Bilibili平台视频分析

班　级： 计算机23A4　　**专　业：** 计算机科学与技术

学　号： 2023013090　　**姓　名：** **蔚嘉琪**

一、引言

1.1 课题背景及研究意义

1.1.1 Bilibili平台的快速发展及现状

Bilibili（简称B站）作为中国年轻用户主导的综合性视频社区平台，已从早期的ACG（动画、漫画、游戏）小众文化社区演变为覆盖多元内容形态的数字文化平台。截至2025年第一季度，B站月均活跃用户（MAU）达3.68亿，日均活跃用户（DAU）突破1.07亿，用户日均使用时长增至108分钟，三项数据均创历史新高。平台用户呈现出显著的年轻化特征，约80%为18–30岁的年轻群体，其中24岁以下用户占比超过50%，本科及以上学历用户占比高达70%。这种用户结构使B站成为观察中国年轻世代文化消费偏好的重要窗口。值得注意的是，平台用户性别比例正趋向均衡（男性52%，女性48%），女性用户增速明显，尤其在美妆、娱乐分区占比超过60%。

B站的内容生态构建呈现出**专业性与多样性并重**的特点。平台已形成36个一级分区，圈层文化标签超200万个，2024年视频日均播放量达到48亿次。其内容体系包含四个核心组成部分：

- **OGV（专业机构生产内容）**：如纪录片《守护解放西》《闪闪的儿科医生》、国创动画《中国奇谭》《时光代理人》等IP化内容，形成从“精品内容”到“长青IP”再到“全网现象级IP”的孵化路径。
- **PUGC（专业用户生产内容）**：由专业内容创作者（如“百大UP主”）生产的高质量视频，2024年数据显示近九成百大UP主持续更新超5年。
- **UGC（用户生成内容）**：普通用户创作的多元化内容，覆盖生活分享、知识科普等领域。

- **AIGC（人工智能生成内容）**：2024年AI内容日均产量同比增长55%，日均播放量超2000万次，成为增长最快的内容形态。

表：B站核心用户画像特征（2025年）

特征维度	构成比例	显著特点
年龄分布	18–30岁占80%	平均年龄26岁，24岁以下超50%
地域分布	一线及新一线城市占40%	三线以下城市增速25%（生活区、三农内容）
学历构成	本科及以上70%	科技兴趣用户在985/211高校渗透率达82%
消费特征	月均付费用户3200万	带货GMV年增长150%+（服装、数码、快消）

1.1.2 数据分析在视频平台发展中的核心价值

在视频平台的精细化运营时代，**多维度数据分析**已成为理解用户行为、优化内容创作、提升商业价值的关键手段。B站用户的兴趣分布极为广泛，从传统ACG文化到新兴的科技、母婴、健康领域均有涉猎：

- **科技内容**：2025年Q1播放时长增长130%，如AI智能体解析视频获44万+播放
- **生活领域**：母婴亲子类内容播放量同比上涨76%，家居家装、美妆时尚等内容快速增长
- **国创动画**：用户累计观看时长超7亿小时，互动量达50亿次

面对如此庞杂的内容生态，传统人工分析难以捕捉深层的用户偏好与内容趋势。通过**爬虫技术与可视化分析相结合**的研究方法，能够系统性地解构以下关键问题：

- 不同视频类型（如Vlog、动画、知识科普等）的热度分布规律
- 用户互动行为（播放、弹幕、评论）与内容属性的关联性
- 新兴内容赛道（如AIGC）的增长轨迹与用户接受度
- 地域、年龄、性别等因素对内容偏好的影响权重

1.2 国内外研究现状

1.2.1 短视频数据分析研究概览

全球范围内，**视频平台的数据挖掘**已成为计算机科学、传播学、社会学等多学科交叉研究热点。现有研究主要聚焦于三个维度：

- **内容传播机制**：探究算法推荐（如协同过滤、深度学习模型）对视频分发的影响
- **用户行为模式**：分析观看时长、互动率、分享意愿等行为背后的心理动因
- **商业价值转化**：研究广告植入方式、电商导流路径、付费内容转化率等经济指标

研究方法的演进呈现出**多技术融合**趋势。早期研究主要依赖平台公开API获取有限数据集，而当前先进研究则结合：

- **自动化爬虫技术**：通过Selenium等工具模拟用户行为，绕过反爬机制获取动态加载数据
- **跨模态分析**：同时处理视频、音频、文字（弹幕/评论）等多源异构数据
- **实时计算框架**：采用Storm、Flink等流处理技术捕捉内容传播动态

1.2.2 国内短视频平台研究进展

中国学术界对短视频平台的研究集中于**抖音、快手、B站**三大平台，研究方法与发现各具特色：

- **抖音研究焦点**：
  以**推荐算法机制**为核心，探究“信息茧房”效应与破圈策略。研究发现抖音的瀑布流推荐依赖强兴趣标签，导致垂直类内容（如美妆教程）易获高曝光，但知识类内容需通过“热点绑定”策略突破圈层。用户平均观看决策时间仅1.7秒，标题与封面图成为点击关键因素。
- **B站研究特色**：
  学者更关注**圈层文化**与**社区黏性**的相互作用。研究表明B站的“一键三连”（点赞、投币、收藏）设计构建了独特的价值衡量体系，其中“投币”行为隐含用户对内容创作的物质认可与精神激励。研究还发现，B站用户的“高学历特性”显著影响内容偏好，科技类视频在985高校用户渗透率达82%，知识区内容（如AI科普）播放时长年增长130%。

  内容生产模式研究揭示B站具备**多层次创作生态**：

- **OGV内容**：如纪录片《守护解放西》通过“警务纪实+文旅传播”创新模式实现破圈，带动长沙网红地标打卡热潮
- **PUGC内容**：专业UP主（如“小Lin说”）以年轻化语言解读专业知识，155万播放量的AI科普视频生命周期超3年
- **UGC内容**：用户二创视频形成对OGV内容的补充传播，如《黑神话·悟空》游戏引发的“山西文旅圣地巡礼”现象

表：B站核心内容板块数据分析研究（2025）

内容类型	研究重点	关键发现	数据来源

国创动画	IP开发价值	人均观看10部，43部新作待上线；《凸变英雄X》全球播放破亿	B站平台年报
科技科普	知识传播效能	科技兴趣用户2亿+，AI内容播放增长130%	某白皮书
生活纪实	地域文化传播	《守护解放西》带动长沙旅游搜索增90%	长沙文旅数据
AIGC内容	用户接受度	日均产量同比增55%，用户渗透率60%	某商业报告

二、项目目标与需求分析

2.1 项目目标

1. 自动化采集B站多类型视频的基本信息（包括播放量、评论数、弹幕数、点赞数、投币数、收藏数、时长、标题、分类等指标）；
2. 实现数据的标准化清洗和去重，确保分析基础的准确性；
3. 多角度可视化展示数据特征与内在规律，包括：视频热度分布、播放量排行榜、上传时间趋势、互动行为关联分析、关键词词云等；
4. 通过分析结果，探索用户偏好及平台内容分布，为内容创作者和平台运营提供参考依据。

2.2 功能需求

- **数据采集模块**：能够应对B站接口防爬机制，实现稳定获取大量视频数据（>10,000条）；
- **数据清洗模块**：处理重复记录、缺失字段与不规范格式；
- **数据分析模块**：可灵活支持不同维度下的聚合与统计；
- **可视化模块**：利用Pyecharts生成交互式HTML页面或高质量PNG图表，适配展示与演示需求。

2.3 性能需求

- 采集效率高，支持多线程抓取，能够在数小时内采集完3万+数据量；

- 数据处理耗时合理，兼容中型数据集（<100MB）；
- 生成图表的时间控制在2秒以内，满足交互性能要求。

三、方案设计

3.1 系统总体结构

总体结构图 Plain Text

```
+-----------------+      +-----------------+      +-----------------+
|    数据采集模块    | --> |   数据预处理模块   | --> |   数据可视化模块    |
+-----------------+      +-----------------+      +-----------------+
```

- 爬虫脚本：基于Python requests模块，模拟浏览器行为，分页抓取**排行榜**与**搜索接口**数据；
- 数据清洗：利用pandas库进行字段转换、去重、空值填补；
- 可视化展示：使用pyecharts生成动态交互式图表，并保存为HTML或PNG文件，适用于Web展示与静态报告。

3.2 技术选型

模块	技术栈
爬虫	Python requests, csv
数据处理	pandas, numpy
可视化	pyecharts, WordCloud, jieba
环境及包管理	Sublime, Miniconda, Python3.13

四、模块设计

4.1 数据采集模块

- 采用B站开放接口或半开放接口，通过设置User-Agent、Referer、Cookie等头部信息规避反爬；

- 抓取排行榜中各分区Top1000视频，并通过关键词检索扩大采样范围；
- 设置合理的请求间隔与异常重试机制，保障采集稳定性；
- 最终输出为合并的CSV数据文件。

4.2 数据预处理模块

- 基于唯一视频标识符（BV号）进行去重；
- 将播放量、评论数、点赞等字段统一转为整型数值，方便后续统计；
- 将上传时间字段解析为datetime对象，提取年/月/日等信息；
- 标题、简介字段统一编码并处理异常字符，确保中文兼容性。

4.3 数据分析与可视化模块

子模块	分析内容
A1	基础分布统计（播放量、评论数、弹幕数等的整体分布）
A2	热门视频类型分析，展示B站不同类型视频的播放量
A3	播放量排行榜 Top20 视频，展示最受欢迎的内容与标题关键词
A4	视频上传时间趋势分析，查看内容发布是否呈周期性或时段性分布
A5	播放量与评论数关系分析，绘制散点图，辅助判断互动与热度的关联
A6	视频标题关键词词云，识别高频词汇反映内容偏好趋势
A7	视频时长与播放量的关系，结合散点图与区间柱状图呈现

五、项目实现

5.1 数据采集实现

5.1.1 概述

- 使用requests库构建请求，加入UA、Referer等头信息；
- 实现分页逻辑，逐页访问接口获取视频数据；

- 添加异常捕获机制，对于失败的请求自动重试；
- 将采集的数据保存为CSV格式，并在本地或远程服务器保存备份；

5.1.2 模块实现流程

B站开放接口为 `https://api.bilibili.com/x/web-interface`

爬虫又称为网络爬虫，是一种基于规则对网址中文本、图片等信息进行自动抓取的程序。爬虫通过模拟真实用户，向服务器发送请求，持续对网页数据进行抓取，直到达成某一条件时停止。爬虫的本质是在海量的互联网信息中通过筛选收集有用的信息，最终进行分析整合以供使用。

而恶意的爬虫会发出大量的请求来抓取页面内容，消耗大量的服务器带宽，尤其是在大规模或高频爬取时。同时，处理每个爬虫请求都需要消耗服务器的CPU、内存和I/O资源。大量爬虫请求会显著增加服务器负载，可能导致服务器响应变慢、处理正常用户请求的能力下降，甚至导致服务器崩溃或服务不可用。这直接影响真实用户的体验。

于是，反爬虫技术应运而生。反爬虫是指对扫描器中的网络爬虫环节进行反制，通过一些反制策略来阻碍或干扰爬虫的正常爬行，从而间接地起到防御目的。

但是俗话说“道高一尺，魔高一丈”，既然为了防范爬虫从而产生了反爬虫技术，那么就也会有规避反爬虫技术的方法。简单来说，就是模仿正常人的操作；对于计算机，就是模仿正常的指令发送。那么具体的反爬机制和规避方法如下：

表：Bilibili平台常见的反爬机制及规避方法

反爬机制	描述	规避办法
UA 检测	检查你是不是浏览器发起请求	设置 headers 中的 `User-Agent`
请求频率限制	请求太快会被断流或返回异常	`time.sleep()` 间隔请求
IP 限制（轻微）	某个 IP 多次请求后数据变少或返回错误	不并发请求、不频繁爬太多页
登录接口校验	如你访问用户空间、投稿、弹幕等接口会校验登录+签名	避免！只用公开 API

那么按照上述方法，编写Python爬虫代码如下：

Bilibili_v1.py Python

```python
import requests
import pandas as pd
import time
import random
import os

# 设置 UA
headers = {
    'User-Agent': ('Mozilla/5.0 (Windows NT 10.0; Win64; x64) '
                   'AppleWebKit/537.36 (KHTML, like Gecko) '
                   'Chrome/122.0.0.0 Safari/537.36'),
    'Referer': 'https://www.bilibili.com',
    'Accept': 'application/json, text/plain, */*'
}

# 视频爬取函数
def crawl_bilibili_videos(keyword, pages=3):
    base_url = "https://api.bilibili.com/x/web-interface/search/type"
    results = []

    for page in range(1, pages + 1):
        params = {
            'search_type': 'video',
            'keyword': keyword,
            'page': page
        }

        try:
            response = requests.get(base_url, headers=headers, params=para
ms, timeout=10)
            data = response.json()

            if 'data' not in data or 'result' not in data['data']:
                print(f"[!] 数据为空或被限速，关键词：{keyword} 第{page}页")
                continue

            for item in data['data']['result']:
                results.append({
                    'title': item['title'],
                    'author': item['author'],
                    'view': item.get('play', '0'),
                    'danmaku': item.get('video_review', '0'),
                    'description': item.get('description', ''),
                    'pubdate': item.get('pubdate', ''),
```

```
                    'type': item.get('typename', ''),
                    'arcurl': item['arcurl']
                })

            print(f"[+] 成功爬取 {keyword} 第{page}页")
            time.sleep(random.uniform(1.5, 3.5))  # 防止被限速
        except Exception as e:
            print(f"[-] 错误：{e}，关键词：{keyword} 第{page}页")
            continue

    return pd.DataFrame(results)

# 多关键词批量爬取
keywords = ['游戏', '知识', '娱乐', '美食', '音乐']  # 可自定义关键词
save_dir = 'bilibili_data'
os.makedirs(save_dir, exist_ok=True)

for kw in keywords:
    df = crawl_bilibili_videos(kw, pages=20)  # 每个关键词爬5页
    save_path = os.path.join(save_dir, f'{kw}.csv')
    df.to_csv(save_path, index=False)
    print(f"[!] 已保存 {kw}.csv，共 {len(df)} 条数据")
    time.sleep(random.uniform(2.5, 4.5))  # 每类间歇防止封IP

print("\n[!] 所有关键词爬取完成！数据保存在 bilibili_data 文件夹 ")
```

上述代码为第1版代码，英文为version1，所以文件名为Bilibili_v1.py。

代码运行结果如下：

```
(base) → VisualData python Bilibili.py
[!] 数据为空或被限速，关键词：游戏 第1页
[!] 数据为空或被限速，关键词：游戏 第2页
[!] 数据为空或被限速，关键词：游戏 第3页
[!] 数据为空或被限速，关键词：游戏 第4页
[!] 数据为空或被限速，关键词：游戏 第5页
[!] 数据为空或被限速，关键词：游戏 第6页
[+] 成功爬取 游戏 第7页
[!] 数据为空或被限速，关键词：游戏 第8页
[!] 数据为空或被限速，关键词：游戏 第9页
[!] 数据为空或被限速，关键词：游戏 第10页
```

可以看到，即使规避了反爬虫策略，在爬虫过程中依然会被拦截。那么根据上面所说的原则“我们要尽可能的让我们的操作更像人类，而不是人机”可以说明，我们目前的操作依然被B站识别为了机器人，从而被触发了拦截策略。那么接下来我们要进行部分改进。

上述代码（v1）是按照分类视频爬取数据，而我们正常人类登陆B站之后，第一个看到的和浏览的一定是推荐页的内容，俗称主页。那么我们将访问的地址改为主页，只获取推荐的视频，是不

是就会离“人类的操作”更加的靠近了呢?

按照思路编写第2版代码如下:

Bilibili_v2.py Python

```python
import requests
import pandas as pd
import time
import os

headers = {
    'User-Agent': ('Mozilla/5.0 (Windows NT 10.0; Win64; x64) '
                   'AppleWebKit/537.36 (KHTML, like Gecko) '
                   'Chrome/122.0.0.0 Safari/537.36'),
    'Referer': 'https://www.bilibili.com'
}

# 分类名与rid的映射（官方）
type_map = {
    1: "动画",
    3: "音乐",
    4: "游戏",
    5: "娱乐",
    36: "知识",
    160: "生活",
    119: "鬼畜",
    129: "舞蹈"
}

def get_ranking_videos(rid):
    url = f"https://api.bilibili.com/x/web-interface/ranking/v2"
    params = {'rid': rid, 'type': 'all'}
    try:
        res = requests.get(url, headers=headers, params=params, timeout=10)
        res.raise_for_status()
        data = res.json()

        result = []
        for item in data['data']['list']:
            stat = item.get('stat', {})
            result.append({
                'title': item['title'],
                'author': item['owner']['name'],
                'type': type_map.get(rid, str(rid)),
                'view': stat.get('view', 0),
                'like': stat.get('like', 0),
                'coin': stat.get('coin', 0),
                'favorite': stat.get('favorite', 0),
```

```
                    'share': stat.get('share', 0),
                    'danmaku': stat.get('danmaku', 0),
                    'pubdate': item.get('pubdate', ''),
                    'bvid': item.get('bvid', ''),
                    'arcurl': f"https://www.bilibili.com/video/{item['bvid']}"
                })

            return pd.DataFrame(result)

        except Exception as e:
            print(f"[-] 获取 rid={rid} 失败：{e}")
            return pd.DataFrame()

    # 多分类爬取
    save_dir = "bilibili_rank_data"
    os.makedirs(save_dir, exist_ok=True)

    for rid in type_map.keys():
        df = get_ranking_videos(rid)
        df.to_csv(os.path.join(save_dir, f"{type_map[rid]}.csv"), index=False)
        print(f"[!] 已保存 {type_map[rid]}.csv，{len(df)} 条")
        time.sleep(30)

    print("\n[!] 所有排行榜数据获取完成！")
```

上述代码为第2版代码，英文为version2，所以文件名为Bilibili_v2.py。

两版代码的爬虫访问目标对比如下：

```
# 视频爬取函数
def crawl_bilibili_videos(keyword, pages=3):
    base_url = "https://api.bilibili.com/x/web-interface/search/type"
    results = []

    for page in range(1, pages + 1):
```

v1

```
def get_ranking_videos(rid):
    url = f"https://api.bilibili.com/x/web-interface/ranking/v2"
    params = {'rid': rid, 'type': 'all'}
    try:
        res = requests.get(url, headers=headers, params=params, timeout=10)
        res.raise_for_status()
```

v2

第2版运行结果如下图所示：

```
(base) → VisualData python Bilibili_v2.py
[!] 已保存 动画.csv，97 条
[!] 已保存 音乐.csv，97 条
[!] 已保存 游戏.csv，99 条
```

如上可见，第二版代码能完美的保存分类数据，并且在爬虫的过程中不会触发反爬策略。但是每一个分类只有不到100条数据，这样的数据量实属渺小，根本不足以支持我们对于视频平台整体的分析的概括。那么需要获取更多的数据，于是对第2版代码中的API接口进行深入分析，并没有发现其他推荐流接口。

于是查阅B站API文档，发现可以访问 `wbi/index/top/feed/rcmd` 进行推荐流的分页处理，这样每页会有20条视频数据，每个分类的爬取上限为50页，也就是1000条视频，是原来数据量的10倍。这样的视频数量就能很好的支撑整个平台的情况。

遵循上述的说明优化第2版代码如下：

Bilibili_v3.py Python

```python
import requests
import pandas as pd
import time
import os

HEADERS = {
    'User-Agent': ('Mozilla/5.0 (Windows NT 10.0; Win64; x64) '
                   'AppleWebKit/537.36 (KHTML, like Gecko) '
                   'Chrome/122.0.0.0 Safari/537.36'),
    'Referer': 'https://www.bilibili.com/',
    'Origin': 'https://www.bilibili.com',
    'Accept': 'application/json, text/plain, */*',
    'Accept-Language': 'zh-CN,zh;q=0.9',
    'Connection': 'keep-alive',
    'Sec-Fetch-Dest': 'empty',
    'Sec-Fetch-Mode': 'cors',
    'Sec-Fetch-Site': 'same-site'
}

# 保存目录
SAVE_DIR = "bilibili_data"
os.makedirs(SAVE_DIR, exist_ok=True)

TYPE_MAP = {
    1: "动画", 3: "音乐", 4: "游戏", 5: "娱乐",
    36: "知识", 160: "生活", 119: "鬼畜", 129: "舞蹈"
}
def get_rank_videos():
    print("\n[!] 正在获取排行榜数据...")
    all_dfs = []

    for rid, typename in TYPE_MAP.items():
        url = "https://api.bilibili.com/x/web-interface/ranking/v2"
        params = {'rid': rid, 'type': 'all'}

        try:
            res = requests.get(url, headers=HEADERS, params=params, timeo
ut=30)

            #Dubugging for Empty Data
            print(f"[调试] rid={rid} 状态码: {res.status_code}")
            #print(f"[调试] 内容摘要（前500字符）: \n{res.text[:500]}\n")

            res.raise_for_status()
```

```
            data = res.json()

            if 'data' not in data or 'list' not in data['data']:
                print(f"[-] 分类【{typename}】数据为空")
                continue

            result = []
            for item in data['data']['list']:
                stat = item.get('stat', {})
                result.append({
                    'title': item['title'],
                    'author': item['owner']['name'],
                    'type': typename,
                    'view': stat.get('view', 0),
                    'like': stat.get('like', 0),
                    'coin': stat.get('coin', 0),
                    'favorite': stat.get('favorite', 0),
                    'share': stat.get('share', 0),
                    'danmaku': stat.get('danmaku', 0),
                    'pubdate': item.get('pubdate', ''),
                    'bvid': item.get('bvid', ''),
                    'arcurl': f"https://www.bilibili.com/video/{item['bvi
d']}"
                })

            df = pd.DataFrame(result)
            df.to_csv(os.path.join(SAVE_DIR, f"rank_{typename}.csv"), ind
ex=False)
            all_dfs.append(df)
            print(f"[+] 分类【{typename}】完成，{len(df)} 条")
            time.sleep(10)

        except Exception as e:
            print(f"[-] 分类【{typename}】失败：{e}")

    # 合并所有排行榜数据
    final_df = pd.concat(all_dfs, ignore_index=True)
    final_df.to_csv(os.path.join(SAVE_DIR, "rank_all.csv"), index=False)
    print(f"\n[!] 排行榜总计 {len(final_df)} 条，已保存为 rank_all.csv")
    return final_df

def get_rcmd_videos(pages=100):
    print("\n[+] 正在获取推荐流视频（分页）...")
    result = []

    for page in range(1, pages + 1):

```

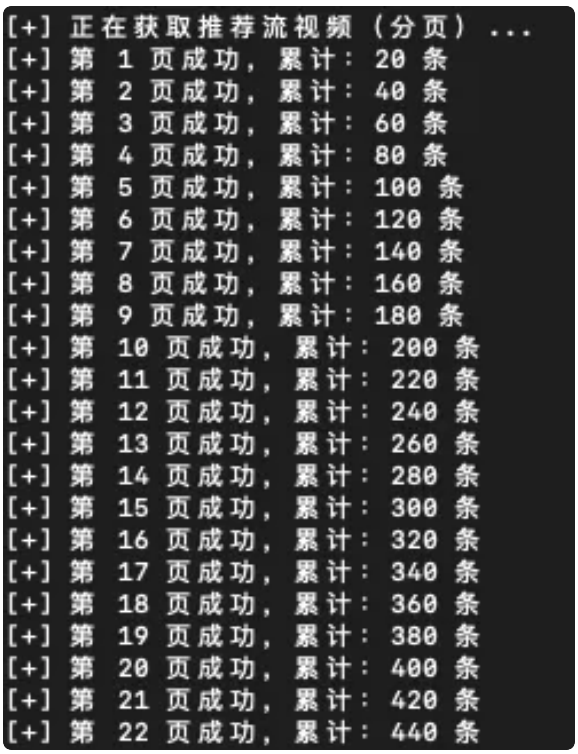
```
        url = "https://api.bilibili.com/x/web-interface/wbi/index/top/feed/rcmd"
        params = {
            'ps': 20,
            'pn': page,
            'fresh_type': 4,
            'feed_version': 'V2',
            'platform': 'web'
        }

        try:
            res = requests.get(url, headers=HEADERS, params=params, timeout=30)
            data = res.json()

            if data['code'] != 0:
                print(f"[-] 第 {page} 页请求失败，code={data['code']}")
                continue

            for item in data['data']['item']:
                stat = item.get('stat', {})
                result.append({
                    'title': item['title'],
                    'author': item['owner']['name'],
                    'view': stat.get('view', 0),
                    'like': stat.get('like', 0),
                    'coin': stat.get('coin', 0),
                    'favorite': stat.get('favorite', 0),
                    'danmaku': stat.get('danmaku', 0),
                    'pubdate': item.get('pubdate', ''),
                    'bvid': item.get('bvid', ''),
                    'arcurl': f"https://www.bilibili.com/video/{item['bvid']}"
                })

            print(f"[+] 第 {page} 页成功，累计：{len(result)} 条")
            time.sleep(5)

        except Exception as e:
            print(f"[-] 第 {page} 页失败：{e}")
            continue

    df = pd.DataFrame(result)
    df.to_csv(os.path.join(SAVE_DIR, "rcmd_all.csv"), index=False)
    print(f"\n[!] 推荐流共采集 {len(df)} 条，已保存为 rcmd_all.csv")
    return df

```

```
133   if __name__ == "__main__":
134       #rank_df = get_rank_videos()
135       rank_df = 0
136       rcmd_df = get_rcmd_videos(pages=100)
137
          print(f"\n[!] 数据采集完成，共采集：{len(rank_df) + len(rcmd_df)} 条视频信
```

特殊说明：第39行--第41行的Debug代码是因为第1次运行的时候返回数据全部为空白，于是加入了Debug代码进行调试，查看原返回报文是否有误。添加后运行报文数据正常，后续运行数据均正常。为了确保稳定，保留了第40行的状态码显示。目前尚不知情空白数据出现的原因。

上述代码为第3版代码，英文为version3，所以文件名为Bilibili_v3.py。

第3版代码运行部分结果如下图：



可以看到代码稳定运行，没有出现报错等其余信息，反爬虫规避也在预期之中。

正常来说到这里就结束了，但是当天晚上我多看了一眼爬虫爬到的数据，发现所有的数据均为重复数据。单个分类获得的1000条数据中，只有前20条为有效数据，剩余的980条数据均为这20条数据的重复循环。

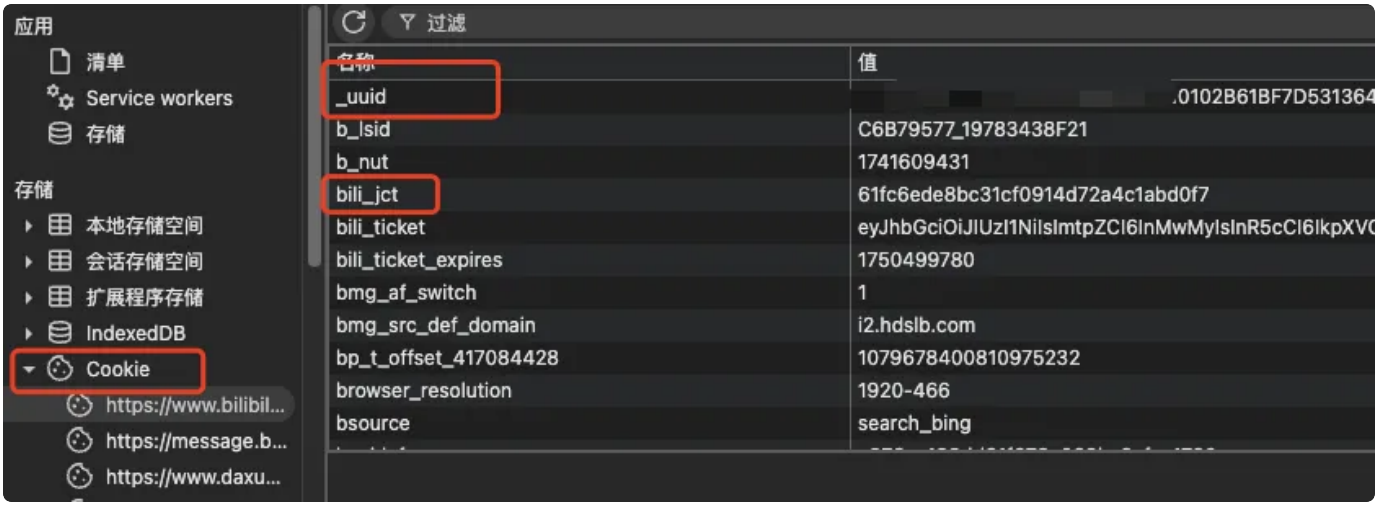
title	author	view	like	coin	favorite
探访阿富汗最硬核的集市，全是美军留下的装备，居然都是白菜价	小钟Johnny	308[illegible]	15449	0	0
一期十分钟的自我介绍。	纪视录	568[illegible]	[illegible]	0	0
单 依 纯 抗 日	打不倒の西撒	546683	18071	0	0
“一首《起风了》带你回到十年前的夏天”	爱狂三的星总	99918	28847	0	0
墨西哥边境毒枭小镇真是地狱？对面美国反而更破败！	街头小小小霸王	81786	4441	0	0
【第五人格全角色】萌娃幼儿园系列	南南南波大王	206838	13259	0	0
氪金五万沦为路边一条，霸服蔚蓝星球还有希望吗	老芒果OL	304382	29714	0	0
我被抄袭了	Serin塞琳	1378155	98592	0	0
这位被官方认可的PS女王，把自己P进梦里！	艺术party	1107528	75675	0	0
走进阿富汗最震撼博物馆，中国大哥阿富汗卖拉面，持枪保镖不离身	小钟Johnny	692876	28351	0	0
「小白」2025手机系统Bug大调查！哪家Bug多？	小白测评	564063	28562	0	0
每个人都有一个武侠梦，我实现了，你们呢？	余铁匠打铁日常	96707	7557	0	0
这 扯 不 扯	圣主帮	134[illegible]	[illegible]	0	0
当我把崩铁和王者结合是什么体验？第2集	王亚索	349300	31930	0	0
【WNS中字】250615 [CHOREOGRAPHY] j-hope Killin' It Girl (Solo Version) Dance Practice	WNS_WeNeedBTS	156369	17432	0	0
你敢起我敢用-号主让我用双管喷清图	战术级子轩	666476	26793	0	0
第25集|钓鱼佬穿越修仙世界，成为半步筑基大修士	虾仁李建国	291076	20318	0	0
【影Sir】欧洲赘婿吃绝户！恋爱脑绝地反杀，劳模姐抖森姐弟大谈特谈，哥特奇幻《猩红山峰》	我是影Sir	180457	14283	0	0
我国首例，脑机接口！科学家研制出全球最小尺寸脑机接口植入体	科学网	916838	85320	0	0
净身高192cm的女孩穿38的鞋，在地铁站太突兀了	高个子账号请看动态	745381	39500	0	0
探访阿富汗最硬核的集市，全是美军留下的装备，居然都是白菜价	小钟Johnny	3[illegible]	[illegible]	0	0
一期十分钟的自我介绍。	纪视录	56[illegible]	[illegible]	0	0
单 依 纯 抗 日	打不倒の西撒	546683	18071	0	0
“一首《起风了》带你回到十年前的夏天”	爱狂三的星总	99918	28849	0	0
墨西哥边境毒枭小镇真是地狱？对面美国反而更破败！	街头小小小霸王	81856	4441	0	0
【第五人格全角色】萌娃幼儿园系列	南南南波大王	206838	13259	0	0
氪金五万沦为路边一条，霸服蔚蓝星球还有希望吗	老芒果OL	304382	29714	0	0
我被抄袭了	Serin塞琳	1378155	98592	0	0
这位被官方认可的PS女王，把自己P进梦里！	艺术party	1107528	75675	0	0
走进阿富汗最震撼博物馆，中国大哥阿富汗卖拉面，持枪保镖不离身	小钟Johnny	692876	28351	0	0
「小白」2025手机系统Bug大调查！哪家Bug多？	小白测评	564063	28562	0	0
每个人都有一个武侠梦，我实现了，你们呢？	余铁匠打铁日常	96707	7557	0	0
这 扯 不 扯	圣主帮	13[illegible]	[illegible]	0	0

上述原因询问了某位不愿意透露姓名的高人得到回复：推荐接口是“**伪分页**”处理，目的是为了缓解流量压力，所以请求参数的变化并不会导致数据的变化。这就意味着，如果你循环访问这个接口（即使加了分页参数），服务器依然返回同一批推荐内容。

根本原因是B站这个接口是为首页推荐准备的，不是为“获取全站数据”准备的。所以我们在v2和v3版本中使用的推荐流接口将无法使用。也就是说现在又回到了v1版本的问题：**如何让我们的操**

作更像人类？而不是人机？

在网络中，为了识别用户信息，通常会加入类似于身份证一样的“唯一识别信息”，例如Cookie，token，uuid等。所以现在要做的就是在我们的访问请求中加入这些信息。就像你打电话给别人，主动告诉你的身份一样。下图为B站的Cookie数据：



接下来按照这个思路编写第4版代码：

Bilibili_v4.py Python

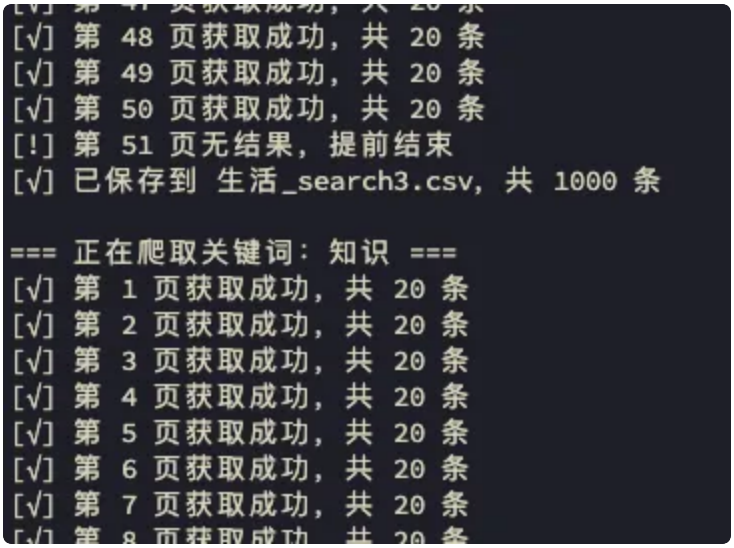
```python
import requests
import csv
import time
import random

HEADERS = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36',
    'Referer': 'https://www.bilibili.com/',
    'Cookie': "_uuid=DA6289104-4AED-D6B8-9F39-510102B61BF7D531364infoc; SESSDATA=9601659c%2C1757173810%2C2bdee%2A32CjDNrJSobl5jcfy9ARW_TFD35GMaHma7mybQEqGrfEjtgu_6lLagkUpDzb4gs2hG_MUSVnhWU0RXQ25nekdGMmNpanNZN0VMVFRQMklBNENjMTItcUFWTTF1UWgwcGpfVkdLYktsZGpkU2dSM1FmNWdDajhybWlTbFY0MUdqbGYxUXhObWptblZnIIEC; bili_jct=61fc6ede8bc31cf0914d72a4c1abd0f7; DedeUserID=417084428",
    'Origin': 'https://www.bilibili.com',
    'Accept': 'application/json, text/plain, */*',
    'Accept-Language': 'zh-CN,zh;q=0.9',
    'Connection': 'keep-alive',
    'Sec-Fetch-Dest': 'empty',
    'Sec-Fetch-Mode': 'cors',
    'Sec-Fetch-Site': 'same-site'
}

def search_videos(keyword, max_pages=20):
    all_results = []
    for page in range(1, max_pages + 1):
        url = "https://api.bilibili.com/x/web-interface/search/type"
        params = {
            "search_type": "video",
            "keyword": keyword,
            "page": page
        }
        try:
            res = requests.get(url, headers=HEADERS, params=params, timeout=30)
            res.raise_for_status()
            data = res.json()

            videos = data.get('data', {}).get('result', [])
            if not videos:
                print(f"[!] 第 {page} 页无结果，提前结束")
                break

            for v in videos:
```

```
                all_results.append({
                    "标题": v.get("title", "").replace("<em class=\"keyword\">", "").replace("</em>", ""),
                    "UP主": v.get("author"),
                    "播放数": v.get("play"),
                    "弹幕数": v.get("video_review"),
                    "评论数": v.get("review"),
                    "时长": v.get("duration"),
                    "发布时间": v.get("pubdate"),
                    "BV号": v.get("bvid"),
                    "链接": f"https://www.bilibili.com/video/{v.get('bvid')}"
                })

            print(f"[√] 第 {page} 页获取成功，共 {len(videos)} 条")
            time.sleep(1)  # 防止被ban

        except Exception as e:
            print(f"[×] 第 {page} 页出错：{e}")
            continue

    return all_results


def save_to_csv(data, filename):
    if not data:
        print("无数据可保存")
        return
    keys = data[0].keys()
    with open(filename, "w", encoding="utf-8-sig", newline="") as f:
        writer = csv.DictWriter(f, fieldnames=keys)
        writer.writeheader()
        writer.writerows(data)
    print(f"[√] 已保存到 {filename}，共 {len(data)} 条")

if __name__ == "__main__":
    keywords = ["动画", "音乐", "游戏", "生活", "知识", "Vlog", "娱乐", "鬼畜", "科普", "电影", "纪录片", "舞蹈"]
    for kw in keywords:
        print(f"\n=== 正在爬取关键词：{kw} ===")
        results = search_videos(keyword=kw, max_pages=20)  # 每个关键词爬10页（约200条）
        save_to_csv(results, f"{kw}_search.csv")
```

上述代码为第4版代码，英文为version4，所以文件名为Bilibili_v4.py。

第4版代码的接口与第1版一样为 `search/type`，在请求报文Header里面加入了用于辨别身份的识别信息“`_uuid`，`SESSDATA`，`bili_jct` 和 `DedeUserID`”。当然，这些信息用的是我自己的。

第4版代码的运行部分结果如下图所示：



到此，爬虫模块已经实现完毕。后续按照v4代码进行数据爬虫，共得到了43000条数据。接下来这些数据就要移交给可视化模块进行数据的可视化处理。

5.2 可视化实现

5.2.1 概述

- Pyecharts图表类型包括柱状图、折线图、散点图、词云图等；
- 所有图表支持缩放、提示框、数据筛选等基本交互功能；
- 利用Grid、Timeline等组件实现复合型展示，增强可读性与互动性；
- 输出HTML便于网页嵌入，输出PNG便于论文插图。

5.2.2 模块实现流程

在进行可视化实现之前，因为标签交叉（即一个视频中会出现多种标签），我们需要对我们的数据进行数据清洗。数据清洗代码过于复杂，同时因为数据量过于庞大，过分占用计算机资源，于是这部分工作我们打算交给ChatGPT去处理，结果如下：

接下来就是按照4.3中的表格进行可视化分析。

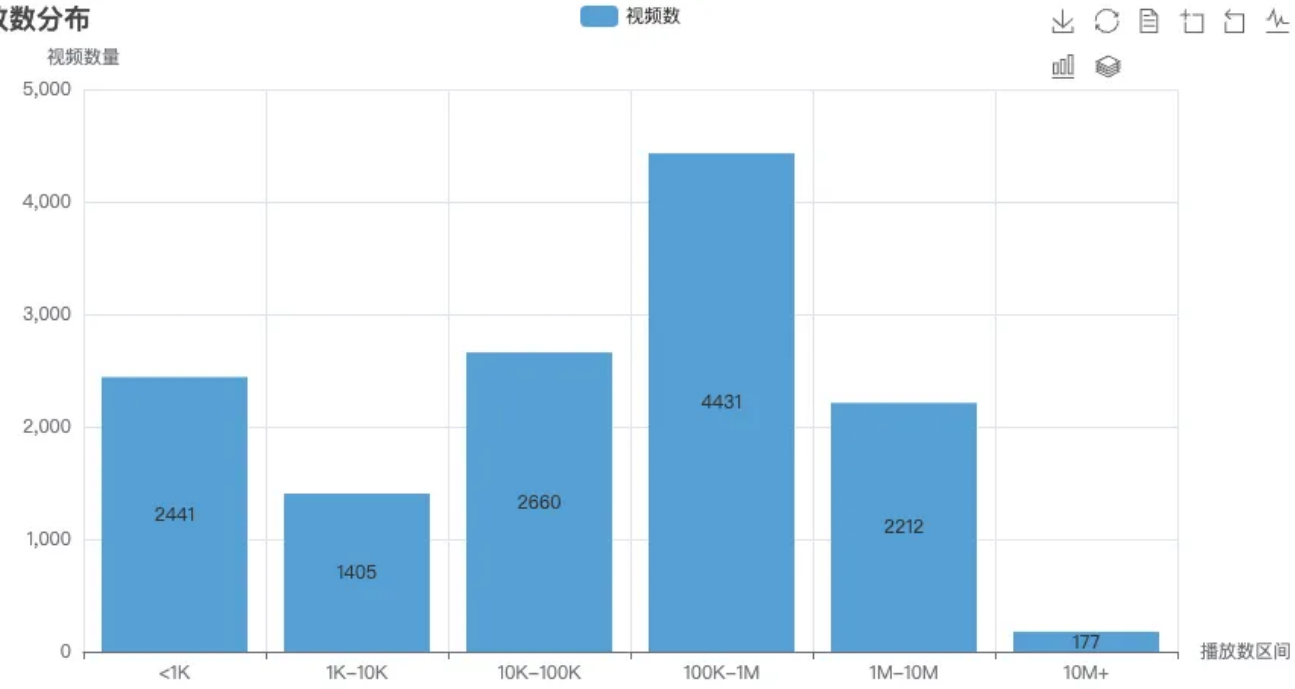
A1 基础分析

BasicAnalysis.py Python

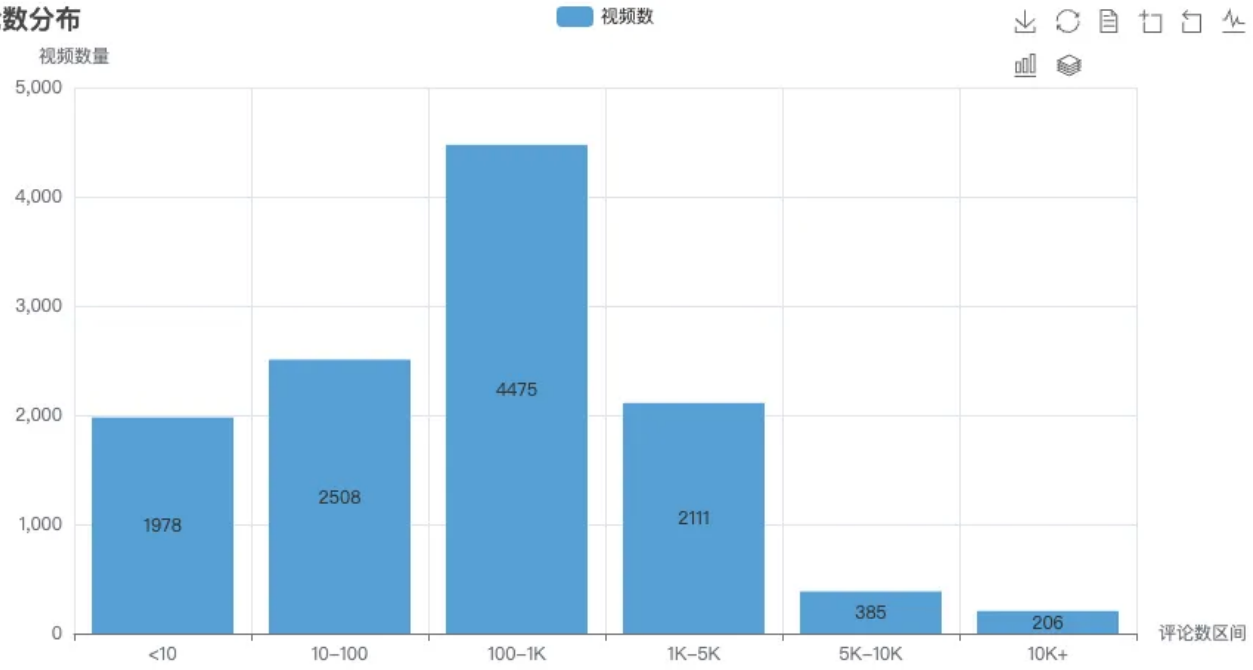
```python
import pandas as pd
import numpy as np
from pyecharts.charts import Bar, Boxplot, Timeline, Line, Page
from pyecharts.options import TitleOpts, ToolboxOpts, LegendOpts, TooltipOpts, AxisOpts, VisualMapOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.replace([np.inf, -np.inf], np.nan)
df = df.dropna(subset=['播放数', '评论数', '弹幕数', '发布时间'])

# 转换时间戳
df['发布时间_dt'] = pd.to_datetime(df['发布时间'], unit='s', errors='coerce')
df = df.dropna(subset=['发布时间_dt'])

# 播放数分布柱状图（分组）
bins = [0, 1000, 10000, 100000, 1000000, 10000000, 1e9]
labels = ['<1K', '1K-10K', '10K-100K', '100K-1M', '1M-10M', '10M+']
df['播放数区间'] = pd.cut(df['播放数'], bins=bins, labels=labels)
view_count_bar = (
    Bar(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(labels)
    .add_yaxis("视频数", df['播放数区间'].value_counts().reindex(labels).fillna(0).astype(int).tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="播放数分布"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="播放数区间"),
        yaxis_opts=AxisOpts(name="视频数量")
    )
)

# 评论数分布（同上）
bins2 = [0, 10, 100, 1000, 5000, 10000, 1e6]
labels2 = ['<10', '10-100', '100-1K', '1K-5K', '5K-10K', '10K+']
df['评论数区间'] = pd.cut(df['评论数'], bins=bins2, labels=labels2)
comment_bar = (
    Bar(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(labels2)

```

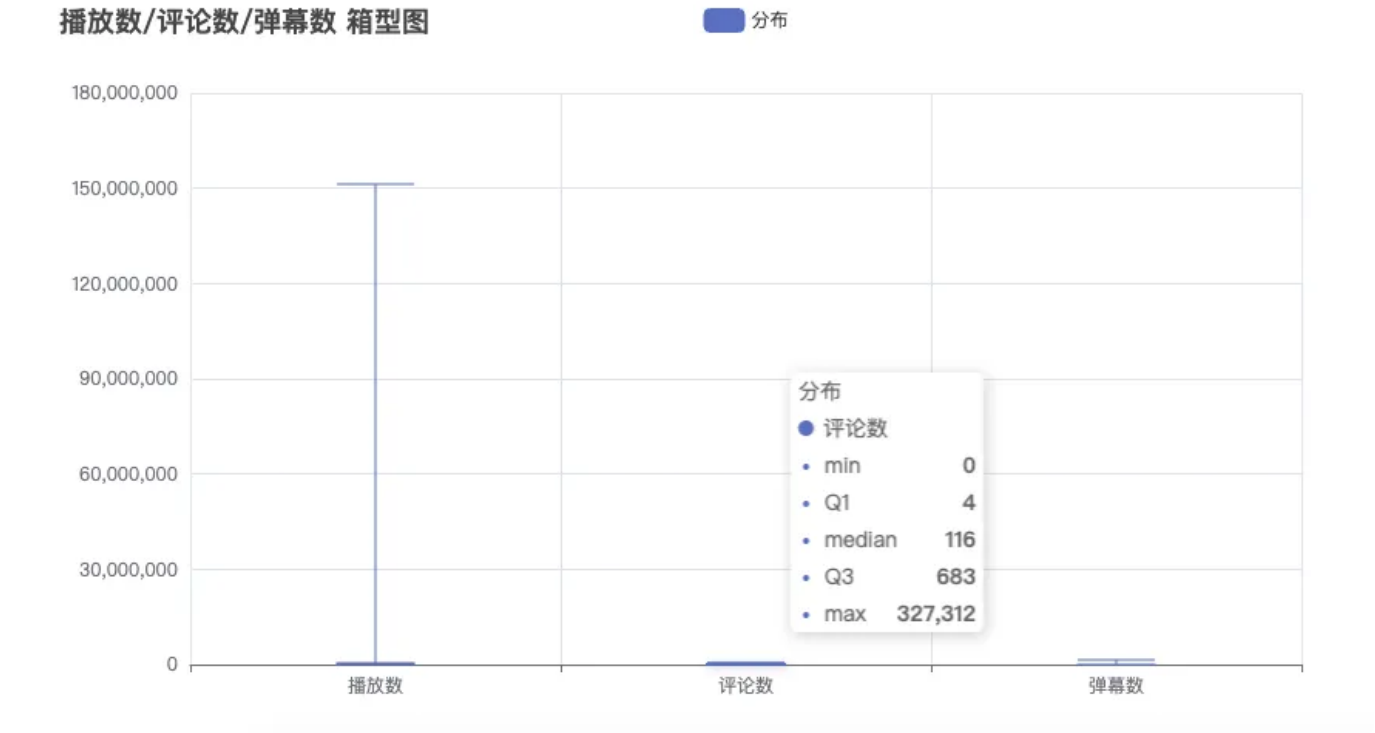
```
    .add_yaxis("视频数", df['评论数区间'].value_counts().reindex(labels2).fillna(0).astype(int).tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="评论数分布"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="评论数区间"),
        yaxis_opts=AxisOpts(name="视频数量")
    )
)

# 播放/评论/弹幕的箱型图
box_data = df[['播放数', '评论数', '弹幕数']].values.T.tolist()
boxplot = Boxplot()
boxplot.add_xaxis(["播放数", "评论数", "弹幕数"])
boxplot.add_yaxis("分布", boxplot.prepare_data(box_data))
boxplot.set_global_opts(title_opts=TitleOpts(title="播放数/评论数/弹幕数 箱型图"))

# 发布时间分布（Line图）
df['日期'] = df['发布时间_dt'].dt.date
date_stats = df.groupby('日期').size().sort_index()
pub_line = (
    Line()
    .add_xaxis(date_stats.index.astype(str).tolist())
    .add_yaxis("视频发布数量", date_stats.tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="视频发布时间趋势"),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="日期"),
        yaxis_opts=AxisOpts(name="发布数量"),
        datazoom_opts={"type": "slider"}
    )
)

# 汇总页面
page = Page(layout=Page.SimplePageLayout)
page.add(view_count_bar, comment_bar, boxplot, pub_line)
page.render("step1_基础分析.html")
```

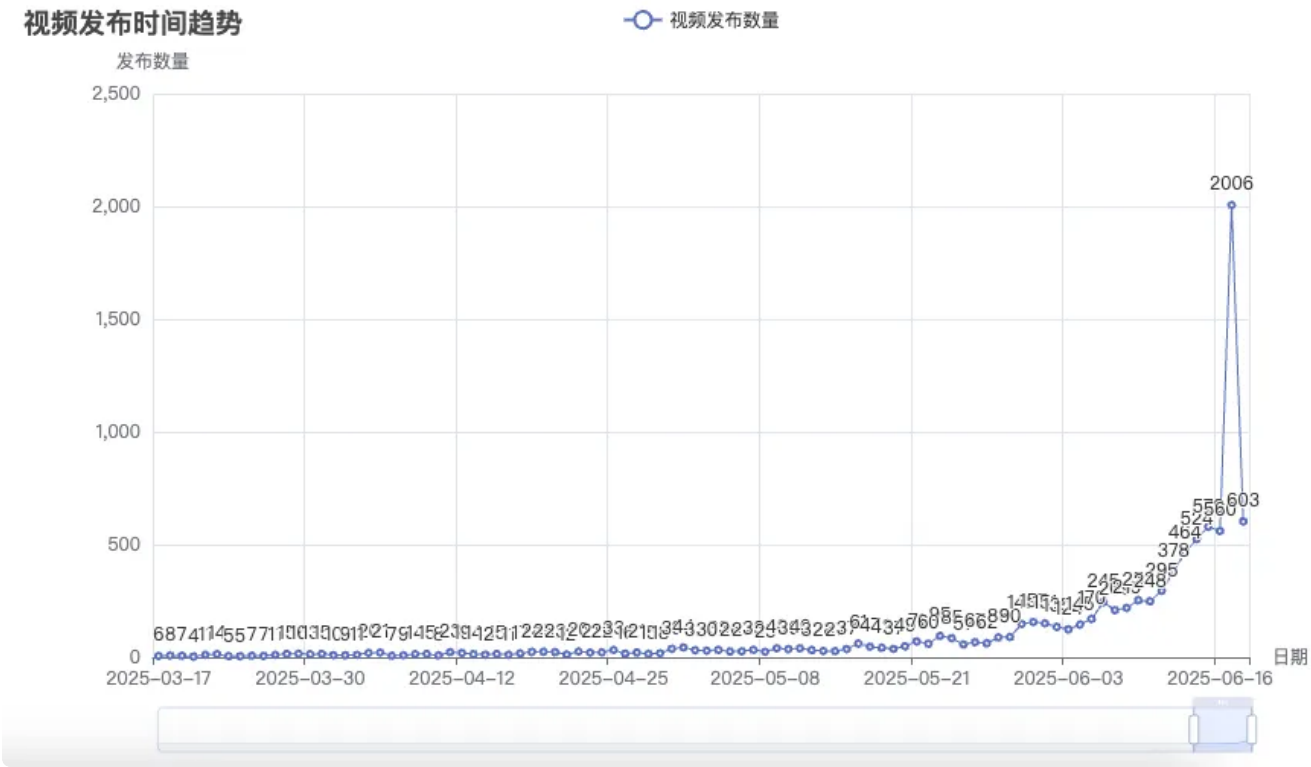
播放数分布


视频数
视频数量
5,000
4,000
3,000
2,000
1,000
0
2441
1405
2660
4431
2212
177
<1K
1K–10K
10K–100K
100K–1M
1M–10M
10M+
播放数区间

评论数分布


视频数
视频数量
5,000
4,000
3,000
2,000
1,000
0
1978
2508
4475
2111
385
206
<10
10–100
100–1K
1K–5K
5K–10K
10K+
评论数区间

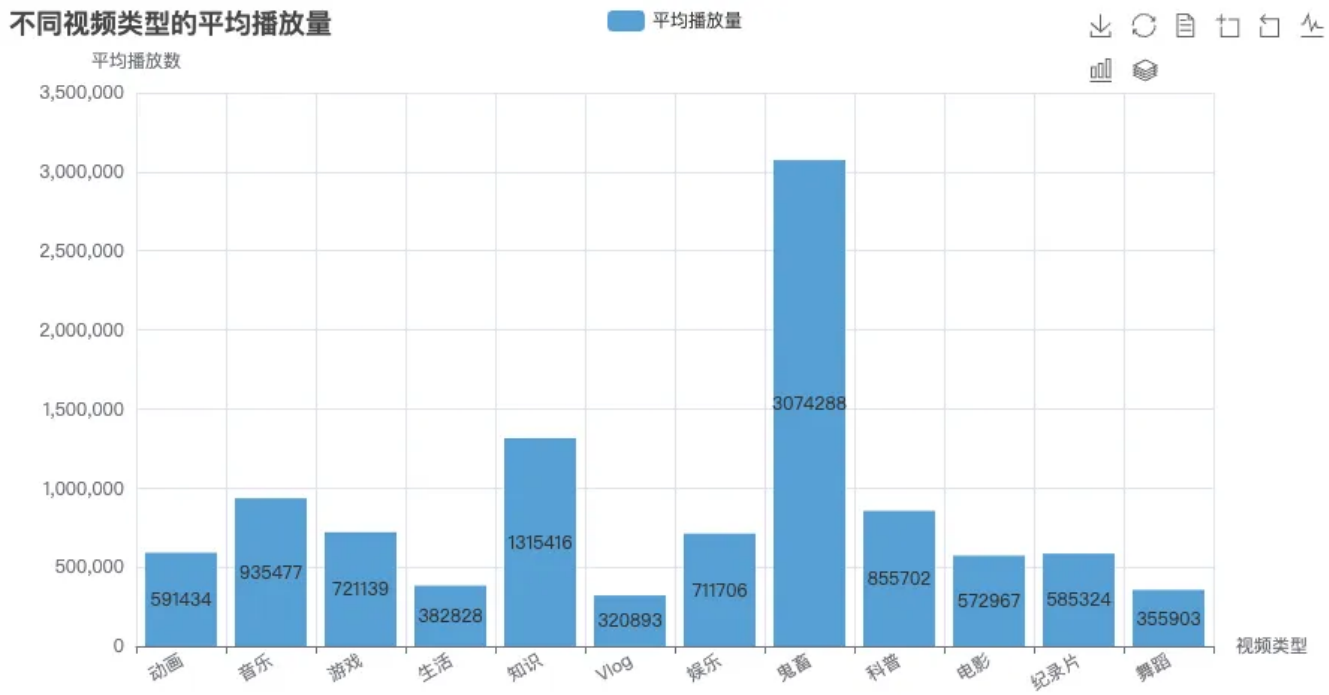

播放数/评论数/弹幕数 箱型图
分布
180,000,000
150,000,000
120,000,000
90,000,000
60,000,000
30,000,000
0
播放数
评论数
弹幕数
分布
评论数
min 0
Q1 4
median 116
Q3 683
max 327,312


视频发布时间趋势
视频发布数量
发布数量
2,500
2,000
1,500
1,000
500
0
2006
603
日期
2025-03-17
2025-03-30
2025-04-12
2025-04-25
2025-05-08
2025-05-21
2025-06-03
2025-06-16

A2 热门视频类型

HotVideo.py Python

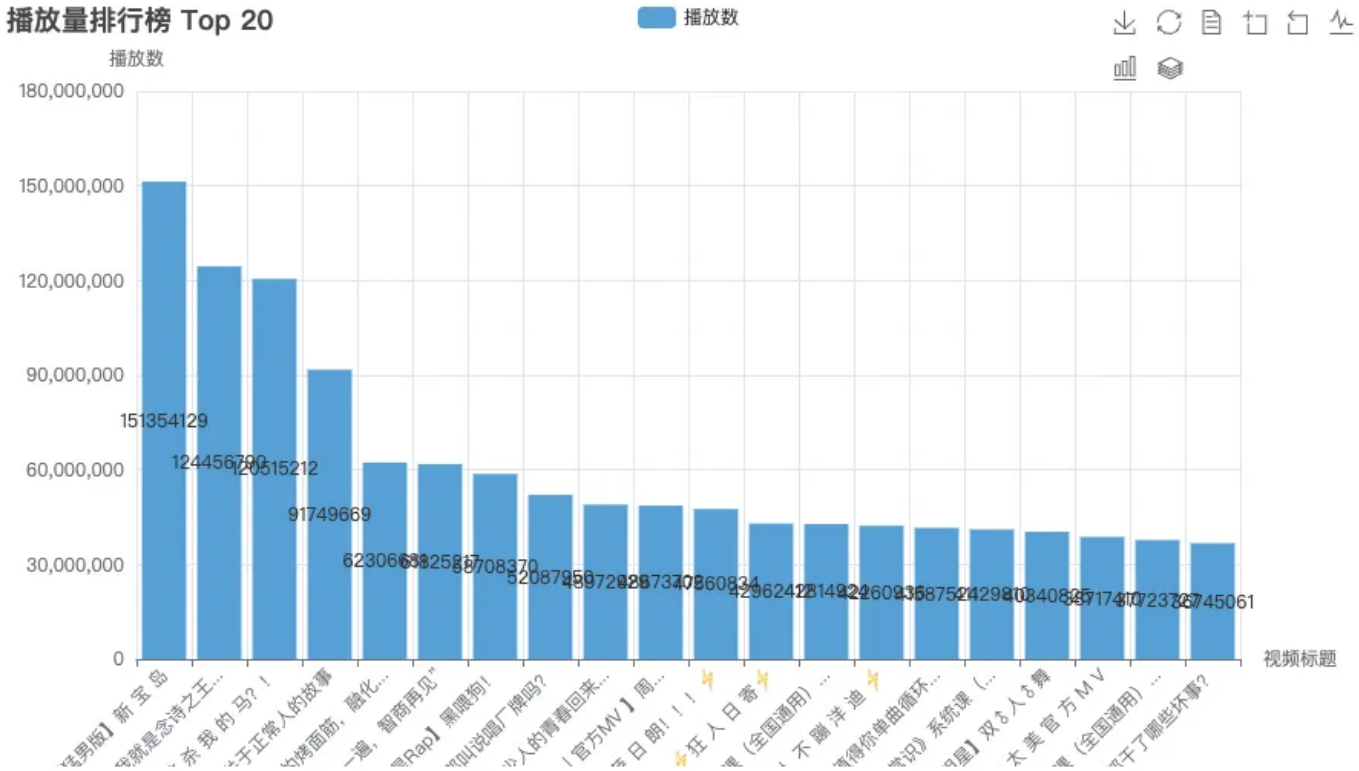
```python
import pandas as pd
import numpy as np
from pyecharts.charts import Bar
from pyecharts.options import TitleOpts, ToolboxOpts, TooltipOpts, AxisOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_data_with_category.csv")
df = df.drop_duplicates(subset='BV号')
df = df.replace([np.inf, -np.inf], pd.NA)
df = df.dropna(subset=['播放数', 'category'])

# 原始顺序手动指定（避免打乱）
type_order = ["动画", "音乐", "游戏", "生活", "知识", "Vlog", "娱乐", "鬼畜", "科普", "电影", "纪录片", "舞蹈"]

# 分区平均播放量
avg_views = df.groupby("category")["播放数"].mean().reindex(type_order)

# 清除没出现的分区
avg_views = avg_views.dropna()

# 绘图
bar = (
    Bar(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(avg_views.index.tolist())
    .add_yaxis("平均播放量", avg_views.round(0).astype(int).tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="不同视频类型的平均播放量"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="视频类型", axislabel_opts={"rotate": 30}),
        yaxis_opts=AxisOpts(name="平均播放数")
    )
)

bar.render("step2_视频类型热度分析.html")
print("step2_视频类型热度分析.html 已生成")
```


不同视频类型的平均播放量
平均播放量
平均播放数
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
591434
935477
721139
382828
1315416
320893
711706
3074288
855702
572967
585324
355903
动画
音乐
游戏
生活
知识
Vlog
娱乐
鬼畜
科普
电影
纪录片
舞蹈
视频类型

A3 播放量排行榜

PlayList.py Python

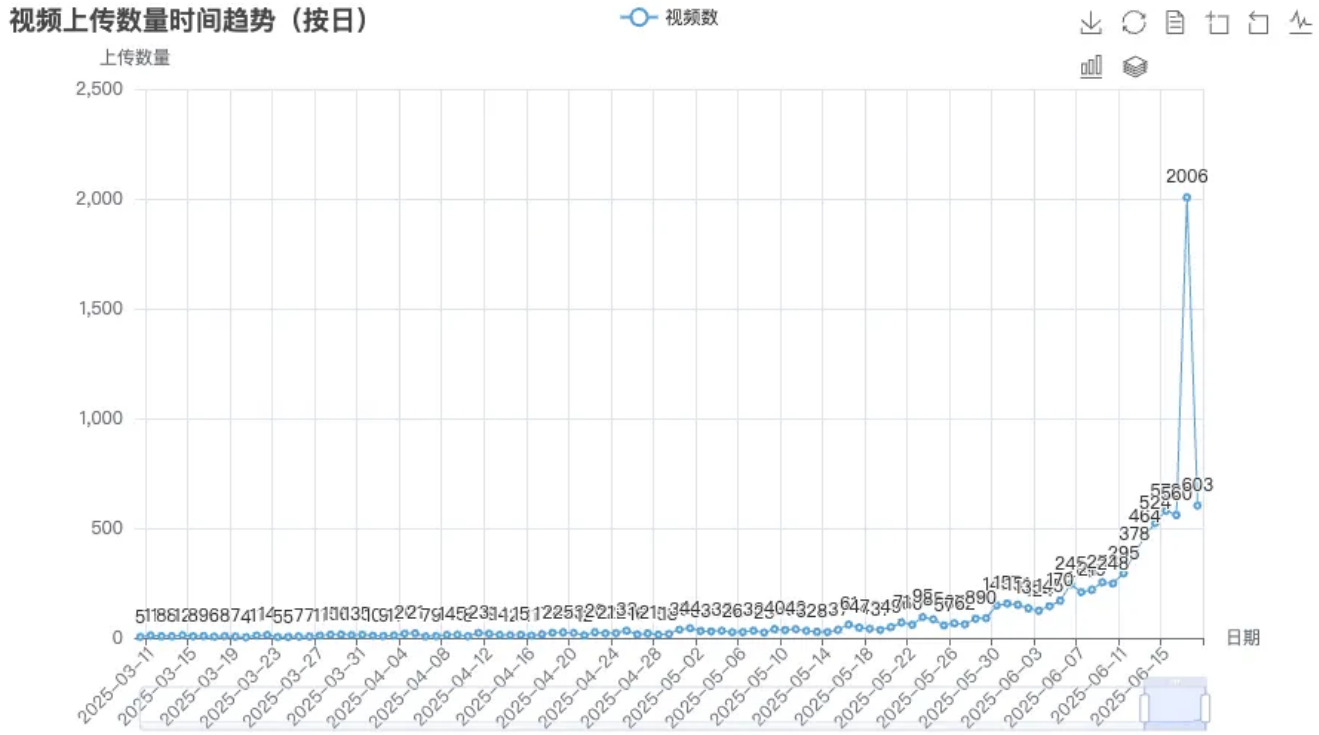
```python
import pandas as pd
from pyecharts.charts import Bar
from pyecharts.options import TitleOpts, ToolboxOpts, TooltipOpts, AxisOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.replace([float('inf'), -float('inf')], pd.NA)
df = df.dropna(subset=['播放数', '标题'])

# 获取播放量Top 20
top_videos = df.sort_values(by="播放数", ascending=False).head(20)

# 横坐标：视频标题（截断避免太长）
titles = [title if len(title) < 20 else title[:17] + '...' for title in top_videos['标题']]
views = top_videos['播放数'].astype(int).tolist()

# 构建柱状图
bar = (
    Bar(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(titles)
    .add_yaxis("播放数", views)
    .set_global_opts(
        title_opts=TitleOpts(title="播放量排行榜 Top 20"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(axislabel_opts={"rotate": 45}, name="视频标题"),
        yaxis_opts=AxisOpts(name="播放数")
    )
)

bar.render("step3_播放排行榜.html")
print("step3_播放排行榜.html 已生成")
```


播放量排行榜 Top 20
播放数
播放数
180,000,000
150,000,000
120,000,000
90,000,000
60,000,000
30,000,000
0
151354129
124456799
120515212
91749669
视频标题

A4 上传时间趋势

TimeAnalysis.py Python

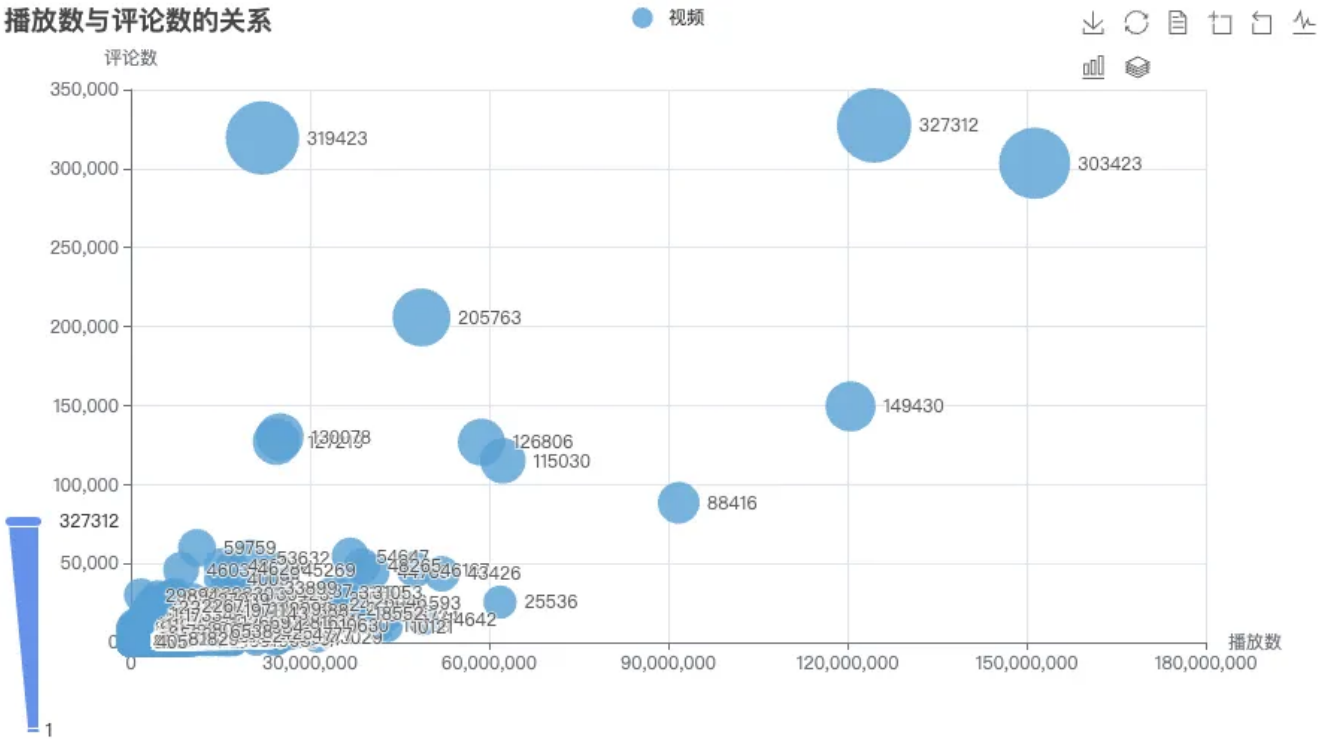
```python
import pandas as pd
from pyecharts.charts import Line
from pyecharts.options import TitleOpts, ToolboxOpts, TooltipOpts, AxisOpts, DataZoomOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.dropna(subset=['发布时间'])

# 时间戳转换为日期
df['发布时间_dt'] = pd.to_datetime(df['发布时间'], unit='s', errors='coerce')
df = df.dropna(subset=['发布时间_dt'])

# 按天统计视频数量
df['日期'] = df['发布时间_dt'].dt.date
date_count = df.groupby('日期').size().sort_index()

# 构造折线图
line = (
    Line(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(date_count.index.astype(str).tolist())
    .add_yaxis("视频数", date_count.tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="视频上传数量时间趋势（按日）"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="日期", axislabel_opts={"rotate": 45}),
        yaxis_opts=AxisOpts(name="上传数量"),
        datazoom_opts=[DataZoomOpts(type_="slider")]
    )
)

line.render("step4_时间趋势.html")
print("step4_时间趋势.html 已生成")
```



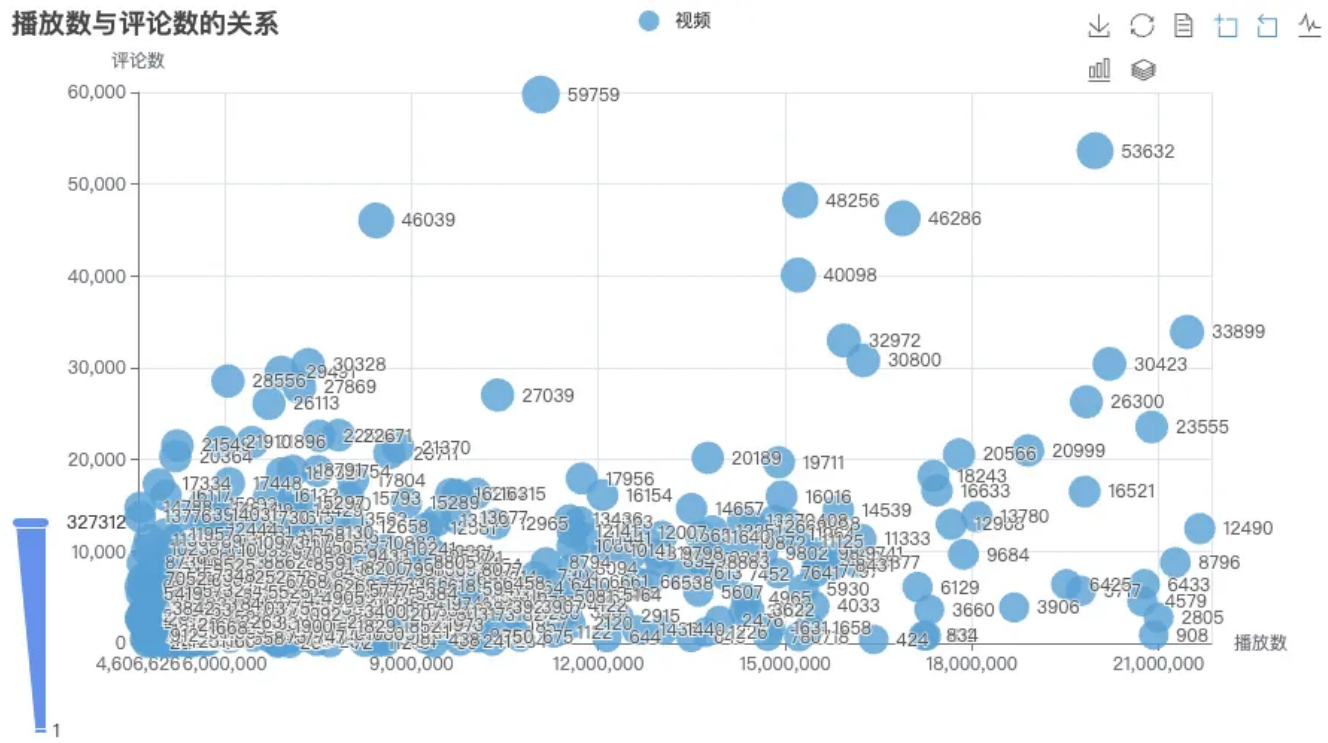
A5 播放量和评论量的关系

PlaywithComment.py Python

```python
import pandas as pd
from pyecharts.charts import Scatter
from pyecharts.options import TitleOpts, TooltipOpts, ToolboxOpts, VisualMapOpts, AxisOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.dropna(subset=['播放数', '评论数'])

# 筛掉异常值（播放数过大或评论数为NaN）
df = df[(df['播放数'] > 0) & (df['评论数'] > 0)]

# 准备数据
data = list(zip(df['播放数'].astype(int), df['评论数'].astype(int)))

# 散点图
scatter = (
    Scatter(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis([x for x, y in data])
    .add_yaxis("视频", [y for x, y in data])
    .set_global_opts(
        title_opts=TitleOpts(title="播放数与评论数的关系"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="item", formatter="播放: {@[0]}<br/>评论: {@[1]}"),
        xaxis_opts=AxisOpts(name="播放数", type_="value"),
        yaxis_opts=AxisOpts(name="评论数", type_="value"),
        visualmap_opts=VisualMapOpts(type_="size", max_=max(df['评论数']), min_=min(df['评论数']), dimension=1)
    )
)

scatter.render("step5_播放与评论关系图.html")
print("step5_播放与评论关系图.html 已生成")
```



上图进行区域放大后如下：



A6 关键词词云

WordClouds.py Python

```python
import pandas as pd
import jieba
from collections import Counter
from pyecharts.charts import WordCloud
from pyecharts.options import TitleOpts, TooltipOpts

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.dropna(subset=['标题'])

# 提取所有标题
titles = df['标题'].astype(str).tolist()

# 中文分词
all_words = []
for title in titles:
    words = jieba.lcut(title)
    all_words.extend(words)

# 常见停用词列表（可以根据需要扩充）
stopwords = set(['的', '了', '是', '我们', '你', '我', '他', '她', '也',
'都', '很', '就', '在', '和', '一个', '上', '下', '啊'])

# 统计词频
words_filtered = [w for w in all_words if w.strip() and w not in stopword
s and len(w) > 1]
counter = Counter(words_filtered)
top_words = counter.most_common(100)

# 构建词云
wc = (
    WordCloud()
    .add(series_name="关键词", data_pair=top_words, word_size_range=[15, 10
0])
    .set_global_opts(
        title_opts=TitleOpts(title="视频标题词云"),
        tooltip_opts=TooltipOpts(is_show=True)
    )
)

wc.render("step6_视频标题词云.html")
print("step6_视频标题词云.html 已生成")
```

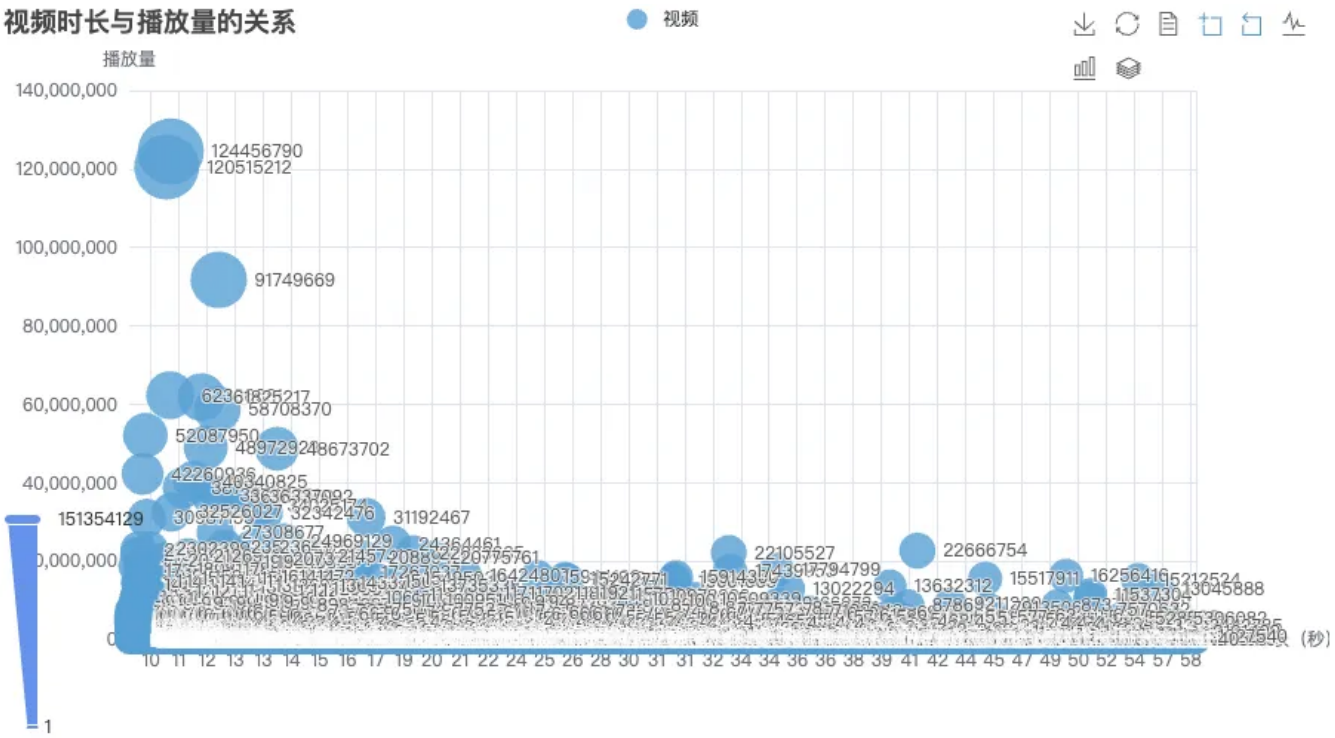
视频标题词云

A7 视频时长和播放量的关系

TimewithPlay.py Python

```python
import pandas as pd
from pyecharts.charts import Scatter
from pyecharts.options import TitleOpts, TooltipOpts, ToolboxOpts, VisualMapOpts, AxisOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.dropna(subset=['播放数', '时长'])

# 时长单位换算（有些平台返回为 xx:yy 格式，要先转化）
def parse_duration(s):
    try:
        if isinstance(s, str) and ':' in s:
            parts = list(map(int, s.split(':')))
            return parts[0] * 60 + parts[1] if len(parts) == 2 else parts[0]
        else:
            return float(s)
    except:
        return None

df['时长_sec'] = df['时长'].apply(parse_duration)
df = df.dropna(subset=['时长_sec'])

# 筛选极端值（例如播放数 > 0 且视频时长合理）
df = df[(df['播放数'] > 0) & (df['时长_sec'] > 0) & (df['时长_sec'] < 7200)]  # 小于2小时
df = df.sort_values(by='时长_sec')

# 构建数据
data = list(zip(df['时长_sec'].astype(int), df['播放数'].astype(int)))

# 生成散点图
scatter = (
    Scatter(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis([x for x, y in data])
    .add_yaxis("视频", [y for x, y in data])
    .set_global_opts(
        title_opts=TitleOpts(title="视频时长与播放量的关系"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="item", formatter="时长: {@[0]}秒<br/>播放: {@[1]}"),
```

```
        xaxis_opts=AxisOpts(name="视频时长（秒）"),
        yaxis_opts=AxisOpts(name="播放量"),
        visualmap_opts=VisualMapOpts(type_="size", dimension=1,
                                     max_=max(df['播放数']),
                                     min_=min(df['播放数']))
    )
)

scatter.render("step7_时长与播放关系.html")
print("step7_时长与播放关系.html 已生成")
```



上图使用了散点图，因为没有对视频类型进行分类，所以不够直观

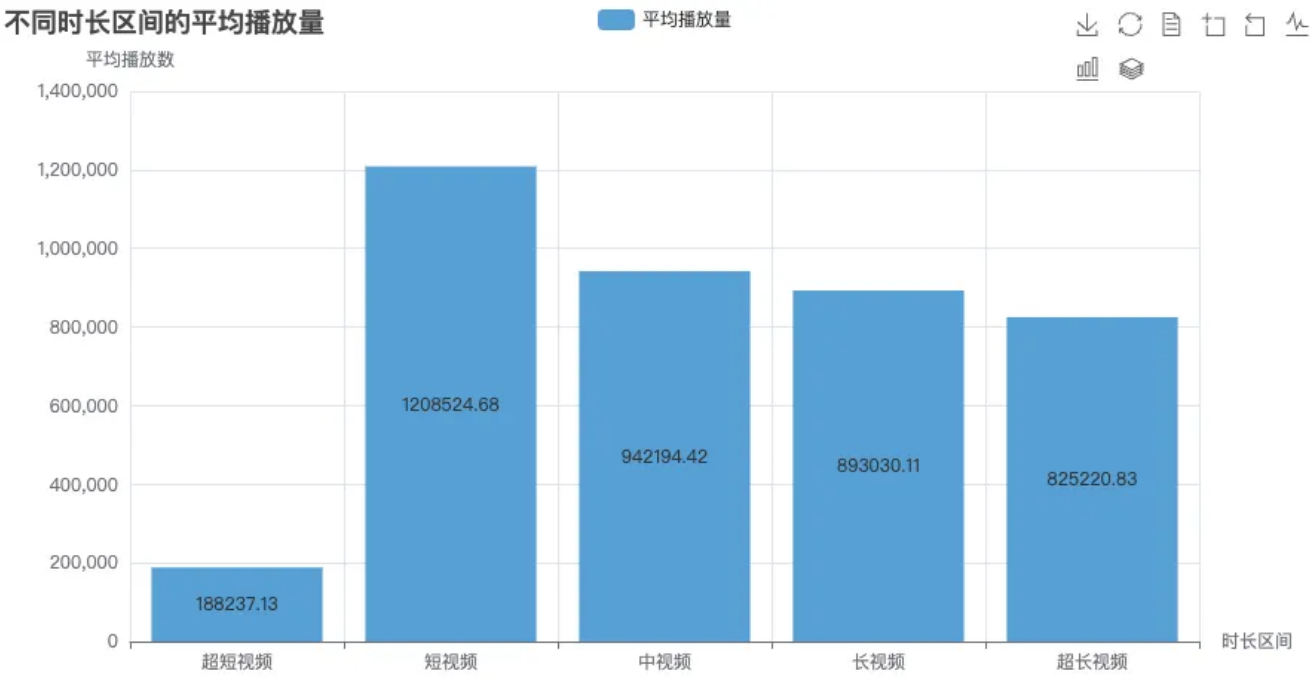
改进分类方法如下：

视频时长	分类标准
小于1分钟	超短视频
1～5分钟	短视频
5～15分钟	中视频
15～30分钟	长视频
大于30分钟	超长视频

Time2withPlay.py Python

```python
import pandas as pd
from pyecharts.charts import Bar
from pyecharts.options import TitleOpts, ToolboxOpts, TooltipOpts, AxisOpts
from pyecharts.globals import ThemeType

# 读取数据
df = pd.read_csv("combined_all.csv")
df = df.drop_duplicates(subset='BV号')
df = df.dropna(subset=['播放数', '时长'])

# 处理时长
def parse_duration(s):
    try:
        if isinstance(s, str) and ':' in s:
            parts = list(map(int, s.split(':')))
            return parts[0] * 60 + parts[1] if len(parts) == 2 else parts[0]
        else:
            return float(s)
    except:
        return None

df['时长_sec'] = df['时长'].apply(parse_duration)
df = df.dropna(subset=['时长_sec'])
df = df[(df['播放数'] > 0) & (df['时长_sec'] > 0) & (df['时长_sec'] < 7200)]

# 定义时长区间
def get_duration_range(sec):
    if sec <= 60:
        return "超短视频"
    elif sec <= 300:
        return "短视频"
    elif sec <= 900:
        return "中视频"
    elif sec <= 1800:
        return "长视频"
    else:
        return "超长视频"

df['时长区间'] = df['时长_sec'].apply(get_duration_range)

# 计算每个区间的平均播放数

```

```
avg_play = df.groupby('时长区间')['播放数'].mean().reindex(['超短视频', '短视频', '中视频', '长视频', '超长视频'])

# 构建柱状图
bar = (
    Bar(init_opts={"theme": ThemeType.LIGHT})
    .add_xaxis(avg_play.index.tolist())
    .add_yaxis("平均播放量", avg_play.values.round(2).tolist())
    .set_global_opts(
        title_opts=TitleOpts(title="不同时长区间的平均播放量"),
        toolbox_opts=ToolboxOpts(),
        tooltip_opts=TooltipOpts(trigger="axis"),
        xaxis_opts=AxisOpts(name="时长区间"),
        yaxis_opts=AxisOpts(name="平均播放数")
    )
)

bar.render("step8_时长区间平均播放量.html")
```



6.1 主要发现

1. 视频时长在1～5分钟之间的内容整体播放量最高，说明中视频更易被观众接受。这类视频能够在保证信息完整度的同时，不至于造成用户疲劳，形成较好的观看体验，是UP主常用的内容制作时长范围。
2. 播放量与评论数存在较明显的正相关趋势，反映出视频越受欢迎，用户越倾向于留言互动。这种趋势在部分高播放视频中特别明显，表明评论区在B站生态中仍然承担着交流与反馈的重要

角色。

3. 热门视频标题中关键词集中在“教学”、“翻跳”、“剪辑”、“日常”、“干货”等，显示观众对内容的实用性与创意性具有较高兴趣。尤其是“干货”、“教学”类内容，往往代表视频具有较高的知识密度和学习价值，吸引特定垂直领域用户。

6.2 不足与改进

1. 数据采集过程中，部分接口存在访问限制，尤其是与投币、收藏、分享等用户互动维度相关的数据，未登录用户访问时容易受到权限约束。这使得部分维度的分析未能展开，影响了数据的全面性。
2. 本项目未能覆盖所有分区的内容，尽管对主流类型如动画、娱乐、游戏等有较为充分的数据支撑，但对于纪录片、Vlog等长尾类别的视频，样本数量偏少，造成在类型分析上的偏倚。
3. 数据收集以公开信息为主，缺乏用户画像与行为路径信息，难以建立完整的内容消费链条，未来可通过引入用户层级信息，增强分析的深度与个性化。
4. 当前分析主要基于静态特征，尚未建立完整的动态预测模型，后续可考虑引入机器学习方法，如随机森林或回归分析，探索建立热度预测机制，实现对视频未来表现的预测与干预。
5. 互动数据分析仅限于数量维度，未深入挖掘评论内容本身。未来可引入自然语言处理技术，对评论文本进行情感倾向分析，识别视频所引发的观众态度变化，从而丰富用户行为理解的维度。

七、结论

本项目围绕Bilibili视频平台展开了全面的数据分析流程，从数据采集、清洗处理到可视化展现与结果解读，完整实现了从原始数据到洞察输出的过程。

通过多个角度的可视化分析，发现中等时长的视频具有更高的观众接受度，播放量与评论数之间具有较强相关性，同时视频发布时间集中于晚间高峰时段，反映出UP主普遍倾向于在观众活跃期推送内容。关键词词云展示了当下受欢迎的视频内容主题，为创作者选题提供了趋势指导。

本研究不仅展现了视频内容数据在传播规律理解中的价值，也提供了基于公开平台数据开展多维分析的一种可复制方法。未来如结合平台推荐机制、用户反馈循环和个性化行为轨迹，或将进一步推动对内容生态与用户偏好之间复杂互动机制的建模研究。