1.数据来源

kesci (https://www.kesci.com/home/dataset/5e9a4884ebb37f002c60bf25/files) 抖音40天内的数据交互记录

字段名称

- uid：用户id
- user_city：用户所在城市
- item_id：作品id
- author_id：作者id
- item_city：作品城市
- channel：观看到该作品的来源
- finish：是否浏览完作品
- like：是否对作品点赞
- music_id：音乐id
- device：设备id
- time：作品发布时间
- duration_time：作品时长

2.分析目的

抖音是一个面向全年龄的音乐短视频社区平台。目前用户众多，有不少人都成为了网红，聚集了大量粉丝。在此希望能看出一些他们之间的共性，帮助后来人少走一些弯路。并结合实际观察抖音运营存的现状，提出一些建议。

2.1 抖音网红建议

1. 抖音98.52%的流量都会流向算法推荐视频，获得算法推荐是获得更多播放的关键所在。
2. 最重要的始终是题材的选择，初始流量池大更容易获得算法青睐。
3. 除去题材外，投稿的最佳时间日常是在0-5点，有平台活动一定要参与。
4. 视频时长最好在7-10s，其次是0-6s及23s以内，最长也不建议超过40s。
5. 背景音乐最好选择当下最流行的歌曲。

2.2 平台运营建议

1. 抖音活动时有大量机器人存在，需要决定是否清除（疑似机器人的[uid]保存在“robot”列表中）。
2. 站内活动小国初见不错，但在去除机器人后并没有大量实质增长，收益很低，再次举办需要慎重。
3. 平台用户始终稳步增长中，但如果想要大量增长，考虑其他渠道，或者开辟新市场是更好的选择。
4. 排名前20%的视频制作者占据了整个平台80%以上的流量，培养或者挖掘已存在的好视频制作者是未来保持流量的关键。

3.数据导入与清洗

- 空值的检查与处理
- 重复值的检查与处理
- 异常值的检查与处理

- 数据类型的检查与调整

In [1]:

```
#导入分析依赖的第三方包
import pandas as pd
import numpy as np
#加载数据可视化包
import matplotlib.pyplot as plt
import time
import seaborn as sns
#可视化显示在页面 jupyter专属 %内置
%matplotlib inline
#更改设计风格
plt.style.use('ggplot')
plt.rcParams['font.sans-serif'] = ['SimHei']#(显示中文)
plt.rcParams['axes.unicode_minus'] = False#(显示负数)
```

In [2]:

```
#读取文件
data = pd.read_table('final_track2_test.txt')
#补充值字段名称
data.columns = ['uid','user_city','item_id','author_id','item_city','channel','finish','like','music
data.head()
```

Out[2]:

	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	device	
0	3456	-1	920393	36070	-1	1	0	0	-1	6895	530
1	17128	-1	1201256	14657	11	1	1	0	-1	61358	530
2	1368	81	2720842	26944	68	0	0	0	-1	1466	530
3	15692	109	691661	18212	213	0	0	0	11513	540	530
4	43686	148	142122	221	35	0	0	0	-1	338	530

In [3]:

```
#备份原始数据
data.to_csv('douyin_rawdata.csv')
```

In [4]:

```
#检测是否存在空值
data.isnull().sum()
```

Out[4]:

```
uid              0
user_city        0
item_id          0
author_id        0
item_city        0
channel          0
finish           0
like             0
music_id         0
device           0
time             0
duration_time    0
dtype: int64
```

In [5]:

```
#查看是否存在重复值
data.duplicated().sum()
```

Out[5]:

```
4924
```

In [6]:

```
#删除重复值
data = data.drop_duplicates()
```

In [7]:

```
#数据是进行过脱敏的数据，无法观察原有情况，不过可以推断其中的-1是缺失值,转换后直接删除即可。
data[data==-1] = np.nan
data.dropna(inplace=True)
```

In [8]:

```
#本次分析中不会使用到device列，和多余Unnamed: 0列，删除
del data['device']
data.head()
```

Out[8]:

	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	tim
3	15692	109.0	691661	18212	213.0	0	0	0	11513.0	530870357
5	44071	80.0	1243212	34500	68.0	0	0	0	1274.0	530864328
16	10902	202.0	3845855	634066	113.0	0	0	0	762.0	530867863
19	25300	21.0	3929579	214923	330.0	0	0	0	2332.0	530867717
24	3656	138.0	2572269	182680	80.0	0	0	0	238.0	530864267

In [9]:

```
#time列是时间戳，此处修改成正常时间
data.time = data.time.astype('str')
data.time = data['time'].apply(lambda x:x[1:])
data.time = data.time.astype('int64')

#将时间戳转换为普通的日期格式
real_time = []
for i in data['time']:
    timeArray = time.localtime(i)
    otherStyleTime = time.strftime("%Y-%m-%d %H:%M:%S", timeArray)
    real_time.append(otherStyleTime)

data['real_time'] = real_time

#time列无用了，删除
del data['time']

data.head()
```

Out[9]:

	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	duration_ti
3	15692	109.0	691661	18212	213.0	0	0	0	11513.0	
5	44071	80.0	1243212	34500	68.0	0	0	0	1274.0	
16	10902	202.0	3845855	634066	113.0	0	0	0	762.0	
19	25300	21.0	3929579	214923	330.0	0	0	0	2332.0	
24	3656	138.0	2572269	182680	80.0	0	0	0	238.0	

In [10]:

```
#数据格式概览
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1737313 entries, 3 to 5886699
Data columns (total 11 columns):
 #   Column         Dtype
---  ------         -----
 0   uid            int64
 1   user_city      float64
 2   item_id        int64
 3   author_id      int64
 4   item_city      float64
 5   channel        int64
 6   finish         int64
 7   like           int64
 8   music_id       float64
 9   duration_time  int64
 10  real_time      object
dtypes: float64(3), int64(7), object(1)
memory usage: 159.1+ MB
```

In [11]:

```
#修改数据格式

for x in data.columns.tolist()[:-5]:
    data[x] = data[x].astype('str')

data['real_time'] = pd.to_datetime(data['real_time'])
data['music_id'] = data['music_id'].astype('str')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1737313 entries, 3 to 5886699
Data columns (total 11 columns):
 #   Column         Dtype
---  ------         -----
 0   uid            object
 1   user_city      object
 2   item_id        object
 3   author_id      object
 4   item_city      object
 5   channel        object
 6   finish         int64
 7   like           int64
 8   music_id       object
 9   duration_time  int64
 10  real_time      datetime64[ns]
dtypes: datetime64[ns](1), int64(3), object(7)
memory usage: 159.1+ MB
```

In [12]:

```
#为数据添加H：小时，和date：日期列
data['H'] = data.real_time.dt.hour
data['date']=data.real_time.dt.date
data.date=data.date.astype('str')
data.head()
```

Out[12]:

	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	duration_ti
3	15692	109.0	691661	18212	213.0	0	0	0	11513.0	
5	44071	80.0	1243212	34500	68.0	0	0	0	1274.0	
16	10902	202.0	3845855	634066	113.0	0	0	0	762.0	
19	25300	21.0	3929579	214923	330.0	0	0	0	2332.0	
24	3656	138.0	2572269	182680	80.0	0	0	0	238.0	

In [13]:

```
data.groupby('date').count()
```

Out[13]:

date	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	du
2067-02-21	1	1	1	1	1	1	1	1	1	
2067-09-21	112	112	112	112	112	112	112	112	112	
2067-09-22	1461	1461	1461	1461	1461	1461	1461	1461	1461	
2067-09-23	2408	2408	2408	2408	2408	2408	2408	2408	2408	
2067-09-24	4122	4122	4122	4122	4122	4122	4122	4122	4122	
2067-09-25	6108	6108	6108	6108	6108	6108	6108	6108	6108	
2067-09-26	7546	7546	7546	7546	7546	7546	7546	7546	7546	
2067-09-27	9492	9492	9492	9492	9492	9492	9492	9492	9492	
2067-09-28	10546	10546	10546	10546	10546	10546	10546	10546	10546	
2067-09-29	12996	12996	12996	12996	12996	12996	12996	12996	12996	
2067-09-30	15235	15235	15235	15235	15235	15235	15235	15235	15235	
2067-10-01	16420	16420	16420	16420	16420	16420	16420	16420	16420	
2067-10-02	16095	16095	16095	16095	16095	16095	16095	16095	16095	
2067-10-03	19059	19059	19059	19059	19059	19059	19059	19059	19059	
2067-10-04	19480	19480	19480	19480	19480	19480	19480	19480	19480	
2067-10-05	20249	20249	20249	20249	20249	20249	20249	20249	20249	
2067-10-06	21460	21460	21460	21460	21460	21460	21460	21460	21460	
2067-10-07	21724	21724	21724	21724	21724	21724	21724	21724	21724	
2067-10-08	24391	24391	24391	24391	24391	24391	24391	24391	24391	
2067-10-09	23004	23004	23004	23004	23004	23004	23004	23004	23004	
2067-10-10	23780	23780	23780	23780	23780	23780	23780	23780	23780	

date	uid	user_city	item_id	author_id	item_city	channel	finish	like	music_id	du
2067-10-11	24341	24341	24341	24341	24341	24341	24341	24341	24341	
2067-10-12	26871	26871	26871	26871	26871	26871	26871	26871	26871	
2067-10-13	29426	29426	29426	29426	29426	29426	29426	29426	29426	
2067-10-14	29353	29353	29353	29353	29353	29353	29353	29353	29353	
2067-10-15	32353	32353	32353	32353	32353	32353	32353	32353	32353	
2067-10-16	32803	32803	32803	32803	32803	32803	32803	32803	32803	
2067-10-17	36447	36447	36447	36447	36447	36447	36447	36447	36447	
2067-10-18	37653	37653	37653	37653	37653	37653	37653	37653	37653	
2067-10-19	47105	47105	47105	47105	47105	47105	47105	47105	47105	
2067-10-20	55145	55145	55145	55145	55145	55145	55145	55145	55145	
2067-10-21	101828	101828	101828	101828	101828	101828	101828	101828	101828	
2067-10-22	122408	122408	122408	122408	122408	122408	122408	122408	122408	
2067-10-23	141415	141415	141415	141415	141415	141415	141415	141415	141415	
2067-10-24	131095	131095	131095	131095	131095	131095	131095	131095	131095	
2067-10-25	123803	123803	123803	123803	123803	123803	123803	123803	123803	
2067-10-26	125541	125541	125541	125541	125541	125541	125541	125541	125541	
2067-10-27	125080	125080	125080	125080	125080	125080	125080	125080	125080	
2067-10-28	107954	107954	107954	107954	107954	107954	107954	107954	107954	
2067-10-29	98756	98756	98756	98756	98756	98756	98756	98756	98756	
2067-10-30	32247	32247	32247	32247	32247	32247	32247	32247	32247	

In [14]:

```
#清除数据不完整的时间
data.drop(data[data.date=='2067-02-21'].index,inplace=True)
data.drop(data[data.date=='2067-10-30'].index,inplace=True)
```

In [15]:

```
#保存清洗后的文件
data.to_csv('douyin_cleaned.csv')
```

4.数据分析

4.1 抖音网红视频数据分析

- 抖音播放量来源分布
- 视频时长与点赞完播率之间的关系
- 作品发布时间与点赞完播率之间的关系

4.2背景音乐与点赞完播率之间的关系

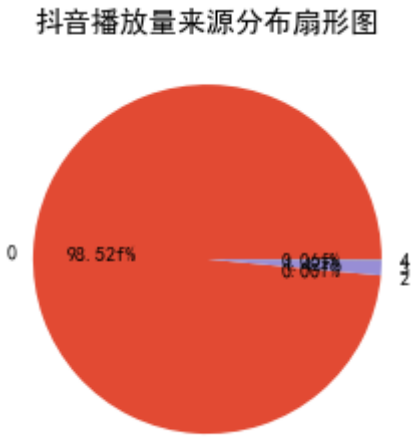
- 热门的音乐ID
- 热门的歌曲点赞率与完播率
- 热门的歌曲点赞率与完播率随时间变化的曲线

In [16]:

```
channel = data.groupby('channel').count()['uid']
```

In [17]:

```
#抖音播放量来源分布
labels = channel.index
plt.pie(channel,labels = labels,autopct='%0.2ff%%')
plt.title("抖音播放量来源分布扇形图")
plt.show()
```



结论1： 虽然没有明确说明，但作为算法驱动的短视频平台，显然可知“0”是算法推荐的视频。那么在抖音获得播

放量的关键就是获得算法推荐进入更大的流量池。

In [18]:

```
#查看时长与完播率和点赞率之间的关系
time_finish = data.groupby('duration_time').mean()
time_finish
```

Out[18]:

duration_time	finish	like	H
0	0.411765	0.000000	11.588235
1	0.388889	0.011111	14.555556
2	0.404453	0.009250	10.753351
3	0.407447	0.010343	10.404308
4	0.398107	0.010145	11.841017
...	...	...	...
119	0.000000	0.000000	5.000000
187	0.000000	0.000000	17.000000
555	0.000000	0.000000	0.000000
620	0.000000	0.000000	22.000000
640	0.500000	0.000000	8.000000

72 rows × 3 columns

In [19]:

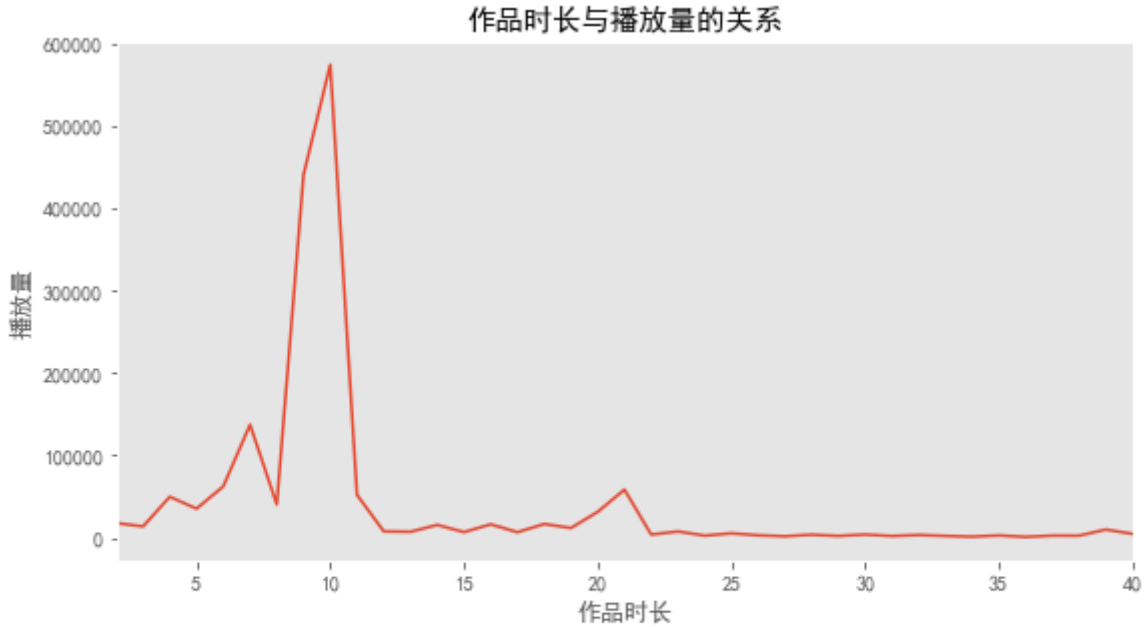
```
#查看作品时长与点赞和完播率的关系
duration_time_f_l = time_finish[data.groupby('duration_time').count()>100]
duration_time_f_l.dropna(inplace=True)
del duration_time_f_l['H']
duration_nums = data.groupby('duration_time').nunique()['item_id']
duration_time_f_l.head()
```

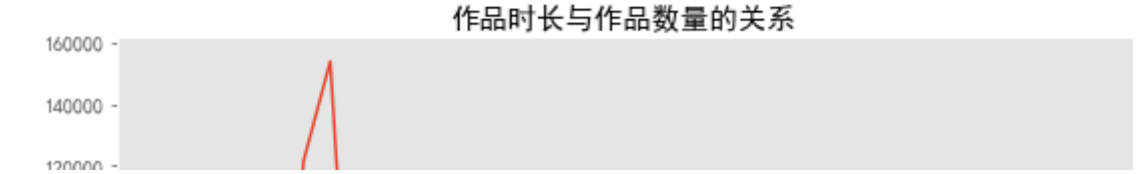
Out[19]:

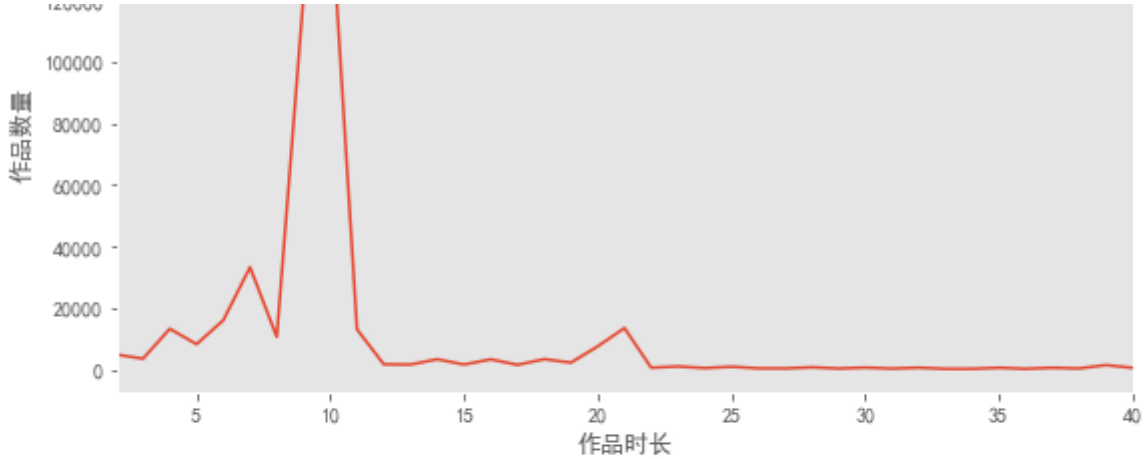
duration_time	finish	like
2	0.404453	0.009250
3	0.407447	0.010343
4	0.398107	0.010145
5	0.398092	0.010326
6	0.406934	0.009191

In [20]:

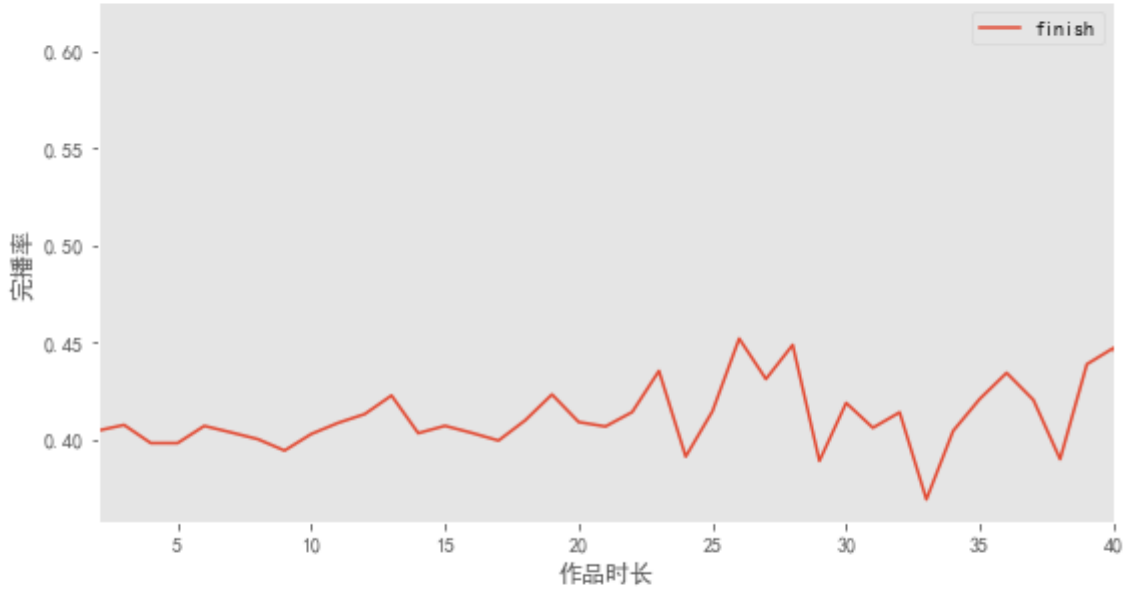
```
fig = plt.figure(figsize=(20,22))

ax1 = fig.add_subplot(4,2,1)
data.groupby('duration_time').count()['uid'].plot(ax=ax1)
plt.xlim(2,40)
plt.xlabel('作品时长')
plt.ylabel('播放量')
plt.title("作品时长与播放量的关系")
plt.grid(False)

ax2 = fig.add_subplot(4,2,3)
duration_nums.plot(ax=ax2)
plt.xlim(2,40)
plt.xlabel('作品时长')
plt.ylabel('作品数量')
plt.title("作品时长与作品数量的关系")
plt.grid(False)

ax3 = fig.add_subplot(4,2,5)
duration_time_f_l.plot(ax=ax3,y='finish')
plt.xlim(2,40)
plt.xlabel('作品时长')
plt.ylabel('完播率')
plt.title("作品时长与完播率的关系")
plt.grid(False)


ax4 = fig.add_subplot(4,2,7)
duration_time_f_l.plot(ax=ax4,y='like')
plt.xlim(2,40)
plt.xlabel('作品时长')
plt.ylabel('点赞率')
plt.title("作品时长与点赞率的关系")
plt.grid(False)
```



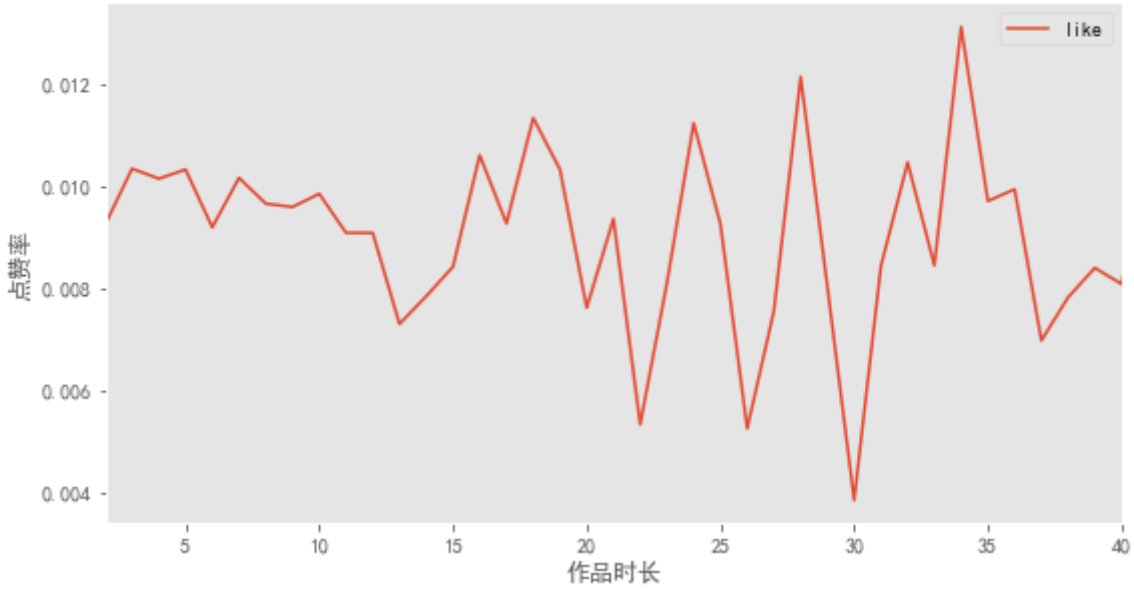




作品时长与完播率的关系



作品时长与点赞率的关系



观察结果：
作品绝大多数分布在7-10s中，总体来说在0s-22s之间都有一定数量的投稿，22s以上的就很少了。
播放量的分布基本与作品数量相同。
完播率在2s-23s内总体在40%以上，23s以后开始在37%-45%之间剧烈波动，
点赞率在2s-12s内基本维持在1%之内，在12s-20s之间会在0.7%-1.1%之间波动，在20s以后数据变化的波动完全没有规律。

结论2： 视频时长最好在7-10s，其次是0-6s及23s以内，最长也不建议超过40s（在580万条记录中没有一条超过

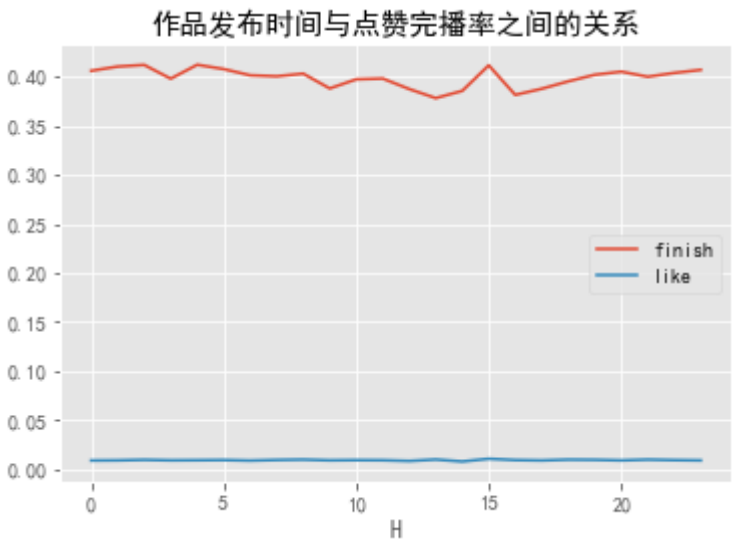
50s视频播放量超过100)

In [21]:

```
#作品发布时间与点赞完播率之间的关系
H_f_l = data.groupby('H').mean()[['finish','like']]
```

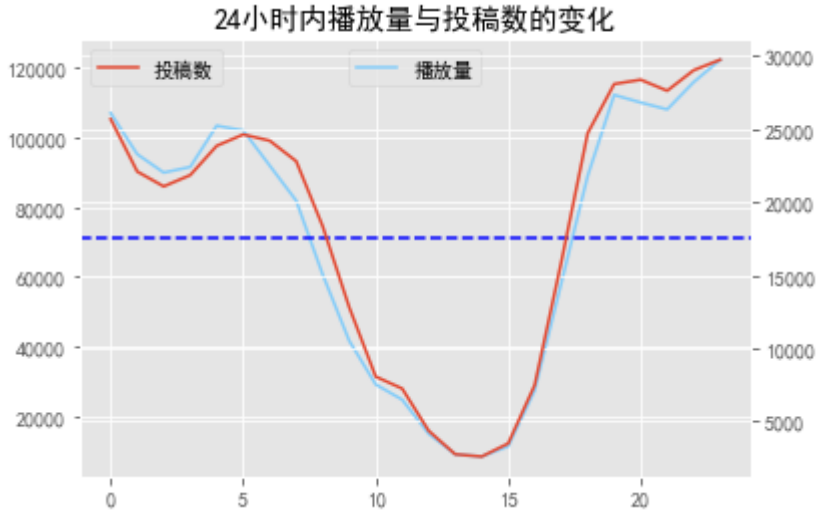
In [22]:

```
#作品发布时间与点赞完播率之间的关系
H_f_l.plot()
plt.title("作品发布时间与点赞完播率之间的关系")
plt.show()
```



In [23]:

```
fig = plt.figure()

ax1 = fig.add_subplot(111)
ax1.plot(data.groupby('H').count()['uid'],c='#87CEFA',label='播放量')
plt.legend(loc='upper center')
plt.axhline(y=data.groupby('H').count()['uid'].mean(),ls='--',c='b')
ax2 = ax1.twinx()
ax2.plot(data.groupby('H')['item_id'].nunique(),label='投稿数')
plt.legend(loc='upper left')

plt.title("24小时内播放量与投稿数的变化")
plt.show()
```



结论3： 不同时间段内发布的作品点赞率和完播率不会有太大变化，整体播放量和投稿数也基本相同这说明播放量和投稿时间关系亦不大，不过还是可以看出0-5点的播放量会略高。
如果投稿最佳时间是在0-5点，但并无特殊优势。

背景音乐与点赞完播率之间的关系

In [24]:

```
#最火的歌曲点赞率和完播率
data.groupby('music_id').mean().loc[data.groupby('music_id').count().sort_values('uid',ascending=Fal
```

Out[24]:

music_id	finish	like	duration_time	H
22.0	0.403623	0.007337	11.729858	10.806888
220.0	0.426520	0.006675	10.866819	10.224621
25.0	0.392903	0.010024	11.204573	11.869727
68.0	0.383736	0.007325	11.342727	9.309602
110.0	0.412637	0.006744	11.187688	10.897785
33.0	0.404485	0.006300	11.454887	10.302295
468.0	0.409408	0.007979	11.315391	10.609943
57.0	0.424433	0.005915	11.116327	10.056040
43.0	0.394313	0.008477	10.870925	11.084611
238.0	0.415926	0.008731	10.900647	11.044786

In [25]:

```
#最火的十首歌歌曲ID
data.groupby('music_id').count().sort_values('uid',ascending=False)[:10].index.tolist()
```

Out[25]:

```
['22.0',
 '220.0',
 '25.0',
 '68.0',
 '110.0',
 '33.0',
 '468.0',
 '57.0',
 '43.0',
 '238.0']
```

In [26]:

```
#最热门歌曲的平均完播率和点赞率
data.groupby('music_id').mean()[data.groupby('music_id').count()>10].mean()
```

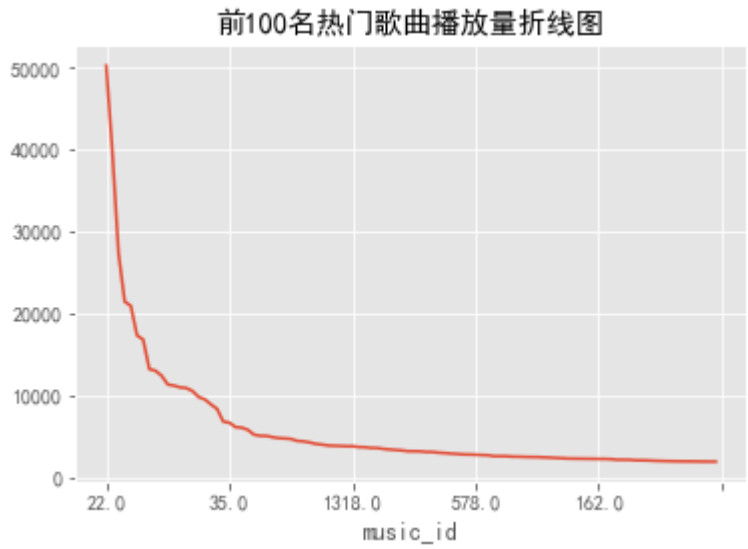
Out[26]:

```
finish             0.396887
like               0.010349
duration_time     10.927413
H                 11.893956
dtype: float64
```

可以看出最热门的歌曲点赞率和完播率也并没有超过平均，可见采用热门歌曲并不能提高自己的完播率和点赞率。

In [27]:

```
#前100名热门歌曲播放量差异
data.groupby('music_id').count().sort_values('uid',ascending=False)['uid'][:100].plot()
plt.title("前100名热门歌曲播放量折线图")
plt.show()
```



结论4： 对于视频配乐更推荐当时最火的歌曲，会比其他歌曲更容易获得高播放量。

In [28]:

```
#热门歌曲每日播放量变化图
data.groupby(['music_id','date']).count()['uid'][data.groupby('music_id').count().sort_values('uid',
```

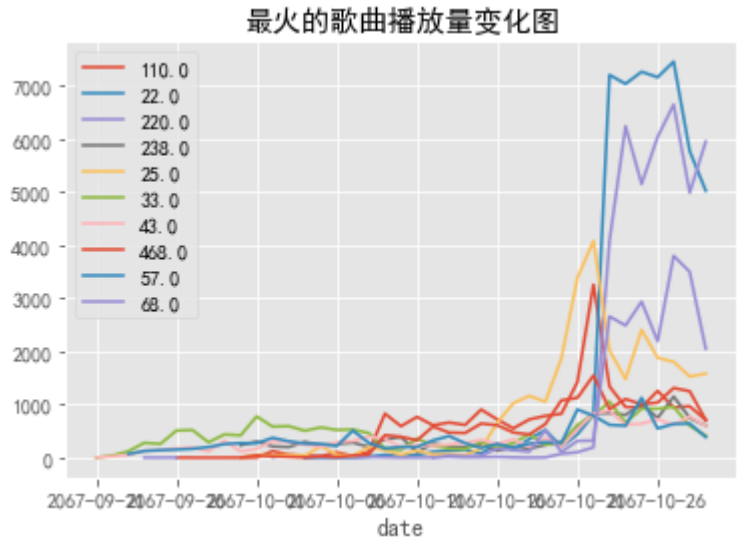
Out[28]:

music_id date	110.0	22.0	220.0	238.0	25.0	33.0	43.0	468.0	57.0	68.0
2067-09-21	NaN	NaN	NaN	NaN	NaN	1.0	2.0	NaN	NaN	NaN
2067-09-22	NaN	NaN	1.0	NaN	NaN	40.0	27.0	NaN	NaN	NaN
2067-09-23	1.0	NaN	NaN	NaN	NaN	121.0	49.0	NaN	70.0	NaN
2067-09-24	NaN	NaN	2.0	NaN	NaN	280.0	130.0	NaN	126.0	NaN
2067-09-25	NaN	NaN	1.0	NaN	NaN	259.0	142.0	NaN	143.0	NaN
2067-09-26	2.0	NaN	3.0	NaN	NaN	511.0	175.0	NaN	153.0	NaN
2067-09-27	1.0	1.0	NaN	NaN	NaN	521.0	190.0	NaN	168.0	NaN
2067-09-28	1.0	NaN	NaN	1.0	NaN	290.0	119.0	NaN	200.0	NaN
2067-09-29	1.0	NaN	NaN	NaN	NaN	439.0	318.0	NaN	257.0	NaN
2067-09-30	2.0	NaN	13.0	235.0	5.0	420.0	117.0	4.0	275.0	NaN
2067-10-01	4.0	NaN	NaN	304.0	57.0	777.0	168.0	43.0	271.0	NaN
2067-10-02	125.0	NaN	11.0	213.0	46.0	585.0	309.0	27.0	374.0	NaN
2067-10-03	58.0	NaN	57.0	200.0	58.0	594.0	299.0	18.0	309.0	NaN
2067-10-04	42.0	1.0	20.0	298.0	38.0	508.0	248.0	5.0	266.0	NaN
2067-10-05	24.0	3.0	11.0	234.0	214.0	569.0	202.0	28.0	260.0	NaN
2067-10-06	104.0	5.0	11.0	258.0	61.0	523.0	293.0	13.0	219.0	NaN
2067-10-07	27.0	3.0	7.0	283.0	39.0	531.0	314.0	42.0	512.0	NaN
2067-10-08	112.0	18.0	14.0	206.0	202.0	462.0	395.0	61.0	277.0	NaN
2067-10-09	829.0	55.0	8.0	345.0	138.0	173.0	357.0	425.0	174.0	NaN
2067-10-10	593.0	27.0	2.0	389.0	56.0	144.0	196.0	393.0	203.0	NaN
2067-10-11	768.0	53.0	3.0	104.0	136.0	363.0	239.0	328.0	201.0	NaN
2067-10-12	611.0	112.0	5.0	261.0	62.0	223.0	222.0	586.0	325.0	4.0
2067-10-13	662.0	127.0	18.0	217.0	67.0	172.0	252.0	473.0	407.0	40.0
2067-10-14	617.0	151.0	6.0	196.0	50.0	162.0	277.0	465.0	273.0	27.0
2067-10-15	906.0	81.0	9.0	260.0	170.0	294.0	337.0	643.0	183.0	39.0
2067-10-16	721.0	259.0	16.0	129.0	658.0	186.0	255.0	612.0	260.0	173.0
2067-10-17	552.0	153.0	6.0	209.0	1025.0	277.0	348.0	475.0	189.0	144.0
2067-10-18	720.0	351.0	1.0	170.0	1164.0	438.0	202.0	436.0	264.0	112.0
2067-10-19	787.0	521.0	5.0	241.0	1053.0	254.0	367.0	633.0	292.0	519.0
2067-10-20	827.0	205.0	74.0	297.0	1868.0	246.0	203.0	1079.0	274.0	109.0
2067-10-21	1425.0	439.0	103.0	576.0	3386.0	617.0	533.0	1127.0	908.0	316.0
2067-10-22	3256.0	808.0	196.0	833.0	4081.0	786.0	830.0	1547.0	790.0	322.0

music_id	110.0	22.0	220.0	238.0	25.0	33.0	43.0	468.0	57.0	68.0
date										
2067-10-23	1341.0	7205.0	4079.0	840.0	2018.0	1058.0	885.0	915.0	618.0	2659.0
2067-10-24	955.0	7037.0	6240.0	804.0	1481.0	657.0	621.0	1105.0	602.0	2489.0
2067-10-25	950.0	7264.0	5152.0	1013.0	2407.0	923.0	640.0	1004.0	1127.0	2938.0
2067-10-26	1257.0	7161.0	6038.0	748.0	1882.0	922.0	724.0	1038.0	550.0	2193.0
2067-10-27	947.0	7455.0	6648.0	1151.0	1804.0	959.0	616.0	1312.0	636.0	3800.0
2067-10-28	969.0	5772.0	4989.0	749.0	1528.0	601.0	778.0	1251.0	643.0	3499.0
2067-10-29	710.0	5025.0	5949.0	606.0	1581.0	415.0	598.0	707.0	388.0	2051.0

In [29]:

```
#抖音最火的歌曲播放量随时间变化图
data.groupby(['music_id','date']).count()['uid'][data.groupby('music_id').count().sort_values('uid',
plt.title("最火的歌曲播放量变化图")
plt.legend(loc='best')
plt.show()
```



从9月21号开始歌曲的播放量有暴增，但目前还不知道原因。不过可以看出抖音确实存在“神曲”，在短时间内播放量暴增。街头巷尾都会播放。
对于普通人来说，抖音“神曲”会让人感到些许不适，那么这个现象是否存在？是否会在抖音用户群中表现出来？

In [30]:

```
#同一首歌随着时间变化的完播率和点赞率变化
data.groupby(['music_id','date']).mean()
```

Out[30]:

		finish	like	duration_time	H
music_id	date				
1.0	2067-09-24	0.333333	0.0	9.666667	12.333333
	2067-09-26	0.000000	0.0	20.000000	0.000000
	2067-09-27	0.428571	0.0	13.000000	11.571429
	2067-09-28	0.000000	0.0	9.000000	22.000000
	2067-09-29	0.333333	0.0	9.333333	15.333333
...	...	...	...	...	...
9998.0	2067-10-29	0.000000	0.0	7.500000	4.000000
9999.0	2067-09-28	0.000000	0.0	7.000000	5.000000
	2067-10-14	0.000000	0.0	10.000000	0.000000
	2067-10-24	0.285714	0.0	11.000000	4.000000
	2067-10-25	0.366667	0.0	9.000000	19.000000

153487 rows × 4 columns

In [31]:

```
data.groupby(['music_id','date']).mean()['finish'][data.groupby('music_id').count().sort_values('uid
```

Out[31]:

```
music_id  date
110.0     2067-09-23    0.000000
          2067-09-26    0.500000
          2067-09-27    0.000000
          2067-09-28    0.000000
          2067-09-29    0.000000
                          ...
68.0      2067-10-25    0.376447
          2067-10-26    0.354309
          2067-10-27    0.372368
          2067-10-28    0.435267
          2067-10-29    0.369576
Name: finish, Length: 319, dtype: float64
```

In [32]:

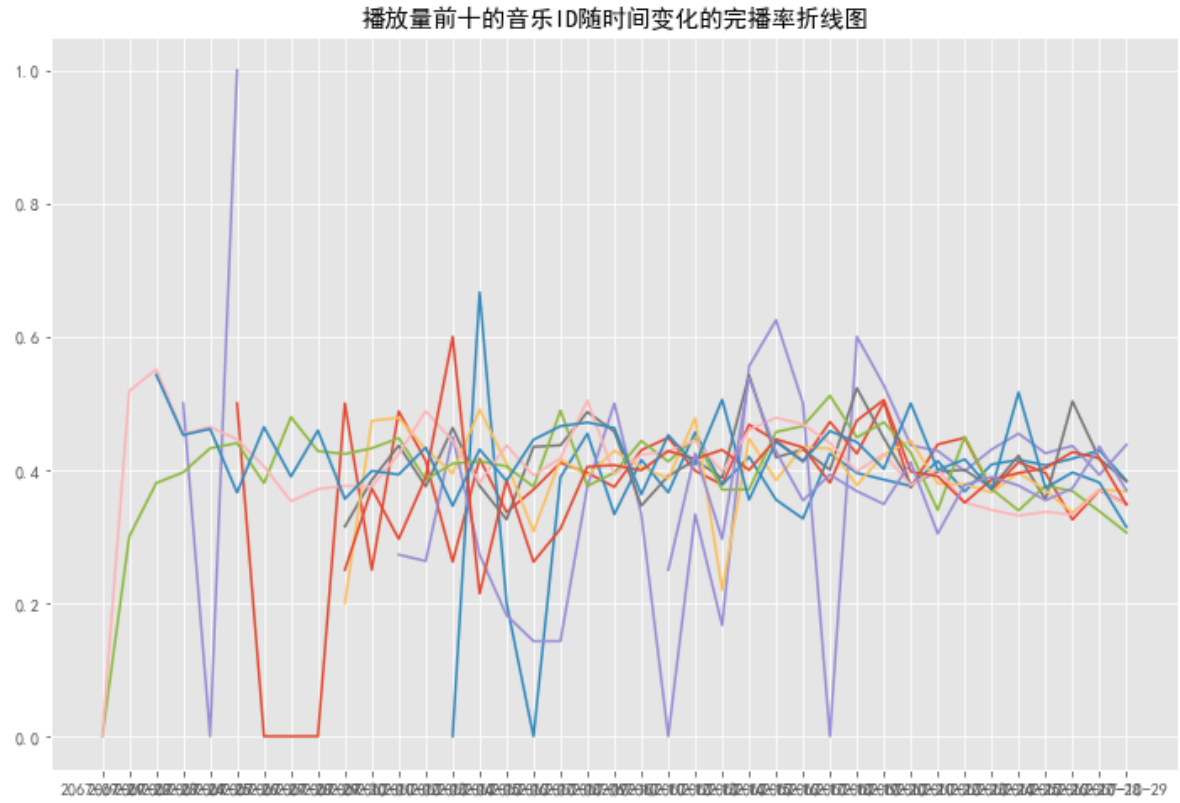
```
#播放量前十的音乐ID随时间变化的完播率
top10_f = data.groupby(['music_id','date']).mean()['finish'][data.groupby('music_id').count().sort_v
```

In [33]:

```
#播放量前十的音乐ID随时间变化的完播率
top10_l = data.groupby(['music_id','date']).mean()['like'][data.groupby('music_id').count().sort_val
```

In [34]:

```
plt.figure(figsize=(12,8))
plt.plot(top10_f.unstack().T)
plt.title("播放量前十的音乐ID随时间变化的完播率折线图")
plt.show()
```



In [35]:

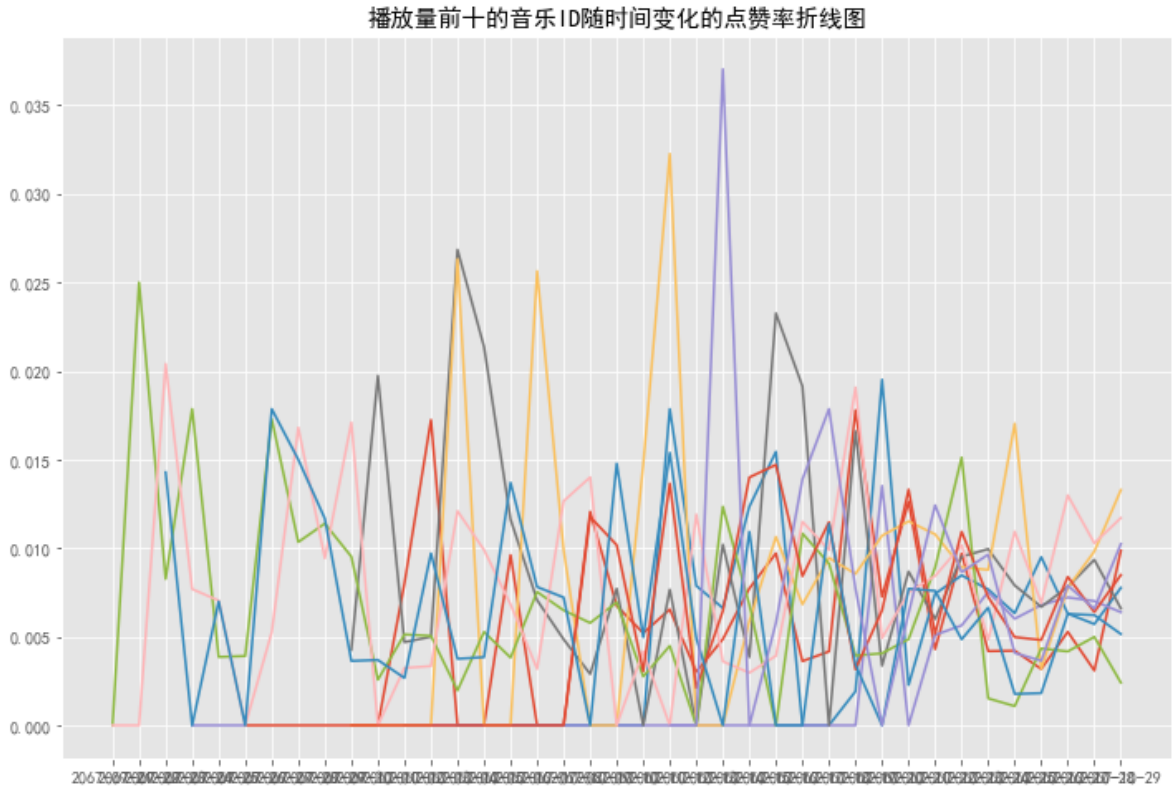
```
#播放量前十的音乐ID随时间变化的完播率线性相关系数
top10_f.unstack().T.corr().mean().mean()
```

Out[35]:

0.1819420336496958

In [36]:

```
plt.figure(figsize=(12,8))
plt.plot(top10_l.unstack().T.sort_index())
plt.title("播放量前十的音乐ID随时间变化的点赞率折线图")
plt.show()
```



In [37]:

```
#播放量前十的音乐ID随时间变化的点赞率线性相关系数
top10_l.unstack().T.corr().mean().mean()
```

Out[37]:

0.08343501293186688

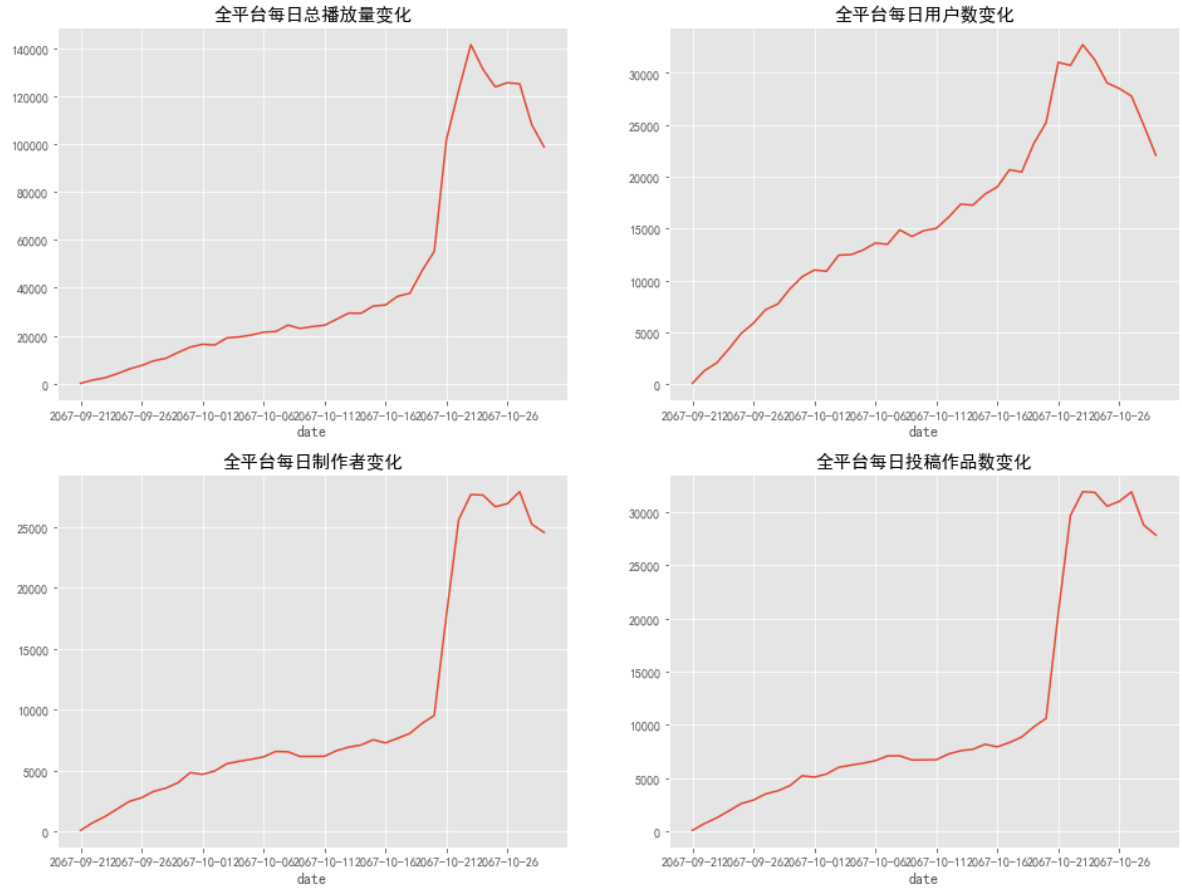
结论5： 虽然不确定抖音以外的人是否存在讨厌的情绪，至少站内流行歌曲ID并没有表现出日期与点赞和完播率之间的强线性关系。

4.2 抖音全平台视频数据分析

- 全平台每日总播放量变化
- 全平台每日用户数变化
- 全平台每日制作者变化
- 全平台每日投稿作品数变化
- 活动播放量异常的人数
- 各个作者为平台贡献的总播放数

In [38]:

```
plt.figure(figsize=(16,12))
plt.subplot(221)
data.groupby('date').count()['uid'].plot()
plt.title("全平台每日总播放量变化")
plt.subplot(222)
data.groupby('date')['uid'].nunique().plot()
plt.title("全平台每日用户数变化")
plt.subplot(223)
data.groupby('date')['author_id'].nunique().plot()
plt.title("全平台每日制作者变化")
plt.subplot(224)
data.groupby('date')['item_id'].nunique().plot()
plt.title("全平台每日投稿作品数变化")
plt.show()
```



可以看出全平台在9-21到10-30日之间，无论是播放量还是用户数所有指标都在上升，尤其是10-21日有一波大量上涨，应该是平台做了一个大型活动。
不过也可以明显看出来用户数并没有播放量上涨的如此夸张，而且制作者和投稿数量也达到了前日的150%，初步猜测是有人（工作室）利用平台规则漏洞，创建大量新号，并使用机器人刷单牟利。
不过遗憾由于没有之后更长时期的数据，难以评估此次活动的效果。不过可以尝试检查出其中的机器人帐号。

In [39]:

```
#观看量异常人数
exception = data.groupby(['uid','date']).count()[data.groupby(['uid','date']).count()>1].dropna()['u
exception.index=exception.index.astype('datetime64[ns]')
exception
```

Out[39]:

uid date	0	1	10	1000	10000	10001	10002	10003	10004	10006	...	9990	9991	999
2067-09-21	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-22	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-23	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-24	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	2.0	NaN
2067-09-25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-27	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-28	NaN	NaN	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-29	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-09-30	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.0	2.0	NaN
2067-10-01	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-10-02	NaN	NaN	12.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-10-03	NaN	NaN	10.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	2.0	NaN	NaN
2067-10-04	NaN	NaN	7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
2067-10-05	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.0	NaN	NaN
2067-10-06	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.0	3.0	NaN
2067-10-07	NaN	NaN	7.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	3.0	NaN	NaN
2067-10-08	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	2.0	NaN	...	NaN	5.0	NaN
2067-10-09	2.0	NaN	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.0	NaN	NaN
2067-10-10	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	3.0	NaN	...	3.0	NaN	NaN

uid date	0	1	10	1000	10000	10001	10002	10003	10004	10006	...	9990	9991	999
2067-10-11	NaN	NaN	6.0	NaN	NaN	NaN	NaN	NaN	2.0	NaN	...	3.0	NaN	NaN
2067-10-12	NaN	NaN	5.0	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	3.0	2.0	NaN
2067-10-13	NaN	NaN	8.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	4.0	NaN	NaN
2067-10-14	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	3.0	2.0	NaN
2067-10-15	NaN	NaN	9.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	2.0	2.0	NaN
2067-10-16	NaN	2.0	10.0	NaN	NaN	2.0	NaN	NaN	3.0	NaN	...	6.0	NaN	NaN
2067-10-17	NaN	NaN	4.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	3.0	2.0	NaN
2067-10-18	NaN	NaN	4.0	NaN	NaN	4.0	NaN	NaN	2.0	NaN	...	2.0	3.0	NaN
2067-10-19	NaN	NaN	8.0	NaN	NaN	2.0	NaN	NaN	3.0	NaN	...	6.0	2.0	NaN
2067-10-20	3.0	2.0	5.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	NaN	NaN	2.(
2067-10-21	NaN	NaN	6.0	3.0	2.0	5.0	NaN	NaN	6.0	NaN	...	12.0	6.0	NaN
2067-10-22	3.0	NaN	12.0	NaN	NaN	7.0	2.0	2.0	2.0	NaN	...	9.0	10.0	NaN
2067-10-23	3.0	NaN	17.0	4.0	2.0	17.0	2.0	4.0	3.0	NaN	...	9.0	22.0	NaN
2067-10-24	4.0	NaN	14.0	2.0	3.0	32.0	6.0	3.0	4.0	NaN	...	7.0	11.0	NaN
2067-10-25	5.0	6.0	6.0	8.0	NaN	7.0	6.0	6.0	2.0	NaN	...	11.0	8.0	NaN
2067-10-26	3.0	2.0	3.0	6.0	NaN	29.0	NaN	5.0	3.0	NaN	...	12.0	13.0	NaN
2067-10-27	NaN	4.0	24.0	11.0	2.0	20.0	3.0	3.0	4.0	NaN	...	2.0	8.0	NaN
2067-10-28	4.0	NaN	20.0	4.0	4.0	20.0	7.0	NaN	2.0	NaN	...	6.0	14.0	NaN
2067-10-29	NaN	2.0	12.0	2.0	2.0	12.0	4.0	4.0	5.0	6.0	...	9.0	17.0	2.(

39 rows × 41044 columns

In [40]:

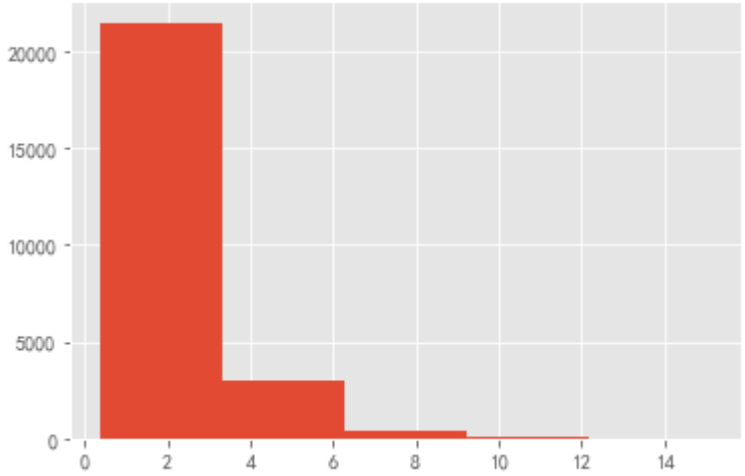
```
#（预测）活动开始前后每人平均观看量的变化倍数
times = exception.query('date>datetime(2067,10,21) and date<datetime(2067,10,29)').mean() / exceptic
```

In [41]:

```
#变化倍数直方图概览
times.hist(bins=5)
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bd3407888>
```



In [42]:

```
#倍数统计概览
times.describe()
```

Out[42]:

```
count    24856.000000
mean         2.109446
std          1.332468
min          0.400000
25%          1.230769
50%          1.700000
75%          2.500000
max         15.112782
dtype: float64
```

In [43]:

```
#机器人（绝大多数人的变化倍数在0-3倍之间故选取3倍）
robot = times[times>3].index.tolist()
len(robot)
```

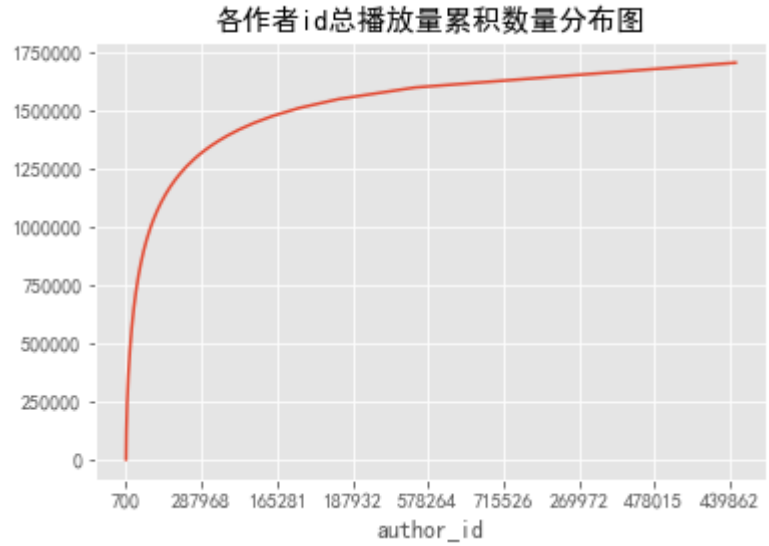
Out[43]:

```
4267
```

结论6： 有4267是机器人的嫌疑，而此次活动新增用户也不过6008人，这进一步印证了现在的网络是以存量市场为主，如果想要吸引新的创作者和用户那么可能更需要去新的市场比如海外或者在抖音外的市场做宣传可能会效果更好。

In [44]:

```
#各作者id总播放量累积数量分布图
data.groupby('author_id').count().sort_values('uid',ascending=False)['uid'].cumsum().plot()
plt.title("各作者id总播放量累积数量分布图")
plt.show()
```



结论7： 可以看出目前抖音整体用户播放十分符合帕累托分布，极少数的制作者吸引了全平台绝大部分流量。如果还想在此基础上再次提升，培养头部制作者，或者挖掘其他平台的头部制作者是比单纯给制作奖励更加有效的方法。具体的效果如何还需要更多数据支撑才能得出结论。

5.总结

5.1 抖音网红建议

1. 抖音98.52%的流量都会流向算法推荐视频，获得算法推荐是获得更多播放的关键所在。
2. 最重要的始终是题材的选择，初始流量池大更容易获得算法青睐。
3. 除去题材外，投稿的最佳时间日常是在0-5点，有平台活动一定要参与。
4. 视频时长最好在7-10s，其次是0-6s及23s以内，最长也不建议超过40s。
5. 背景音乐最好选择当下最流行的歌曲。

5.2 平台运营建议

1. 抖音活动时有大量机器人存在，需要决定是否清除（疑似机器人的[uid]保存在“robot”列表中）。
2. 站内活动小国初见不错，但在去除机器人后并没有大量实质增长，收益很低，再次举办需要慎重。
3. 平台用户始终稳步增长中，但如果想要大量增长，考虑其他渠道，或者开辟新市场是更好的选择。
4. 排名前20%的视频制作者占据了整个平台80%以上的流量，培养或者挖掘已存在的好视频制作者是未来保持流量的关键。