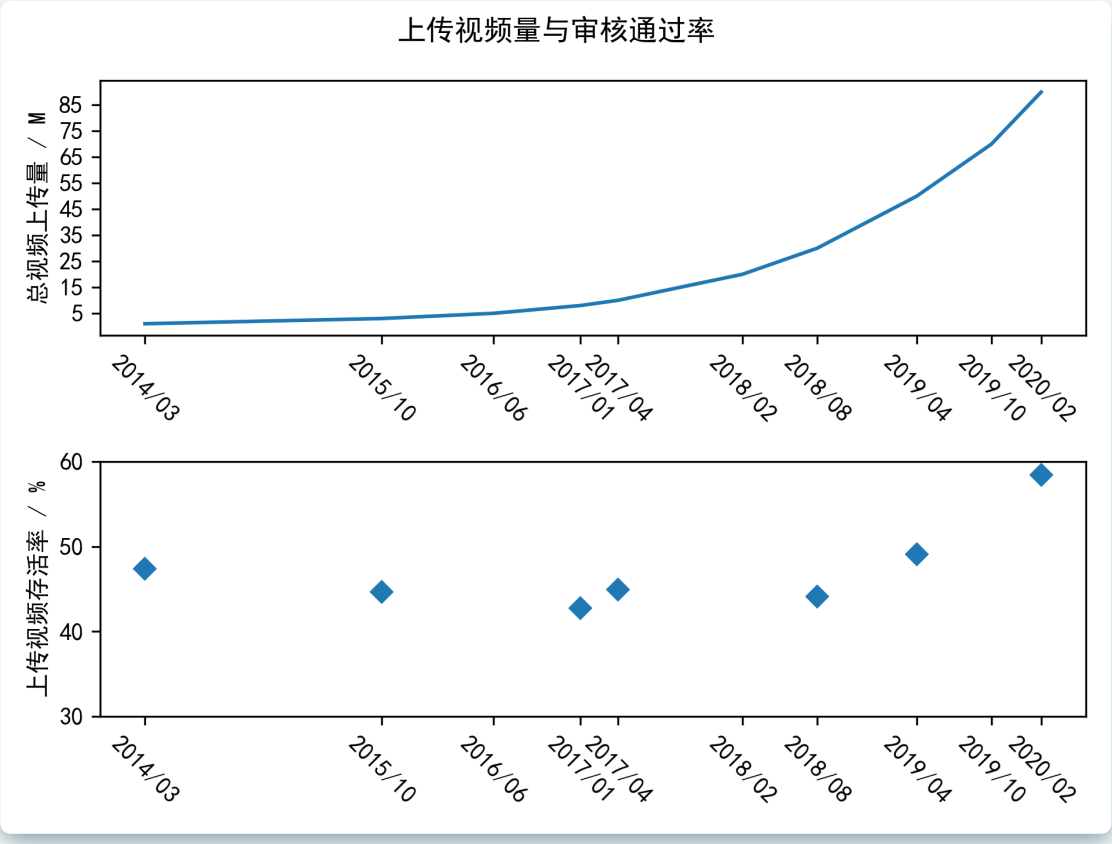
✓ 数据分析

综合数据的选取，数据的有效性，数据内容等等因素，在 Python 相关数据分析工具的帮助下，我们可以得出以下结论。

结论 1

近年来 B 站可播放视频总量增长趋势逐年加快



图表说明

可以看到，近年来总视频上传量成指数级增长趋势；而视频上传后，其通过率在一定范围之内维持相对稳定；因此根据【可播放视频量 = 上传量 * 通过率】，我们得出结论，近年来 B 站的可播放视频总量的增长趋势逐年加快。

结论来源分析

在【数据获取】节的【数据规模与范围】中我们已经提及到，我们采用了分段式的数据获取模式，即在不同的 AV 号区段内尝试获取一定规模的数据。根据我们获取到的数据总数与尝试获取规模数的比值，我们可以在一定程度上推断当时的视频审核通过率（图2）。对于每一个视频【分段】，由于 AV 号是连续的，上传时间也大致相同，我们可以对每一个分段取平均 ID 作为其 ID，平均上传时间作为其上传时间。然后我们描点绘制出折线图（图1），便可以得到视频上传总量随时间变化的大致趋势。

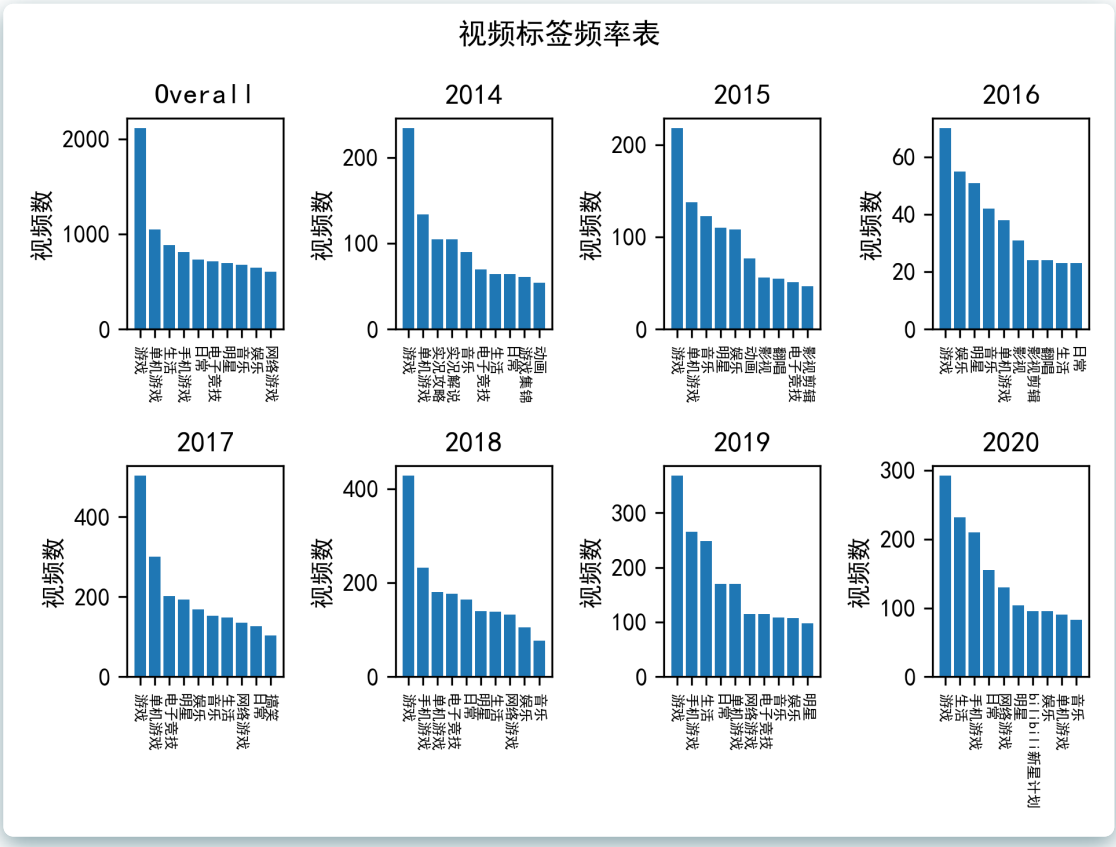
代码实现

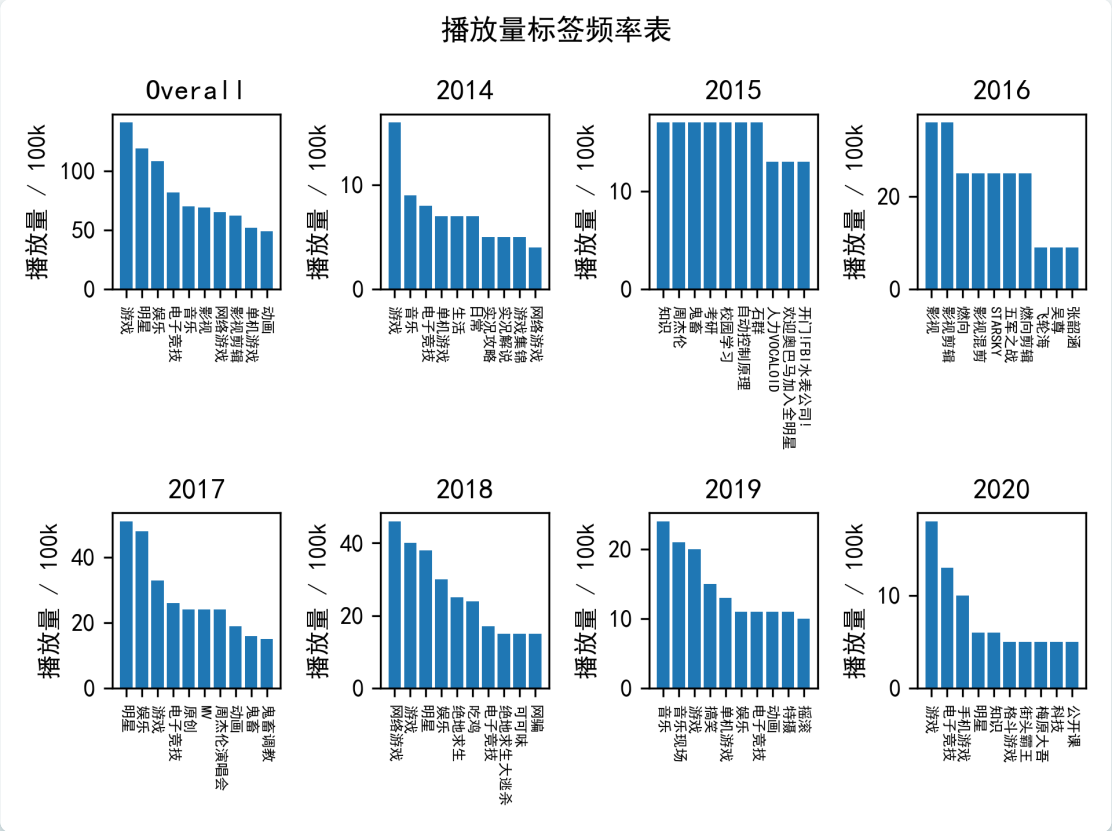
```
def Conclusion1():
    idList = np.array(videos['id'])
    timeList = list(videos['uploadTime'])
    timeList = np.array(list(map(lambda x: x.timestamp(), timeList)))
    idRepresent = idList // 1000000

    # Paint Graphs Preparation
    x = []
    x_label = []
    y = []
    y2 = []

    # Calc average time for each uniqueId
    idUnique = np.unique(idRepresent)
    for tid in idUnique:
        mask = (idRepresent == tid)
        averageTime = np.dot(mask, timeList) // mask.sum()
        x.append((datetime.datetime.fromtimestamp(averageTime) -
                  datetime.datetime.fromtimestamp(1388505600)).days // 30)  # 2014-1-1
00:00:00
        x_label.append(datetime.datetime.fromtimestamp(
            averageTime).strftime('%Y/%m'))
        y2.append(mask.sum())

    y = idUnique
    y2 = np.array(y2) / np.array([1100, 1499, 1499,
                                  1499, 1499, 1499, 1499, 1499, 1499, 1499]) * 100

    # Calc x-id and x-label
    # Start 2014.01 <-> 1, with month as unit
    # print(x)
    # print(y)
    # print(x_label)
    plt.rcParams['savefig.dpi'] = 300
    plt.suptitle('上传视频量与审核通过率')

    # Draw graph 1
    plt.subplot(2, 1, 1)
    plt.plot(x, y)

    # Chinese tags support and minus symbol, reference
https://blog.csdn.net/qq_40563761/article/details/102989770
    plt.rcParams['font.family'] = ['sans-serif']
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.ylabel('总视频上传量 / M')

    plt.xticks(x, x_label, rotation=315)
    plt.yticks(np.arange(5, 95, 10))

    # plt.savefig('1-1.png')
    # plt.close()

    # Draw graph 2
    plt.subplot(2, 1, 2)
    plt.plot(x, y2, 'D')
```

```
    plt.ylabel('上传视频存活率 / %')

    plt.xticks(x, x_label, rotation=315)
    plt.ylim(30, 60)

    # Save Figure
    plt.tight_layout()
    plt.savefig('1.png')
    plt.close()
```

结论 2

游戏持续是 B 站视频总量与播放量的杰出贡献者，而 B 站在多元化的道路上不断迈进





图表说明

我们给出了所有视频标签的频率表（图1）与所有播放量的标签的频率表（图2）。可以看到，游戏始终在各个年份的频率表中处于高位，为 B 站贡献了大量的视频与播放量。而从每播放量的标签频率分布来看，知识、影视、明星等等播放量也不时深居榜首，这说明 B 站在走向多元化的道路上不断前进。

结论来源分析

在爬取视频时，我们已经收集了每个视频的标签。我们将不同视频首先按照年份归类，然后按照每视频（每视频对其下属所有标签的贡献权重为 1）、每播放量（每视频对其下属所有标签的贡献权重为其播放量）两种方式，分别统计不同标签的权重后，进行一次降序排序。然后按照排序后的结果切片出前十名最受欢迎的标签后，形成图像。

代码实现

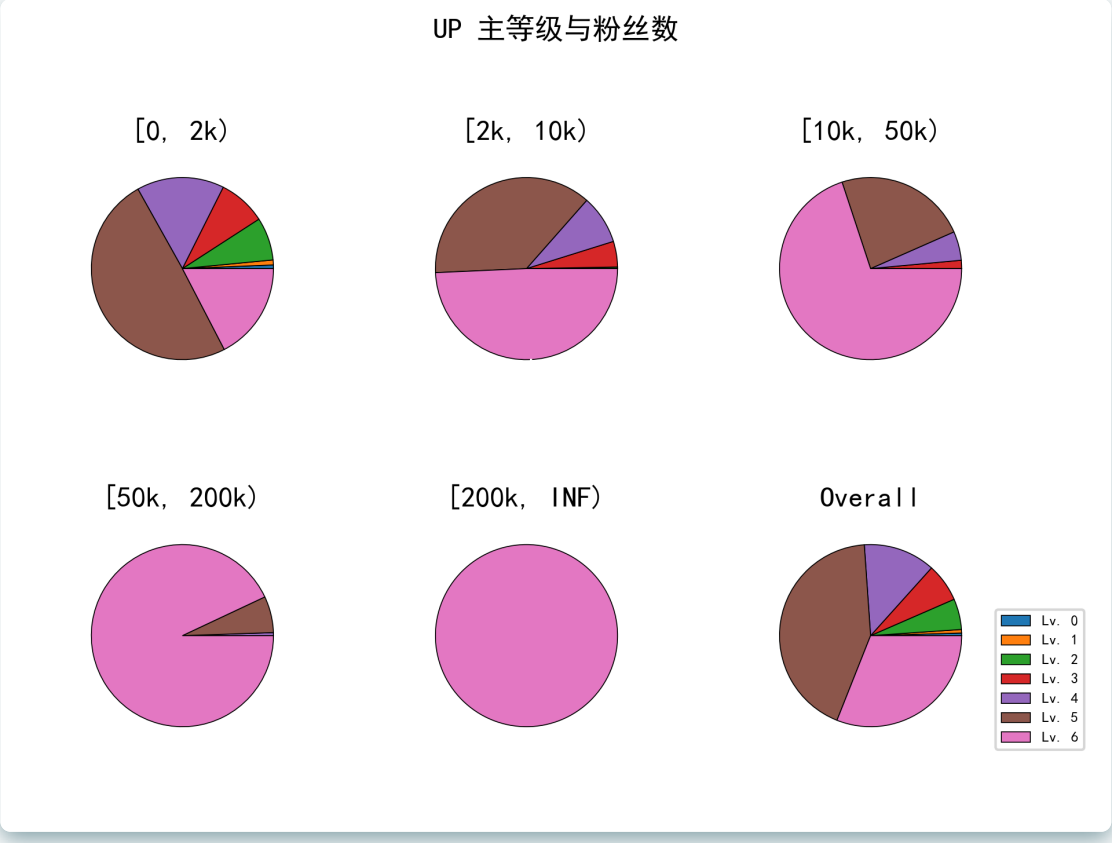
```
def Conclusion2():
    frequencies = {}  # 2014~2020, all
    frequencies['all'] = {}
    frequenciesWithHits = {}  # 2014~2020, all
    frequenciesWithHits['all'] = {}
    # print(frequencies)

    for index, row in videos.iterrows():
        id = (row['id'])
        uploadYear = str(row['uploadTime'].year)
        hits = row['playCount']

        # Find tags related to id
```

```
        relatedTags = tags[tags['video_id'] == id]
        for index, tagRow in relatedTags.iterrows():
            tagName = tagRow['tagName']

            # Push Tagname to frequencies list
            if uploadYear not in frequencies:
                frequencies[uploadYear] = {}
                frequenciesWithHits[uploadYear] = {}
            if tagName not in frequencies[uploadYear]:
                frequencies[uploadYear][tagName] = 0
                frequenciesWithHits[uploadYear][tagName] = 0
            if tagName not in frequenciesWithHits['all']:
                frequenciesWithHits['all'][tagName] = 0
            if tagName not in frequencies['all']:
                frequencies['all'][tagName] = 0

            frequencies[uploadYear][tagName] += 1
            frequencies['all'][tagName] += 1
            frequenciesWithHits[uploadYear][tagName] += hits
            frequenciesWithHits['all'][tagName] += hits

    # Carry out sort in frequencies dict and truncate top 10 tags
    for key in frequencies.keys():
        frequencies[key] = sorted(
            frequencies[key].items(), key=lambda x: x[1], reverse=True)[0:10]
    for key in frequenciesWithHits.keys():
        frequenciesWithHits[key] = sorted(
            frequenciesWithHits[key].items(), key=lambda x: x[1], reverse=True)[0:10]

    # print(frequencies)
    # print(frequenciesWithHits)

    # Draw Graph 1
    plt.rcParams['savefig.dpi'] = 300
    plt.suptitle('视频标签频率表')
    k = 0
    for key in frequencies.keys():
        k += 1
        plt.subplot(2, 4, k)
        if key == "all":
            plt.title('Overall')
        else:
            plt.title(key)
        plt.bar(np.arange(1, 11, 1), list(
            map(lambda x: x[1], frequencies[key])))
        plt.xticks(np.arange(1, 11, 1), list(
            map(lambda x: x[0], frequencies[key])), rotation=270, fontsize=6)
        plt.ylabel('视频数')

    # Save Figure
    plt.tight_layout()
    plt.savefig('2-1.png')
    plt.close()

    # Draw Graph 2
    plt.rcParams['savefig.dpi'] = 300
    plt.suptitle('播放量标签频率表')
    k = 0
    for key in frequenciesWithHits.keys():
        k += 1
        plt.subplot(2, 4, k)
```

```
        if key == "all":
            plt.title('Overall')
        else:
            plt.title(key)
        plt.bar(np.arange(1, 11, 1), list(
            map(lambda x: x[1]//100000, frequenciesWithHits[key])))
        plt.xticks(np.arange(1, 11, 1), list(
            map(lambda x: x[0], frequenciesWithHits[key])), rotation=270, fontsize=6)
        plt.ylabel('播放量 / 100k')

    # Save Figure
    plt.tight_layout()
    plt.savefig('2-2.png')
    plt.close()
```

结论 3

粉丝数越多的 UP 主，等级一般就越高



图表说明

（图1）~（图5）的标题代表 UP 主的粉丝数所处的范围，而对应的饼图则代表在这个区段的 UP 主的等级频数分布。可以看到，随着 UP 主粉丝数的逐渐升高，处于低等级的用户的占比逐渐下降，而处于高等级的用户的占比逐渐增加。也就是说，一个 UP 主的粉丝数越多，ta 的等级一般也就越高。

结论来源分析

在爬取 UP 主信息时，我们已经获取了每个 UP 主的等级。于是我们可以在 UP 主的数据表中进行信息提取。首先，我们根据 UP 主的粉丝数所在范围，对 UP 主进行分段，最终分成了图中所呈现的 5 个区间。然后，我们再统计每个区间中的 UP 主的等级频数，并分别绘制成饼图。

代码实现

```
def Conclusion3():

    # Get author follower rank
    def getRank(n):
        checkpoint = [0, 2000, 10000, 50000, 200000, 1145141919810]
        for i in range(1, 6):
            if (checkpoint[i-1] <= n < checkpoint[i]):
                return i
        return -1

    LevelFrequencies = {}
    LevelFrequencies['all'] = [0, 0, 0, 0, 0, 0, 0]

    for index, row in ups.iterrows():
        lvl = row['level']
        rawFollowerCount = row['followerCount']
        if rawFollowerCount[-1] != '万':
            follower = int(rawFollowerCount)
        else:
            follower = int(float(rawFollowerCount[:-1]) * 10000)
        rank = str(getRank(follower))

        # Push to dict
        if not rank in LevelFrequencies:
            LevelFrequencies[rank] = [0, 0, 0, 0, 0, 0, 0]
        LevelFrequencies[rank][lvl] += 1
        LevelFrequencies['all'][lvl] += 1

    # print(LevelFrequencies)

    # Paint Figure
    plt.rcParams['savefig.dpi'] = 300
    plt.suptitle('UP 主等级与粉丝数')

    i = 0
    for key in sorted(LevelFrequencies.keys()):
        i += 1
        plt.subplot(2, 3, i)

        # Set subfigure title
        if key == 'all':
            plt.title('Overall')
        elif key == '1':
            plt.title('[0, 2k)')
        elif key == '2':
            plt.title('[2k, 10k)')
        elif key == '3':
            plt.title('[10k, 50k)')
        elif key == '4':
            plt.title('[50k, 200k)')
```

```
        elif key == '5':
            plt.title('[200k, INF)')

        labels = ['Lv. ' + str(i) for i in range(0, 7)]
        plt.pie(LevelFrequencies[key], wedgeprops={
                'edgecolor': 'black',
                'linewidth': '0.4'
                }, textprops={'color': 'white'}, labels=labels)
    plt.legend(prop={'size': 6}, bbox_to_anchor=(1.05, 0), loc=3, borderaxespad=0)

    # Save Figure
    plt.tight_layout()
    plt.savefig('3.png')
    plt.close()
```