(count)

# 添加基础地图层，使用热力图着色

fig2.add\_trace(go.Choropleth(

locations=locations,

z=z\_values,

locationmode='ISO-3',

colorscale=[

[0, 'rgb(242,240,247)'], # 最浅色

[0.2, 'rgb(218,218,235)'],

[0.4, 'rgb(188,189,220)'],

[0.6, 'rgb(158,154,200)'],

[0.8, 'rgb(117,107,177)'],

[1, 'rgb(84,39,143)'] # 最深色

],

colorbar\_title='用户数量',

showscale=True,

marker=dict(

line=dict(

color='rgb(180,180,180)',

width=0.5

)

)

))

# 添加用户位置标记

valid\_locations = df[df['loc'].notna()]

fig2.add\_trace(go.Scattergeo(

lon=valid\_locations['longitude'],

lat=valid\_locations['latitude'],

mode='markers',

marker=dict(

size=8,

color='red',

opacity=0.7,

line=dict(

color='rgb(40,40,40)',

width=0.5

)

),

hovertext=valid\_locations['name'] + '<br>' + valid\_locations['loc'],

name='GitHub用户'

))

fig2.update\_layout(

title='Top 200 GitHub用户地理分布',

showlegend=True,

height=600,

geo=dict(

showframe=False,

showcoastlines=True,

projection\_type='equirectangular',

coastlinecolor='rgb(180,180,180)',

showocean=True,

oceancolor='rgb(239, 243, 255)',

showcountries=True,

countrycolor='rgb(180,180,180)',

showlakes=True,

lakecolor='rgb(239, 243, 255)',

showrivers=False

),

paper\_bgcolor='white',

plot\_bgcolor='white'

)

st.plotly\_chart(fig2, use\_container\_width=True)

# ... 保持原有的柱状图代码不变 ...

主要修改：

添加了国家代码映射字典，将国家名称映射到 ISO-3 代码

使用 Choropleth 图层来展示国家用户数量的热力分布

使用渐变色系来表示用户数量的多少，从浅色（用户少）到深色（用户多）

添加了颜色条来显示数值范围

保留了红色标记点来显示具体用户位置

这样的效果会让地图更加直观地展示各国的 GitHub 用户分布密度，颜色越深的国家表示用户数量越多。

A：用户加入Github的年份分布，我希望做成一条水平带箭头的时间轴，轴两侧有上下对称的柱形，长度代表对应年份加入Github的用户数的大小。

C：我来帮你创建一个水平双向柱状图来展示用户加入 Github 的年份分布：

with col6:

# Chart6: 加入年份分布（水平双向柱状图）

join\_year\_counts = df['join\_year'].value\_counts().sort\_index()

fig6 = go.Figure()

# 添加上半部分柱形

fig6.add\_trace(go.Bar(

x=join\_year\_counts.index,

y=join\_year\_counts.values,

name='加入用户数',

marker\_color='#3480b8',

opacity=0.7

))

# 添加下半部分柱形（使用相同的数据但取负值）

fig6.add\_trace(go.Bar(

x=join\_year\_counts.index,

y=-join\_year\_counts.values, # 取负值使柱形向下显示

name='加入用户数',

marker\_color='#3480b8',

opacity=0.7,

showlegend=False # 不显示重复的图例

))

# 更新布局

fig6.update\_layout(

title='用户加入 Github 年份分布',

showlegend=False,

height=400,

margin=dict(t=30, b=30, l=30, r=30),

paper\_bgcolor='white',

plot\_bgcolor='white',

yaxis=dict(

zeroline=True,

zerolinewidth=2,

zerolinecolor='black', # 中间线颜色

showgrid=False,

showticklabels=False # 隐藏y轴刻度

),

xaxis=dict(

showgrid=False,

zeroline=False,

tickmode='linear',

dtick=1 # 设置x轴刻度间隔为1年

)

)

# 添加箭头

fig6.add\_annotation(

x=max(join\_year\_counts.index) + 0.5, # 右箭头

y=0,

ax=max(join\_year\_counts.index) + 0.1,

ay=0,

xref='x',

yref='y',

axref='x',

ayref='y',

text='',

showarrow=True,

arrowhead=2,

arrowsize=1.5,

arrowwidth=2,

arrowcolor='black'

)

fig6.add\_annotation(

x=min(join\_year\_counts.index) - 0.5, # 左箭头

y=0,

ax=min(join\_year\_counts.index) - 0.1,

ay=0,

xref='x',

yref='y',

axref='x',

ayref='y',

text='',

showarrow=True,

arrowhead=2,

arrowsize=1.5,

arrowwidth=2,

arrowcolor='black'

)

st.plotly\_chart(fig6, use\_container\_width=True)

这段代码：

创建了上下对称的柱状图

添加了水平时间轴（黑色中线）

在轴两端添加了箭头

隐藏了y轴刻度，保持视觉简洁

使用了半透明的蓝色来表示柱形

每年显示一个刻度值

这样就形成了一个时间轴式的年份分布图，直观地展示了不同年份加入 Github 的用户数量分布。

A：现在箭头上下的柱形不是对称的，而是错开的，请修改

C：我来修改代码，确保上下柱形完全对称：

with col6:

# Chart6: 加入年份分布（水平双向柱状图）

join\_year\_counts = df['join\_year'].value\_counts().sort\_index()

fig6 = go.Figure()

# 添加上半部分柱形

fig6.add\_trace(go.Bar(

x=join\_year\_counts.index,

y=join\_year\_counts.values,

name='加入用户数',

marker\_color='#3480b8',

opacity=0.7,

width=1,

offset=0 # 确保不会偏移

))

# 添加下半部分柱形

fig6.add\_trace(go.Bar(

x=join\_year\_counts.index,

y=-join\_year\_counts.values,

name='加入用户数',

marker\_color='#3480b8',

opacity=0.7,

width=1,

offset=0, # 确保不会偏移

showlegend=False

))

# 更新布局

fig6.update\_layout(

title='用户加入 Github 年份分布',

showlegend=False,

height=400,

margin=dict(t=30, b=30, l=30, r=30),

paper\_bgcolor='white',

plot\_bgcolor='white',

yaxis=dict(

zeroline=True,

zerolinewidth=2,

zerolinecolor='black',

showgrid=False,

showticklabels=False

),

xaxis=dict(

showgrid=False,

zeroline=False,

tickmode='linear',

dtick=1

),

bargap=0,

bargroupgap=0,

barmode='overlay' # 使用叠加模式而不是分组模式

)

主要修改：

添加了 offset=0 参数确保柱形不会偏移

设置 barmode='overlay' 使用叠加模式

保持其他设置不变

这样应该能确保上下柱形完全对称对齐。

A：我不想用柱形图表示了，而是用折线来表示用户注册的趋势，并为其与时间轴围成区域着色

C：我来修改代码，使用平滑的折线图来展示趋势：

with col6:

# Chart6: 加入年份分布（平滑曲线）

join\_year\_counts = df['join\_year'].value\_counts().sort\_index()

fig6 = go.Figure()

# 添加上半部分曲线

fig6.add\_trace(go.Scatter(

x=join\_year\_counts.index,

y=join\_year\_counts.values,

mode='lines',

line=dict(

shape='spline',

smoothing=1.3,

width=3,

color='rgb(117,107,177)'

),

fill='tonexty',

fillcolor='rgba(117,107,177,0.2)',

name='加入用户数',

hovertemplate='%{y}人<extra></extra>'

))

# 添加下半部分曲线（对称）

fig6.add\_trace(go.Scatter(

x=join\_year\_counts.index,

y=-join\_year\_counts.values,

mode='lines',

line=dict(

shape='spline',

smoothing=1.3,

width=3,

color='rgb(117,107,177)'

),

fill='tonexty',

fillcolor='rgba(117,107,177,0.2)',

name='加入用户数',

showlegend=False,

hovertemplate='%{y}人<extra></extra>'

))

# 添加箭头

fig6.add\_annotation(

x=max(join\_year\_counts.index) + 1.5,

y=0,

ax=max(join\_year\_counts.index) + 0.8,

ay=0,

xref='x',

yref='y',

axref='x',

ayref='y',

text='',

showarrow=True,

arrowhead=3,

arrowsize=1.5,

arrowwidth=2,

arrowcolor='rgba(0,0,0,0.8)'

)

# 更新布局

fig6.update\_layout(

title='用户加入 Github 年份分布',

showlegend=False,

height=400,

margin=dict(t=30, b=30, l=30, r=30),

paper\_bgcolor='white',

plot\_bgcolor='white',

yaxis=dict(

zeroline=True,

zerolinewidth=2,

zerolinecolor='rgb(117,107,177)',

showgrid=False,

showticklabels=False

),

xaxis=dict(

showgrid=False,

zeroline=False,

tickmode='linear',

dtick=1

)

)

st.plotly\_chart(fig6, use\_container\_width=True)

主要修改：

使用 go.Scatter 创建平滑的折线图

设置 shape='spline' 和 smoothing=1.3 使曲线更平滑

使用 fill='tonexty' 填充曲线与轴之间的区域

保持上下对称的设计

使用半透明的填充色

保留了时间轴和箭头的设计

这样会形成一个更流畅的趋势展示效果。

A：我希望折线能够连接相邻年份，而不是平滑略过中间用户数为0的年份

C：我来修改代码，确保折线连接所有相邻年份：

with col6:

# Chart6: 加入年份分布（折线图）

join\_year\_counts = df['join\_year'].value\_counts().sort\_index()

# 补充缺失年份的数据（用户数为0）

all\_years = range(join\_year\_counts.index.min(), join\_year\_counts.index.max() + 1)

join\_year\_counts = join\_year\_counts.reindex(all\_years, fill\_value=0)

fig6 = go.Figure()

# 添加上半部分曲线

fig6.add\_trace(go.Scatter(

x=join\_year\_counts.index,

y=join\_year\_counts.values,

mode='lines',

line=dict(

shape='spline',

smoothing=0, # 设为0以保持直线连接

width=3,

color='rgb(117,107,177)'

),

fill='tonexty',

fillcolor='rgba(117,107,177,0.2)',

name='加入用户数',

hovertemplate='%{y}人<extra></extra>'

))

# 添加下半部分曲线（对称）

fig6.add\_trace(go.Scatter(

x=join\_year\_counts.index,

y=-join\_year\_counts.values,

mode='lines',

line=dict(

shape='spline',

smoothing=0, # 设为0以保持直线连接

width=3,

color='rgb(117,107,177)'

),

fill='tonexty',

fillcolor='rgba(117,107,177,0.2)',

name='加入用户数',

showlegend=False,

hovertemplate='%{y}人<extra></extra>'

))

# ... 其余代码保持不变 ...

主要修改：

添加了缺失年份的数据补充，确保所有年份都有数据点

将 smoothing 参数设为 0，使折线直接连接相邻点

其他设置保持不变

这样折线会连接所有相邻年份，包括用户数为0的年份。

A：如何把chart1的图例移到下方？

C：可以通过修改 legend 参数来将 Chart1 的图例移到下方

fig1.update\_layout(

title='Top 200 用户的关注者数量和2024年贡献数对比',

height=870,

yaxis=dict(

tickfont=dict(size=12),

range=[180, 200],

autorange=False,

showticklabels=True,

type="category",

constrain="domain",

fixedrange=False,

tickmode='linear',

dtick=1

),

xaxis=dict(

range=[0, 250000],

showgrid=True,

fixedrange=True,

rangemode='nonnegative',

side='bottom',

domain=[0, 1]

),

xaxis2=dict(

range=[0, 5000],

showgrid=True,

fixedrange=True,

rangemode='nonnegative',

side='bottom',

overlaying='x',

position=1,

domain=[0, 1]

),

showlegend=True,

legend=dict(

orientation='h', # 水平排列

yanchor='top', # 顶部对齐

y=-0.1, # 位置在图表下方

xanchor='center', # 水平居中

x=0.5 # 水平位置

),

margin=dict(l=20, r=20, t=40, b=50), # 增加底部边距以容纳图例

dragmode='pan',

bargap=0.15,

bargroupgap=0.1,

barmode='group'

)

主要修改：

修改了 legend 参数：

orientation='h' 设置为水平排列

y=-0.1 将图例移到图表下方

x=0.5 和 xanchor='center' 使图例水平居中

增加了底部边距 margin=dict(b=50) 以给图例留出空间

这样图例就会显示在图表的下方了。

（注：下面这段对话来自ChatGPT o-1，并非来自Cursor-claude 3.5）

A：请你再做一些炫酷的可视化分析。

C：下面将提供一些更加炫酷和多样化的可视化分析示例，这些示例侧重于更高级或具有交互特性的可视化形式。假设之前的数据处理步骤（如数据加载、清洗、数值转换）已完成并存储在df中。

以下示例包括：

\*\*雷达图（Radar Chart）\*\*对比多名顶级用户在多个指标上的表现

\*\*旭日图（Sunburst Chart）\*\*展示用户国家分布与语言分布层次结构

\*\*桑基图（Sankey Diagram）\*\*可视化组织与语言之间的关系流

\*\*词云（Word Cloud）\*\*用于可视化组织或位置文本的出现频率（需额外安装wordcloud库）

\*\*地理可视化（Choropleth Map）\*\*展示用户所在国家的分布（需要已提取的国家字段）

请根据实际情况对数据做适配与清洗。

代码示例

import pandas as pd

import plotly.express as px

import plotly.graph\_objects as go

from collections import Counter

import math

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import seaborn as sns

import os

# 假设df已经存在且包含以下列：

# 'name', 'followers\_num', 'stars\_num', 'repo\_num\_num', 'contributions\_2024\_num', 'country', 'languages', 'org'

# ---------------------------

# 1. 雷达图（Radar Chart）

# 对比Followers最高的前5名用户的多个指标: followers\_num, repo\_num\_num, contributions\_2024\_num, stars\_num

# ---------------------------

# 选出前5个最高followers用户

top5\_followers = df.nlargest(5, 'followers\_num')

# 需要为雷达图构建数据格式：每个维度为一个指标，行代表一个用户

metrics = ['followers\_num', 'repo\_num\_num', 'contributions\_2024\_num', 'stars\_num']

categories = ['Followers', 'Repos', 'Contributions\_2024', 'Stars']

fig\_radar = go.Figure()

for i, row in top5\_followers.iterrows():

values = [row[m] if not pd.isna(row[m]) else 0 for m in metrics]

fig\_radar.add\_trace(go.Scatterpolar(

r=values,

theta=categories,

fill='toself',

name=row['name']

))

fig\_radar.update\_layout(

polar=dict(

radialaxis=dict(visible=True)

),

showlegend=True,

title="Top 5 Users by Followers - Radar Chart Comparison"

)

fig\_radar.show()

# ---------------------------

# 2. 旭日图（Sunburst Chart）

# 分析国家 -> 语言 -> 用户数量的分层结构

# 首先统计 (Country, Language) 对出现的频率

# ---------------------------

# 为每个用户对其languages进行计数，并关联到国家，如果没有国家就用'Unknown'

country\_lang\_counts = []

for idx, row in df.dropna(subset=['languages']).iterrows():

country = row['country'] if pd.notna(row['country']) else 'Unknown'

for lang in row['languages']:

country\_lang\_counts.append((country, lang))

cl\_df = pd.DataFrame(country\_lang\_counts, columns=['country', 'language'])

cl\_count = cl\_df.groupby(['country','language']).size().reset\_index(name='count')

fig\_sunburst = px.sunburst(

cl\_count,

path=['country', 'language'],

values='count',

title="Sunburst of Users by Country and Language"

)

fig\_sunburst.show()

# ---------------------------

# 3. 桑基图（Sankey Diagram）

# 展示组织与其主要编程语言之间的关系流

# 思路：节点为组织和语言，link为计数

# 需要对org进行清洗，空org的跳过，选取top N org，然后统计这些org中使用的语言数量

# ---------------------------

org\_lang\_counts = []

org\_df = df.dropna(subset=['org', 'languages'])

for idx, row in org\_df.iterrows():

for lang in row['languages']:

org\_lang\_counts.append((row['org'], lang))

ol\_df = pd.DataFrame(org\_lang\_counts, columns=['org', 'language'])

top\_orgs = ol\_df['org'].value\_counts().head(10).index # top 10 org

ol\_filtered = ol\_df[ol\_df['org'].isin(top\_orgs)]

ol\_count = ol\_filtered.groupby(['org','language']).size().reset\_index(name='count')

# 构建Sankey数据

all\_orgs = ol\_count['org'].unique().tolist()

all\_langs = ol\_count['language'].unique().tolist()

nodes = all\_orgs + all\_langs

# 节点索引映射

node\_dict = {node: i for i, node in enumerate(nodes)}

links = {

'source': [],

'target': [],

'value': []

}

for \_, r in ol\_count.iterrows():

links['source'].append(node\_dict[r['org']])

links['target'].append(node\_dict[r['language']])

links['value'].append(r['count'])

fig\_sankey = go.Figure(data=[go.Sankey(

node = dict(

pad = 15,

thickness = 20,

line = dict(color = "black", width = 0.5),

label = nodes

),

link = dict(

source = links['source'],

target = links['target'],

value = links['value']

))])

fig\_sankey.update\_layout(title\_text="Sankey Diagram: Top Orgs to Languages", font\_size=10)

fig\_sankey.show()

# ---------------------------

# 4. 词云（Word Cloud） - 分析组织或位置数据

# 将org字段生成词云，用于可视化哪一些组织名称出现频率高

# ---------------------------

org\_texts = df['org'].dropna().tolist()

org\_str = " ".join(org\_texts)

wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(org\_str)

plt.figure(figsize=(10,5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.title("Word Cloud of Organizations")

plt.show()

# ---------------------------

# 5. 地理可视化（Choropleth Map）

# 映射国家到用户数量, 需要已有的'country'字段为标准英文国家名或ISO码。

# 对于国家名与ISO码映射较复杂，这里假设country字段中为英文国家名，可使用plotly.express自带的iso映射

# 若实际数据不匹配，需手动建立映射表。这里仅作示例。

# ---------------------------

country\_counts = df['country'].value\_counts().reset\_index()

country\_counts.columns = ['country', 'count']

# 为了choropleth，需要ISO-3编码，可使用pycountry进行映射，若无此库则手工映射或对数据简化。

# 如果您安装了pycountry:

try:

import pycountry

country\_iso\_map = {}

for country\_name in country\_counts['country']:

try:

country\_data = pycountry.countries.search\_fuzzy(country\_name)[0]

country\_iso\_map[country\_name] = country\_data.alpha\_3

except:

# 如果找不到匹配的，就标记为None

country\_iso\_map[country\_name] = None

country\_counts['iso\_alpha'] = country\_counts['country'].map(country\_iso\_map)

# 去掉无效编码的

country\_counts = country\_counts.dropna(subset=['iso\_alpha'])

fig\_choropleth = px.choropleth(

country\_counts,

locations="iso\_alpha",

color="count",

hover\_name="country",

color\_continuous\_scale=px.colors.sequential.Plasma,

title="User Distribution by Country"

)

fig\_choropleth.show()

except ImportError:

print("请安装pycountry库以启用地理映射。例如: pip install pycountry")

# ---------------------------

# 其他炫酷可视化思路：

# - 使用Plotly的3D散点图或3D气泡图展示多个维度数据。

# - 使用NetworkX与Plotly绘图将用户-语言-组织构建成图网络可视化。

# - 利用动画（animation\_frame）展示随年份变化的用户数量动态变化。

# ---------------------------

效果和意义说明

雷达图（Radar Chart）：展示多个主要指标（如followers、repos、contributions）在不同用户间的对比，使得我们能一目了然地比较这些顶级用户的多维实力。

旭日图（Sunburst Chart）：通过层次结构（国家 -> 语言）展示用户分布和语言偏好，让人直观地看出某些国家用户更偏好哪类语言。

桑基图（Sankey Diagram）：将组织与语言的关系可视化为流动图形，清晰表达出各组织中主要编程语言的分布和关联强度。

词云（Word Cloud）：通过组织字段的词云展示哪些组织在数据中出现更频繁，从视觉层面突出高频出现的组织。

地理可视化（Choropleth Map）：以世界地图为背景，用颜色深浅表示不同国家的用户数量，能够快速识别用户集中度高的国家或地区。

这些可视化方法不仅使分析结果更加生动、直观，也能让人更容易发现数据中隐藏的模式与趋势。

（与ChatGPT o-1的对话结束，下面的对话恢复和Cursor-claude 3.5的对话）

A：我想添加一个桑基图，用于展示组织与其主要编程语言之间的关系流。具体要求见下述代码的注释部分。请参照以下代码为我实现

# ---------------------------

# 3. 桑基图（Sankey Diagram）

# 展示组织与其主要编程语言之间的关系流

# 思路：节点为组织和语言，link为计数

# 需要对org进行清洗，空org的跳过，org与loc字段相同的跳过，选取top 10 org，然后统计这些org中使用的语言数量

# ---------------------------

org\_lang\_counts = []

org\_df = df.dropna(subset=['org', 'languages'])

for idx, row in org\_df.iterrows():

for lang in row['languages']:

org\_lang\_counts.append((row['org'], lang))

ol\_df = pd.DataFrame(org\_lang\_counts, columns=['org', 'language'])

top\_orgs = ol\_df['org'].value\_counts().head(10).index # top 10 org

ol\_filtered = ol\_df[ol\_df['org'].isin(top\_orgs)]

ol\_count = ol\_filtered.groupby(['org','language']).size().reset\_index(name='count')

# 构建Sankey数据

all\_orgs = ol\_count['org'].unique().tolist()

all\_langs = ol\_count['language'].unique().tolist()

nodes = all\_orgs + all\_langs

# 节点索引映射

node\_dict = {node: i for i, node in enumerate(nodes)}

links = {

'source': [],

'target': [],

'value': []

}

for \_, r in ol\_count.iterrows():

links['source'].append(node\_dict[r['org']])

links['target'].append(node\_dict[r['language']])

links['value'].append(r['count'])

fig\_sankey = go.Figure(data=[go.Sankey(

node = dict(

pad = 15,

thickness = 20,

line = dict(color = "black", width = 0.5),

label = nodes

),

link = dict(

source = links['source'],

target = links['target'],

value = links['value']

))])

fig\_sankey.update\_layout(title\_text="Sankey Diagram: Top Orgs to Languages", font\_size=10)

fig\_sankey.show()

C：我来帮你实现这个桑基图，并添加一些优化：

with col7:

# Chart7: 桑基图

# 数据准备

org\_lang\_counts = []

org\_df = df.dropna(subset=['org', 'languages'])

# 清洗数据：跳过空org和org与loc相同的情况

org\_df = org\_df[org\_df['org'] != org\_df['loc']]

for idx, row in org\_df.iterrows():

for lang in row['languages']:

org\_lang\_counts.append((row['org'], lang))

ol\_df = pd.DataFrame(org\_lang\_counts, columns=['org', 'language'])

top\_orgs = ol\_df['org'].value\_counts().head(10).index

ol\_filtered = ol\_df[ol\_df['org'].isin(top\_orgs)]

ol\_count = ol\_filtered.groupby(['org','language']).size().reset\_index(name='count')

# 构建Sankey数据

all\_orgs = ol\_count['org'].unique().tolist()

all\_langs = ol\_count['language'].unique().tolist()

nodes = all\_orgs + all\_langs

# 节点索引映射

node\_dict = {node: i for i, node in enumerate(nodes)}

# 准备links数据

links = {

'source': [],

'target': [],

'value': []

}

for \_, r in ol\_count.iterrows():

links['source'].append(node\_dict[r['org']])

links['target'].append(node\_dict[r['language']])

links['value'].append(r['count'])

# 创建桑基图

fig7 = go.Figure(data=[go.Sankey(

node = dict(

pad = 15,

thickness = 20,

line = dict(

color = "white", # 节点边框颜色

width = 0.5

),

label = nodes,

color = ['rgb(117,107,177)'] \* len(all\_orgs) + # 组织节点颜色

['rgb(77,61,143)'] \* len(all\_langs), # 语言节点颜色

customdata = ['组织'] \* len(all\_orgs) + ['语言'] \* len(all\_langs), # 用于hover显示

hovertemplate='%{label}<br>类型: %{customdata}<extra></extra>'

),

link = dict(

source = links['source'],

target = links['target'],

value = links['value'],

color = ['rgba(117,107,177,0.3)'] \* len(links['value']), # 连接的颜色

hovertemplate='%{source.label} → %{target.label}<br>数量: %{value}<extra></extra>'

)

)])

# 更新布局

fig7.update\_layout(

title=dict(

text='Top 10 组织与编程语言关系图',

x=0.5,

y=0.95,

xanchor='center',

yanchor='top',

font=dict(color='white', size=16)

),

font=dict(color='white', size=12),

paper\_bgcolor='rgb(17,17,17)',

plot\_bgcolor='rgb(17,17,17)',

height=600,

margin=dict(t=60, b=30, l=30, r=30)

)

st.plotly\_chart(fig7, use\_container\_width=True)

主要特点：

数据清洗：

跳过空org

跳过org与loc相同的情况

选取top 10的组织

可视化优化：

使用紫色主题，与其他图表保持一致

区分组织节点和语言节点的颜色

半透明的连接线

自定义的悬停提示

适配深色主题

交互性：

鼠标悬停时显示详细信息

可以拖动节点调整位置

显示流向和数量

这个桑基图可以直观地展示组织和编程语言之间的关系，以及各种语言在不同组织中的使用情况。

A：参考以下代码为我添加一个旭日图：

# ---------------------------

# 2. 旭日图（Sunburst Chart）

# 分析国家 -> 语言 -> 用户数量的分层结构

# 首先统计 (Country, Language) 对出现的频率

# ---------------------------

# 为每个用户对其languages进行计数，并关联到国家，如果没有国家就用'Unknown'

country\_lang\_counts = []

for idx, row in df.dropna(subset=['languages']).iterrows():

country = row['country'] if pd.notna(row['country']) else 'Unknown'

for lang in row['languages']:

country\_lang\_counts.append((country, lang))

cl\_df = pd.DataFrame(country\_lang\_counts, columns=['country', 'language'])

cl\_count = cl\_df.groupby(['country','language']).size().reset\_index(name='count')

fig\_sunburst = px.sunburst(

cl\_count,

path=['country', 'language'],

values='count',

title="Sunburst of Users by Country and Language"

)

fig\_sunburst.show()

C：我来帮你实现这个旭日图，并添加一些优化：

with col8:

# Chart8: 旭日图

# 数据准备

country\_lang\_counts = []

for idx, row in df.dropna(subset=['languages']).iterrows():

country = row['country'] if pd.notna(row['country']) else 'Unknown'

for lang in row['languages']:

country\_lang\_counts.append((country, lang))

cl\_df = pd.DataFrame(country\_lang\_counts, columns=['country', 'language'])

cl\_count = cl\_df.groupby(['country','language']).size().reset\_index(name='count')

# 过滤掉 'Other' 和 'Unknown' 国家

cl\_count = cl\_count[~cl\_count['country'].isin(['Other', 'Unknown'])]

# 创建旭日图

fig8 = px.sunburst(

cl\_count,

path=['country', 'language'],

values='count',

color='count',

color\_continuous\_scale=[

[0, 'rgb(117,107,177)'], # 最浅的紫色

[0.5, 'rgb(97,84,160)'], # 中等紫色

[1, 'rgb(77,61,143)'] # 最深的紫色

],

maxdepth=2,

branchvalues='total'

)

# 更新布局

fig8.update\_layout(

title=dict(

text='国家与编程语言分布',

x=0.5,

y=0.95,

xanchor='center',

yanchor='top',

font=dict(color='white', size=16)

),

font=dict(color='white', size=12),

paper\_bgcolor='rgb(17,17,17)',

plot\_bgcolor='rgb(17,17,17)',

height=600,

margin=dict(t=60, b=30, l=30, r=30)

)

# 更新颜色轴

fig8.update\_coloraxes(

showscale=True,

colorbar=dict(

title=dict(

text="用户数量",

font=dict(color='white')

),

tickfont=dict(color='white'),

len=0.5,

thickness=15

)

)

# 更新悬停信息

fig8.update\_traces(

hovertemplate="<b>%{label}</b><br>" +

"用户数量: %{value}<br>" +

"占比: %{percentParent:.1%} of %{parent}<extra></extra>"

)

st.plotly\_chart(fig8, use\_container\_width=True)

主要特点：

数据处理：

过滤掉了 'Other' 和 'Unknown' 国家

保持了国家和语言的层级结构

可视化优化：

使用紫色渐变主题

颜色深浅表示用户数量

适配深色主题

添加了颜色图例

交互性：

自定义的悬停提示，显示：

名称

用户数量

相对于父级的百分比

可以点击扇形进行下钻

可以点击中心返回上层

布局优化：

居中的标题

合适的图表大小和边距

清晰的文字标签

这个旭日图可以直观地展示不同国家的编程语言分布情况，以及它们之间的层级关系。

A：能否使用Plotly的3D散点图或3D气泡图展示多个维度数据？

C：我来帮你添加一个3D散点图，展示用户的多个维度数据：

with col10:

# Chart10: 3D散点图

# 准备数据

plot\_df = df.copy()

plot\_df['followers\_num'] = df['followers'].apply(convert\_to\_number)

plot\_df['stars\_num'] = df['stars'].apply(convert\_to\_number)

# 创建3D散点图

fig10 = go.Figure()

# 添加散点

fig10.add\_trace(go.Scatter3d(

x=plot\_df['followers\_num'],

y=plot\_df['repo\_num'],

z=plot\_df['contributions\_in\_2024'],

mode='markers',

marker=dict(

size=plot\_df['stars\_num'].apply(lambda x: np.log(x + 1) \* 2), # 使用对数缩放

color=plot\_df['stars\_num'], # 颜色基于stars数量

colorscale=[

[0, 'rgb(255,140,0)'], # 橙色

[0.2, 'rgb(255,0,128)'], # 粉红

[0.4, 'rgb(128,0,255)'], # 紫色

[0.6, 'rgb(0,128,255)'], # 蓝色

[0.8, 'rgb(0,255,192)'], # 青色

[1, 'rgb(0,255,64)'] # 绿色

],

opacity=0.8,

colorbar=dict(

title=dict(

text="获赞数",

font=dict(color='white')

),

tickfont=dict(color='white')

)

),

text=plot\_df['name'], # 用于悬停显示

hovertemplate=

"<b>%{text}</b><br>" +

"关注者: %{x:,.0f}<br>" +

"仓库数: %{y:,.0f}<br>" +

"2024贡献: %{z:,.0f}<br>" +

"<extra></extra>"

))

# 更新布局

fig10.update\_layout(

title=dict(

text='用户多维度3D分布',

x=0.5,

y=0.95,

xanchor='center',

yanchor='top',

font=dict(color='white', size=16)

),

scene=dict(

xaxis=dict(

title='关注者数量',

titlefont=dict(color='white'),

tickfont=dict(color='white'),

gridcolor='rgba(255,255,255,0.1)',

backgroundcolor='rgb(17,17,17)',

showbackground=True

),

yaxis=dict(

title='仓库数量',

titlefont=dict(color='white'),

tickfont=dict(color='white'),

gridcolor='rgba(255,255,255,0.1)',

backgroundcolor='rgb(17,17,17)',

showbackground=True

),

zaxis=dict(

title='2024年贡献',

titlefont=dict(color='white'),

tickfont=dict(color='white'),

gridcolor='rgba(255,255,255,0.1)',

backgroundcolor='rgb(17,17,17)',

showbackground=True

),

bgcolor='rgb(17,17,17)'

),

paper\_bgcolor='rgb(17,17,17)',

plot\_bgcolor='rgb(17,17,17)',

height=700,

margin=dict(t=60, b=30, l=30, r=30)

)

# 添加相机视角

fig10.update\_layout(

scene\_camera=dict(

up=dict(x=0, y=0, z=1),

center=dict(x=0, y=0, z=0),

eye=dict(x=1.5, y=1.5, z=1.5)

)

)

st.plotly\_chart(fig10, use\_container\_width=True)

这个3D散点图的特点：

数据维度：

X轴：关注者数量

Y轴：仓库数量

Z轴：2024年贡献数

气泡大小：获赞数（使用对数缩放）

气泡颜色：也基于获赞数（使用渐变色）

视觉效果：

炫丽的渐变色系

半透明的气泡效果

深色主题背景

3D网格线

交互特性：

可自由旋转和缩放

详细的悬停提示

颜色图例

预设的最佳观察角度

优化：

对数缩放气泡大小，避免极值影响

格式化数字显示

清晰的坐标轴标签

这个图表可以让用户直观地看到不同维度数据之间的关系，以及数据的分布情况。

（以下对话阐释了本设计中采用的影响力计算函数的出处及意义）

A：请结合followers，contributions\_in\_2024，repo\_num，join\_date这四个数据给出一个合理的计算用户综合影响力的函数，并绘制top200用户综合影响力的趋势图：以followers数排名从左到右降序排列用户，横坐标为followers数排名，纵坐标为综合影响力分数，并将相邻横坐标的点连起来绘制趋势线。

C：我来实现一个用户综合影响力评分系统和趋势图

with col10:

# Chart10: 用户综合影响力趋势图

def calculate\_influence\_score(row):

"""计算用户综合影响力分数

计算公式：

1. followers\_score: 关注者数量的对数值 (权重: 0.4)

2. contribution\_score: 2024年贡献数的对数值 (权重: 0.3)

3. repo\_score: 仓库数量的对数值 (权重: 0.2)

4. experience\_score: 注册年限 (权重: 0.1)

所有分数都归一化到0-100的范围

"""

# 获取当前年份

current\_year = 2024

# 计算各项原始分数

followers\_num = convert\_to\_number(row['followers'])

followers\_score = np.log10(followers\_num + 1)

contribution\_score = np.log10(float(row['contributions\_in\_2024']) + 1)

repo\_score = np.log10(float(convert\_to\_number(row['repo\_num'])) + 1)

join\_year = pd.to\_datetime(row['join\_date']).year

experience\_years = current\_year - join\_year

experience\_score = experience\_years

return {

'followers\_score': followers\_score,

'contribution\_score': contribution\_score,

'repo\_score': repo\_score,

'experience\_score': experience\_score

}

# 计算所有用户的原始分数

scores = []

for \_, row in df.iterrows():

if row['contributions\_in\_2024'] == None:

continue

score\_dict = calculate\_influence\_score(row)

scores.append(score\_dict)

# 转换为DataFrame

scores\_df = pd.DataFrame(scores)

# 对每个维度进行归一化（0-100）

for column in scores\_df.columns:

min\_val = scores\_df[column].min()

max\_val = scores\_df[column].max()

scores\_df[column] = 100 \* (scores\_df[column] - min\_val) / (max\_val - min\_val)

# 计算加权总分

weights = {

'followers\_score': 0.4,

'contribution\_score': 0.3,

'repo\_score': 0.2,

'experience\_score': 0.1

}

df['influence\_score'] = sum(scores\_df[col] \* weight

for col, weight in weights.items())

# 排序并添加排名

df['rank'] = df['influence\_score'].rank(ascending=False)

# 创建趋势图

fig11 = go.Figure()

# 添加主要趋势线

fig11.add\_trace(go.Scatter(

x=df['rank'],

y=df['influence\_score'],

mode='lines',

name='影响力趋势',

line=dict(

color='rgba(0,128,255,0.8)',

width=3

)

))

# 添加散点

fig11.add\_trace(go.Scatter(

x=df['rank'],

y=df['influence\_score'],

mode='markers',

name='用户',

marker=dict(

size=10,

color=df['influence\_score'],

colorscale=[

[0, 'rgb(255,140,0)'], # 橙色

[0.2, 'rgb(255,0,128)'], # 粉红

[0.4, 'rgb(128,0,255)'], # 紫色

[0.6, 'rgb(0,128,255)'], # 蓝色

[0.8, 'rgb(0,255,192)'], # 青色

[1, 'rgb(0,255,64)'] # 绿色

],

showscale=True,

colorbar=dict(

title=dict(

text="影响力分数",

font=dict(color='white')

),

tickfont=dict(color='white')

)

),

text=df['name'],

hovertemplate=

"<b>%{text}</b><br>" +

"排名: %{x:.0f}<br>" +

"影响力分数: %{y:.1f}<br>" +

"<extra></extra>"

))

# 更新布局

fig11.update\_layout(

title=dict(

text='GitHub Top 200 用户影响力分布',

x=0.5,

y=0.95,

xanchor='center',

yanchor='top',

font=dict(color='white', size=16)

),

xaxis=dict(

title='排名',

titlefont=dict(color='white'),

tickfont=dict(color='white'),

gridcolor='rgba(255,255,255,0.1)',

tickmode='linear',

dtick=20

),

yaxis=dict(

title='影响力分数',

titlefont=dict(color='white'),

tickfont=dict(color='white'),

gridcolor='rgba(255,255,255,0.1)'

),

paper\_bgcolor='rgb(17,17,17)',

plot\_bgcolor='rgb(17,17,17)',

height=600,

margin=dict(t=60, b=30, l=30, r=30),

showlegend=True,

legend=dict(

font=dict(color='white'),

bgcolor='rgba(0,0,0,0)',

bordercolor='rgba(255,255,255,0.2)',

borderwidth=1

)

)

st.plotly\_chart(fig11, use\_container\_width=True)

# 显示 Top 10 影响力用户

st.markdown("### Top 10 最具影响力的用户")

top\_10\_users = df.nsmallest(10, 'rank')[['name', 'influence\_score', 'rank']]

st.table(top\_10\_users)

特点：

影响力计算：

followers (40%): 反映用户的影响范围

2024年贡献 (30%): 反映用户的活跃度

仓库数量 (20%): 反映用户的产出

注册年限 (10%): 反映用户的经验

数据处理：

使用对数处理大数值

归一化所有指标到0-100

加权计算最终分数

可视化：

趋势线展示整体分布

散点标记具体用户

颜色渐变表示分数高低

可交互的悬停提示

额外信息：

显示Top 10影响力用户列表

清晰的坐标轴刻度

完整的图例说明

这个综合评分系统考虑了用户的多个重要维度，可以较为客观地反映用户的综合影响力。

A：我希望世界地图（fig）能够单独占满一行，而不用分列显示

C：让我修改布局，将世界地图单独放在一行：

# 标题

st.title("Github Top 200 用户数据分析")

# 世界地图单独占一行

fig = go.Figure()

# ... (世界地图的代码保持不变)

st.plotly\_chart(fig, use\_container\_width=True)

# 其他图表使用两列布局

col1, col2 = st.columns(2)

with col1:

# Chart1: 水平双向柱状图

# ... (其他代码保持不变)

with col2:

# 其他图表

# ... (其他代码保持不变)

主要修改：

移除了最初的 col, col0 = st.columns(2)

将世界地图直接放在列布局之前

保持其他图表的两列布局不变

这样世界地图就会占据整个页面宽度显示，视觉效果会更好。

A：我希望将我的代码进行模块化处理，每一个chart分成一个.py文件

C：好的，我来帮你将代码模块化。首先创建一个新的目录结构：

project/

├── main.py # 主文件

├── utils/

│ └── data\_loader.py # 数据加载和预处理函数

└── charts/

├── \_\_init\_\_.py

├── world\_map.py # 世界地图

├── user\_bar.py # 用户柱状图

├── radar\_chart.py # 雷达图

├── growth\_animation.py # 增长动画

├── influence\_trend.py # 影响力趋势图

└── top10\_line.py # Top10折线图

为其他每个图表创建单独的文件，每个文件都包含一个 plot 函数，接收必要的参数并绘制相应的图表。

这种模块化的好处是：

代码更容易维护和理解

每个图表可以独立开发和测试

可以轻松添加新的图表或修改现有图表

主文件更简洁清晰

你想要我继续提供其他图表模块的具体代码吗？

A：对于chart8，请将node和link的颜色全部变得更鲜艳

C：好的，让我们修改 chart\_08.py 中的颜色设置，使用更鲜艳的颜色：

def chart\_08(df, pcolor, title\_font, tick\_font):

# ... 其他代码保持不变 ...

# 为节点设置更鲜艳的颜色

org\_colors = [

'rgb(255,0,0)', # 鲜红

'rgb(255,128,0)', # 亮橙

'rgb(255,255,0)', # 鲜黄

'rgb(0,255,0)', # 鲜绿

'rgb(0,255,255)', # 青色

'rgb(0,128,255)', # 亮蓝

'rgb(0,0,255)', # 鲜蓝

'rgb(128,0,255)', # 紫色

'rgb(255,0,255)', # 品红

'rgb(255,0,128)' # 玫红

]

lang\_colors = [

'rgb(255,50,50)', # 亮红

'rgb(255,150,50)', # 橙红

'rgb(255,255,50)', # 亮黄

'rgb(50,255,50)', # 亮绿

'rgb(50,255,255)', # 亮青

'rgb(50,150,255)', # 天蓝

'rgb(150,50,255)', # 亮紫

'rgb(255,50,255)', # 亮粉

'rgb(255,128,128)', # 浅红

'rgb(128,255,128)' # 浅绿

]

# 为连接设置更鲜艳的颜色

link\_colors = []

for \_, r in ol\_count.iterrows():

source\_idx = node\_dict[r['org']]

target\_idx = node\_dict[r['language']] - len(all\_orgs)

# 混合源节点和目标节点的颜色，使其更鲜艳

source\_color = np.array([int(c) for c in org\_colors[source\_idx][4:-1].split(',')])

target\_color = np.array([int(c) for c in lang\_colors[target\_idx][4:-1].split(',')])

# 增加颜色的饱和度

mix\_color = np.minimum(255, (source\_color + target\_color) \* 0.7)

link\_colors.append(f'rgba({int(mix\_color[0])},{int(mix\_color[1])},{int(mix\_color[2])},0.6)')

# 创建桑基图

fig7 = go.Figure(data=[go.Sankey(

node = dict(

pad = 15,

thickness = 20,

line = dict(

color = "rgba(0,0,0,0)", # 完全透明的边框

width = 0

),

label = nodes,

color = org\_colors + lang\_colors,

customdata = ['组织'] \* len(all\_orgs) + ['语言'] \* len(all\_langs),

hovertemplate='%{label}<br>类型: %{customdata}<br>连接数: %{value}<extra></extra>',

textfont = dict(color='white', size=12, family='Arial')

),

link = dict(

source = links['source'],

target = links['target'],

value = links['value'],

color = link\_colors,

hovertemplate='%{source.label} → %{target.label}<br>数量: %{value}<extra></extra>'

)

)])

# ... 其他代码保持不变 ...

主要修改：

使用了更高饱和度的 RGB 颜色值

增加了颜色的对比度

调整了连接的透明度为 0.6，使其更加醒目

使用了纯色而不是渐变色，让颜色更加鲜艳

确保了源节点和目标节点的颜色搭配协调

这样修改后，整个桑基图会显得更加鲜艳和生动。如果觉得某些颜色还不够鲜艳，我们可以继续调整 RGB 值。

A：还是会出现数组下标错误。我有10个org，21个language，请据此为我设置一个大小为10的org色彩数组和大小为21的language色彩数组，并且直接对应，这样就不会出现下标问题。同时让颜色更鲜艳一点，对比更鲜明一点。尽量避免采用暗色

C：好的，让我们根据具体的节点数量设置颜色数组：

# 为10个组织设置鲜艳的颜色

org\_colors = [

'rgb(255,0,0)', # 鲜红

'rgb(255,165,0)', # 橙色

'rgb(255,255,0)', # 鲜黄

'rgb(0,255,0)', # 鲜绿

'rgb(0,255,255)', # 青色

'rgb(0,128,255)', # 亮蓝

'rgb(255,0,255)', # 品红

'rgb(255,105,180)', # 热粉红

'rgb(255,215,0)', # 金色

'rgb(0,191,255)' # 深天蓝

]

# 为21个语言设置鲜艳的颜色

lang\_colors = [

'rgb(255,69,0)', # 红橙色

'rgb(255,140,0)', # 深橙色

'rgb(255,215,0)', # 金色

'rgb(255,255,0)', # 黄色

'rgb(173,255,47)', # 绿黄色

'rgb(50,205,50)', # 石灰绿

'rgb(0,255,127)', # 春绿

'rgb(0,255,255)', # 青色

'rgb(30,144,255)', # 道奇蓝

'rgb(0,191,255)', # 深天蓝

'rgb(147,112,219)', # 中紫色

'rgb(255,20,147)', # 深粉色

'rgb(255,105,180)', # 热粉色

'rgb(255,192,203)', # 粉红

'rgb(255,160,122)', # 浅鲑鱼色

'rgb(255,127,80)', # 珊瑚色

'rgb(255,99,71)', # 番茄色

'rgb(255,64,64)', # 红色

'rgb(255,128,0)', # 橙色

'rgb(255,215,0)', # 金色

'rgb(152,251,152)' # 淡绿色

]

# 简化连接颜色的设置

link\_colors = []

for source, target in zip(links['source'], links['target']):

# 使用源节点的颜色

if source < len(org\_colors):

color = org\_colors[source]

else:

color = lang\_colors[source - len(org\_colors)]

# 添加透明度

color = color.replace('rgb', 'rgba').replace(')', ',0.6)')

link\_colors.append(color)

# 创建桑基图

fig7 = go.Figure(data=[go.Sankey(

node = dict(

pad = 15,

thickness = 20,

line = dict(

color = "rgba(0,0,0,0)", # 完全透明的边框

width = 0

),

label = nodes,

color = org\_colors + lang\_colors, # 直接使用完整的颜色列表

customdata = ['组织'] \* len(all\_orgs) + ['语言'] \* len(all\_langs),

hovertemplate='%{label}<br>类型: %{customdata}<br>连接数: %{value}<extra></extra>',

textfont = dict(color='white', size=12, family='Arial')

),

link = dict(

source = links['source'],

target = links['target'],

value = links['value'],

color = link\_colors,

hovertemplate='%{source.label} → %{target.label}<br>数量: %{value}<extra></extra>'

)

)])

主要改进：

精确设置了10个组织的颜色

精确设置了21个语言的颜色

所有颜色都选择了明亮、鲜艳的色调

避免了使用暗色

确保了颜色数组的长度与节点数量完全匹配

简化了连接颜色的设置逻辑

这样应该不会再出现数组下标错误，同时图表会更加鲜艳醒目。

A：# 设置页面配置

st.set\_page\_config(layout="wide", page\_title="Github Top 200 用户数据分析")

# 设置页面背景颜色

page\_bg\_color = """

<style>

[data-testid="stAppViewContainer"] {

background-color: #222222;

}

[data-testid="stHeader"] {

background-color: #333333;

}

[data-testid="stToolbar"] {

right: 2rem;

}

</style>

"""

此处如何将背景设置为图片？

C：要将 Streamlit 页面背景设置为图片，你可以修改 background-color 为 background-image，并指定图片的 URL 或路径。以下是如何将背景设置为图片的示例：

# 设置页面配置

st.set\_page\_config(layout="wide", page\_title="Github Top 200 用户数据分析")

# 设置页面背景为图片

page\_bg\_image = """

<style>

[data-testid="stAppViewContainer"] {

background-image: url('https://example.com/your-image.jpg'); # 替换为你的图片 URL 或路径

background-size: cover; # 图片覆盖整个页面

background-position: center center; # 图片居中

background-repeat: no-repeat; # 防止图片重复

}

[data-testid="stHeader"] {

background-color: rgba(51, 51, 51, 0.8); # 可选：修改标题背景的透明度

}

[data-testid="stToolbar"] {

right: 2rem;

}

</style>

"""

# 应用背景样式

st.markdown(page\_bg\_image, unsafe\_allow\_html=True)

A：请结合数据和我提供的图片中由上述数据绘制出的图表，为我逐张图表分析数据

C：以下是对各张图表的数据分析：

第一张图表：Top 200 Github 用户地理分布

图表类型：世界地图

数据内容：通过不同颜色（红色、黄色、橙色）标记不同国家和地区，显示了 Github Top 200 用户在全球的分布情况。蓝色点标记具体用户位置。

分析：

从图中可以看出，Github Top 200 用户主要集中在北美（红色区域）、欧洲（黄色区域）和亚洲部分地区（橙色区域）。

北美地区的用户数量最多，尤其是美国，显示出美国在开源领域的主导地位。

亚洲的中国和日本有较多用户，反映了这些国家在技术领域的活跃度。

欧洲有不少国家也有较高的用户分布，表明欧洲在开源社区的参与度也较高。

第二张图表（左）：Top 200 用户的关注者数量和 2024 年贡献数对比

图表类型：水平双向柱状图

数据内容：对比了 Top 200 用户的关注者数量和 2024 年贡献数。

分析：

可以看到部分用户关注者数量多，但 2024 年贡献数相对较少，这些用户可能是有较高知名度但近期活跃度不高的开发者。

有部分用户虽然关注者数量少，但 2024 年贡献数高，这些可能是专注于代码贡献的活跃开发者。

整体上，关注者数量和贡献数有一定相关性，但并非完全线性相关，说明用户影响力来源多样。

第二张图表（右）：Github Top 200 用户国家分布

图表类型：柱状图

数据内容：展示了 Github Top 200 用户在各个国家的分布情况。

分析：

美国用户数量遥遥领先，占据了很大比例，反映了其在开源社区的核心地位。

中国紧随其后，显示中国开发者在 Github 上的活跃度不断提升。

其他国家如英国、德国等也有一定数量的用户，体现了开源社区的全球性。

第三张图表（左）：仓库数量分布

图表类型：直方图

数据内容：展示了 Top 200 用户的仓库数量分布情况。

分析：

大部分用户的仓库数量集中在一定范围内，少数用户拥有较多仓库，这些可能是经验丰富、项目众多的开发者或开源项目团队。

从图中可以大致看出仓库数量的分布区间和集中趋势。

第三张图表（右）：用户加入 Github 年份分布

图表类型：折线图和矩形图组合

数据内容：展示了用户加入 Github 的年份分布，包括每年加入的用户数量以及总体趋势。

分析：

早期加入 Github 的用户数量较少，随着时间推移，用户数量逐渐增加。

近年来用户增长速度有所变化，可能与开源文化普及、行业发展趋势以及 Github 平台自身推广和优化等因素相关。

第四张图表（左）：Top 10 组织与编程语言关系

图表类型：桑基图

数据内容：展示了 Top 10 组织与编程语言之间的关系，包括组织使用的编程语言种类以及编程语言在不同组织中的应用情况。

分析：

一些大型组织使用多种编程语言，体现了其在多技术领域的布局和项目需求多样性。

某些编程语言在多个组织中广泛应用，表明其通用性和重要性。

第四张图表（右）：国家与编程语言分布

图表类型：旭日图

数据内容：展示了不同国家的用户使用编程语言的分布情况。

分析：

不同国家在编程语言使用上有偏好差异，例如某些国家在特定编程语言上使用比例较高，可能与该国教育体系、产业结构和技术发展方向有关。

第五张图表（左）：用户多维度 3D 分布

图表类型：3D 散点图

数据内容：展示了用户在关注者数量、仓库数量、2024 年贡献数和 Stars 数四个维度上的分布情况。

分析：

可以观察到不同用户在各个维度上的表现差异，一些用户在多个维度上都表现出色，而另一些用户可能在某个维度上具有突出优势，但在其他维度上相对较弱。

第五张图表（右）：Github Top 200 用户影响力分布 (随关注者排名)

图表类型：趋势图和散点图结合

数据内容：通过计算用户的综合影响力分数（考虑关注者数量、贡献数、仓库数量和注册年限等因素），展示用户影响力随关注者排名的变化情况。

分析：

影响力分数较高的用户通常位于趋势线的上方，他们在多个方面综合表现优秀。

影响力分数较低的用户则分布在趋势线下方，可能需要在某些方面进一步提升。

以上就是本设计中AI对话的关键部分。后续根据代码结构进行了配色以及布局上的微调，此处不一一列出。

1. 结论与展望
2. 研究成果总结

本研究通过对Github关注量Top200用户数据的收集、清洗、转换和可视化分析，成功实现了多维度数据的直观展示。从地理分布、贡献情况、技术偏好等多个方面深入剖析了用户特征，为理解Github用户群体的行为模式和平台生态提供了丰富信息。通过多种可视化图表的设计与实现，展示了Python在数据可视化领域的强大功能和应用灵活性，为相关研究和实践提供了有益参考。

1. 研究局限性

·数据完整性：尽管努力获取了多方面的用户数据，但仍可能存在部分数据未被完全收集或因平台限制无法获取的情况，这可能影响分析结果的全面性。例如，一些用户的历史贡献数据可能不完整，导致对其长期贡献趋势的评估不够准确。·分析维度局限性：本研究主要关注了用户的部分关键维度信息，对于其他潜在重要因素（如用户所属行业、项目类型、协作模式等）未进行深入分析。这些因素可能对用户行为和影响力产生重要影响，未来研究可进一步拓展分析维度。·可视化方法优化空间：在可视化设计过程中，虽然尽力选择了合适的图表类型和交互功能，但仍存在一些可优化之处。例如，对于复杂的桑基图和旭日图，可能需要进一步改进布局和交互方式，以提高用户对数据关系的理解效率；部分图表在大数据量情况下的性能表现有待提升。

1. 未来研究方向

·拓展数据来源与维度：进一步整合更多Github平台数据以及外部相关数据，如项目代码质量评估数据、用户在其他开源平台的活动数据等，从更广泛的视角深入分析用户行为和平台生态。同时，挖掘更多用户行为相关维度，如代码审查活动、社区参与度等，构建更全面的用户画像。·深入分析用户行为模式：基于更丰富的数据，运用数据挖掘和机器学习技术，深入研究用户行为模式和影响力传播机制。例如，预测用户未来的贡献趋势、识别潜在的高影响力用户群体、分析不同用户行为对项目成功的影响等，为平台运营和开发者决策提供更精准的支持。·优化可视化技术与交互体验：持续探索新的可视化技术和方法，如结合虚拟现实（VR）/增强现实（AR）技术提供沉浸式数据可视化体验、利用动画效果展示数据动态变化过程、开发更智能的交互功能（如语音控制、自动推荐分析视角等），以提升用户对复杂数据的理解和探索能力。·跨平台与跨领域比较研究：将Github用户数据与其他类似开源平台或相关领域（如学术界的科研合作网络、企业内部的代码管理平台等）的数据进行对比分析，探索不同平台和领域之间用户行为、技术偏好和社区结构的异同点，为跨平台协作、知识共享和技术创新提供新的思路和方法。

1. 对计算机可视化领域的贡献

本研究通过实际项目展示了如何运用Python语言和相关可视化库，对复杂的Github用户数据进行全面、深入的可视化分析。在数据处理方面，提供了一套完整的数据清洗、转换和整合方法，为处理类似结构的数据提供了参考范例。在可视化设计上，通过多种定制化图表的实现，展示了如何根据数据特点和分析需求选择合适的可视化类型，并通过交互功能增强用户对数据的探索能力。研究成果有助于推动计算机可视化技术在开源社区分析、软件开发项目管理、开发者行为研究等领域的应用，为相关领域的研究人员和从业者提供了实践经验和技术借鉴，促进了计算机可视化领域与实际应用场景的深度融合。

综上所述，本研究在Github关注量Top200用户数据可视化分析方面取得了一定成果，但仍有诸多可拓展和优化的空间。未来研究将朝着更全面、深入、智能的方向发展，不断挖掘数据价值，提升可视化效果和用户体验，为开源社区和软件开发领域的发展提供有力支持。