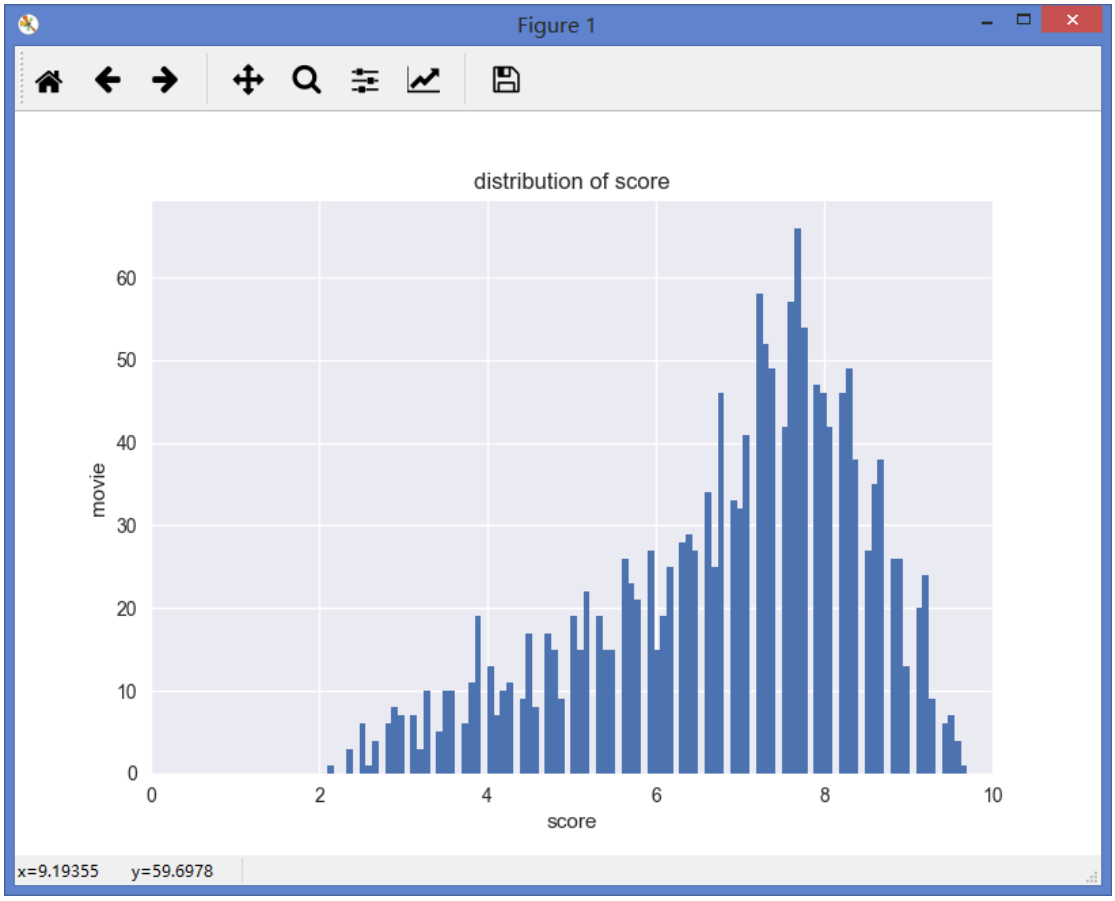
2017 数据科学夏令营

豆瓣电影作业笔记 （第三题）

王力为　数理统计
语言：Python3.6
环境：Pycharm+Anaconda

一、



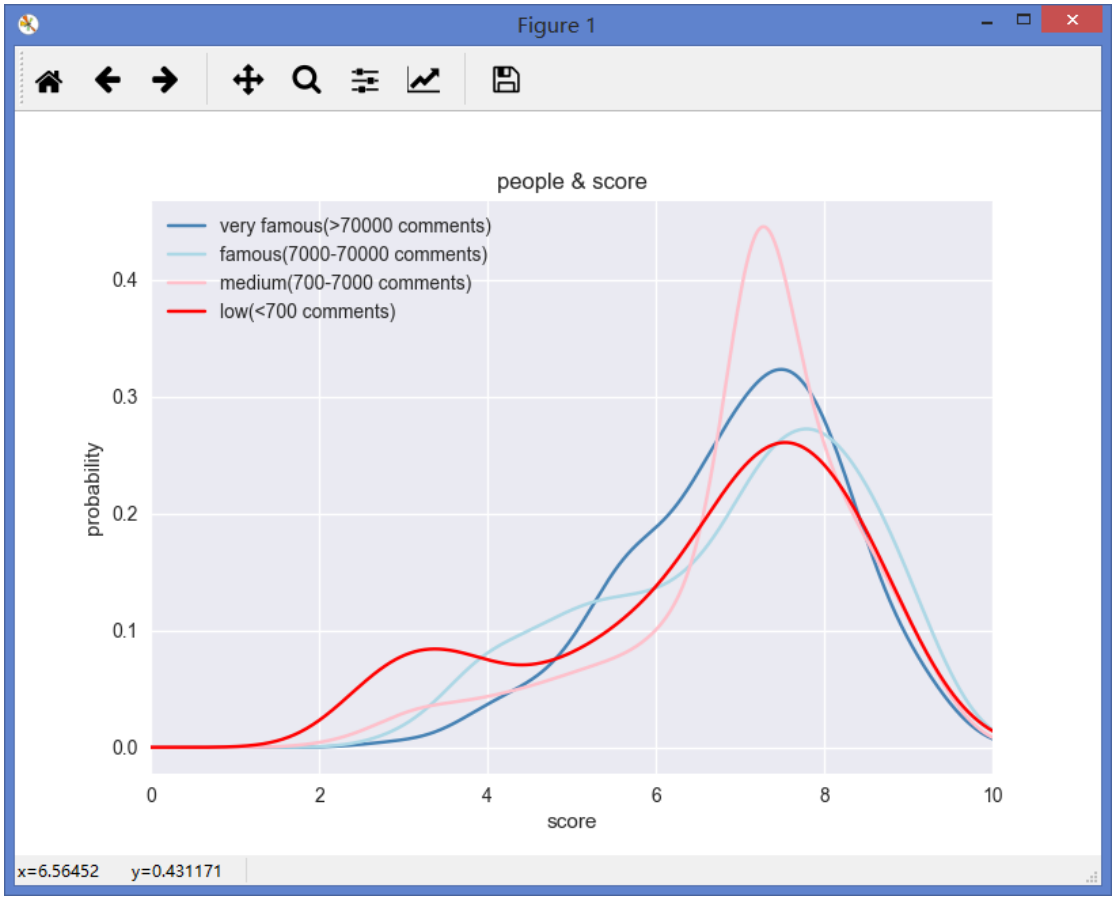
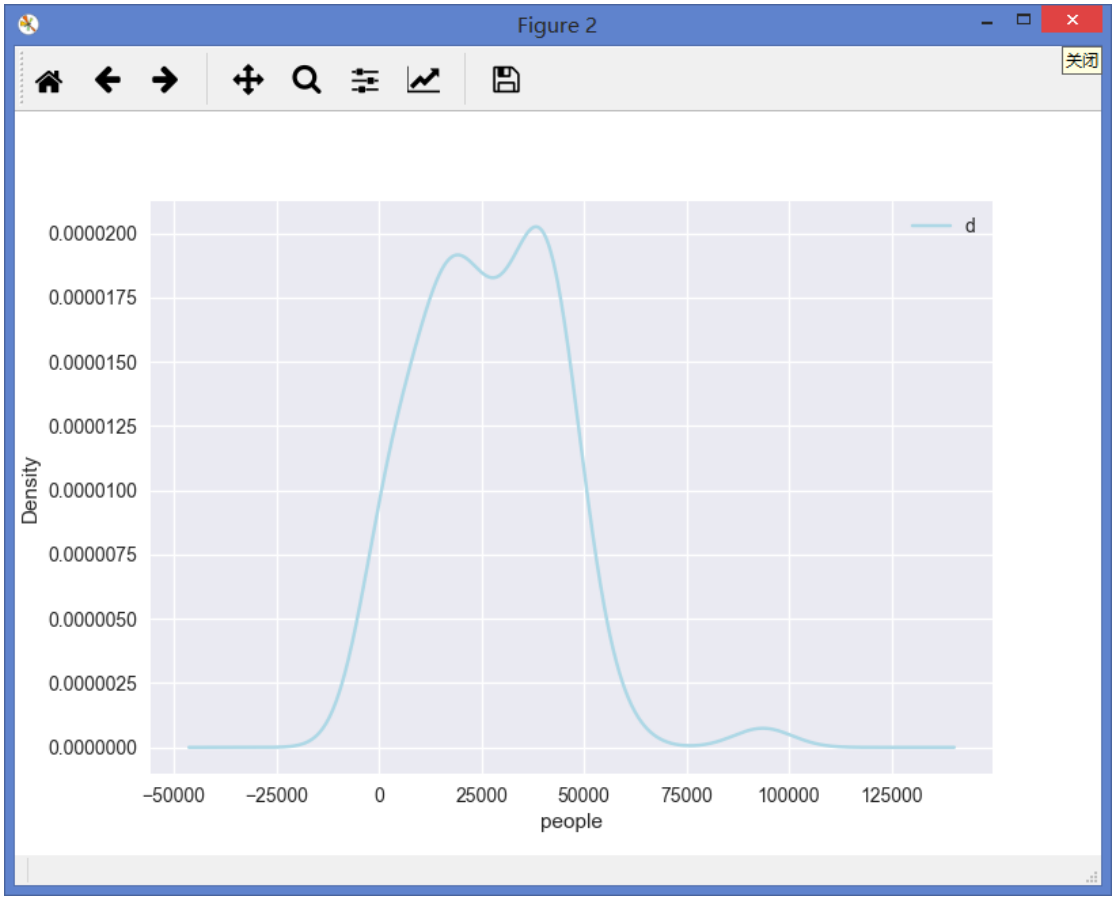


Figure 1
people & score
very famous(>70000 comments)
famous(7000-70000 comments)
medium(700-7000 comments)
low(<700 comments)
probability
score
0.0
0.1
0.2
0.3
0.4
0
2
4
6
8
10
x=6.56452 y=0.431171


Figure 1
people & score
movie
people
2
3
4
5
6
7
8
9
10
0
100000
200000
300000
400000
500000
600000
x=391587 y=9.06745



画了几个图发现并不能得到什么有用的信息，究其原因是 float/int 型数据太少，其他几乎都是汉子类型，较难操作。这方面经验实在不足，完成的不好，还请多多指教。

但是这个没完。我发现，上映年份这一个栏目完全可以利用起来，如果把它做成可用的 float/int 型数据怎么样呢？

为了达到这一点，我重新写了个爬虫从头开始。网上关于豆瓣 TOP250 的爬虫讲解太多了。这次我选择的是豆瓣电影标签——爱情这一栏目的电影的爬取和数据处理

更加详尽的分析过程和编码在后面

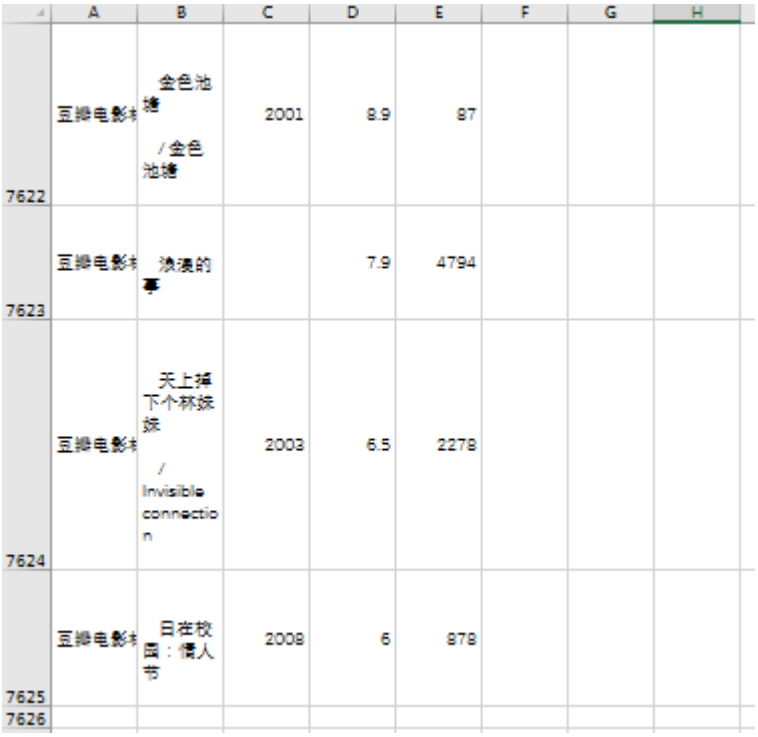
二、补充：

由于 float 型数据太少，我写了个代码自己在爬取数据上将上映年份的值也做了转换，以便利用其来分析年份、分数与热度的关系。

①数据采集：

以爱情标签为例。
我将爱情标签 391 页数据爬取下来。
其中分成完整版的 csv 数据和只有三个 float 型数据的 csv 两份

中途主要是遇到了从 html 标签查找定位内容、内容的提取（函数及正则等）、爬取输出的格式（如何写成可以导入的 csv）、换页（网页 URL 构造）、缺失值判断（ifelif 语句）、编码转换（utf-8 的大坑，中文字不知道如何是好）、网站反爬（IP）等一系列问题，然后一步步解决。

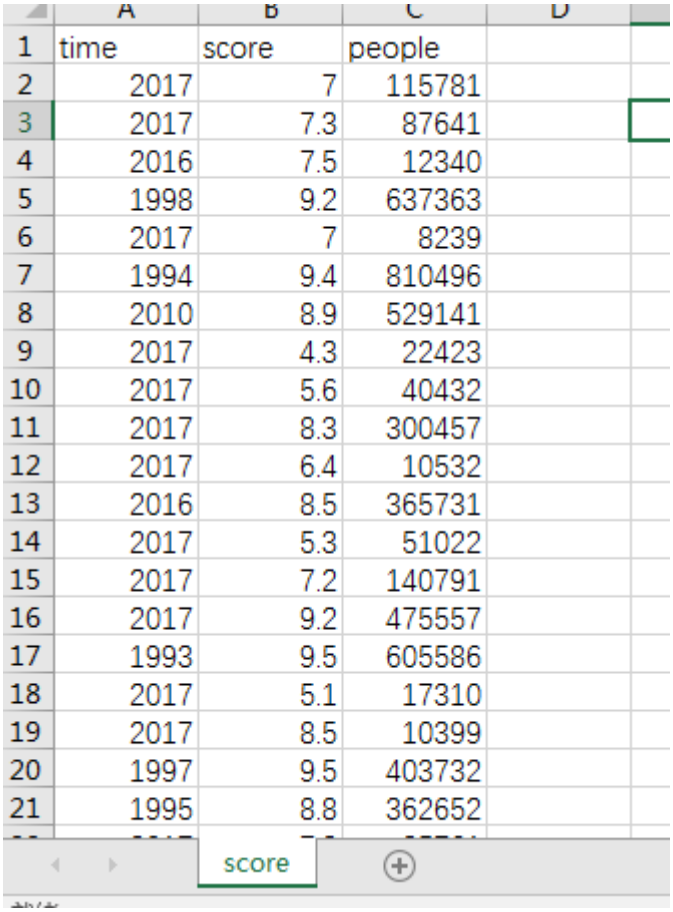


	A	B	C	D	E	F	G	H
7622	豆瓣电影	金色池塘 /金色池塘	2001	8.9	87			
7623	豆瓣电影	浪漫的事		7.9	4794			
7624	豆瓣电影	天上掉下个林妹妹 / Invisible connection	2003	6.5	2278			
7625	豆瓣电影	日在校园：情人节	2008	6	878			
7626								

为了方便后面的分析，特意写多了一个只有 int 和 float 型的 csv 文件。可用变量即上映时间、评分和评价人数

注：

遇到问题：有效采集和能够挖掘的数据太少。文字数据不会使用



	A	B	C	D
1	time	score	people	
2	2017	7	115781	
3	2017	7.3	87641	
4	2016	7.5	12340	
5	1998	9.2	637363	
6	2017	7	8239	
7	1994	9.4	810496	
8	2010	8.9	529141	
9	2017	4.3	22423	
10	2017	5.6	40432	
11	2017	8.3	300457	
12	2017	6.4	10532	
13	2016	8.5	365731	
14	2017	5.3	51022	
15	2017	7.2	140791	
16	2017	9.2	475557	
17	1993	9.5	605586	
18	2017	5.1	17310	
19	2017	8.5	10399	
20	1997	9.5	403732	
21	1995	8.8	362652	

score

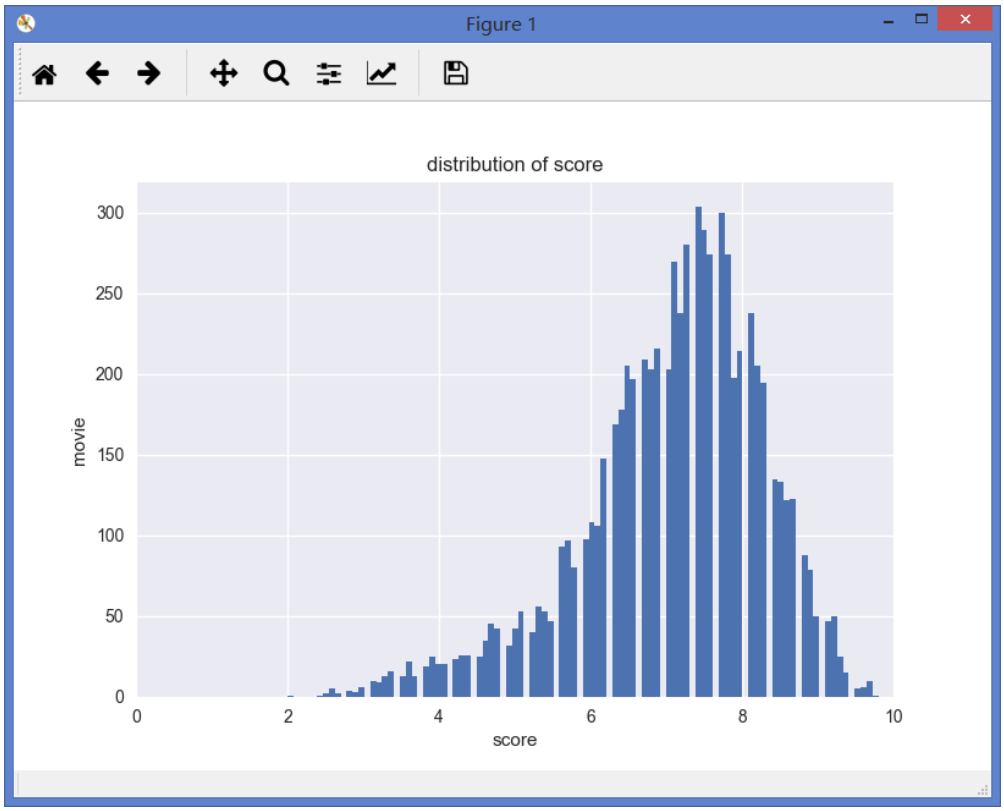
②数据预处理

填充 time 和 people 两栏，然后用 KNN 的方法建模，填充 score 的缺失值。然而效果一般般…由于这里多了一项 time 的数据可使用，建模效果应该更好。

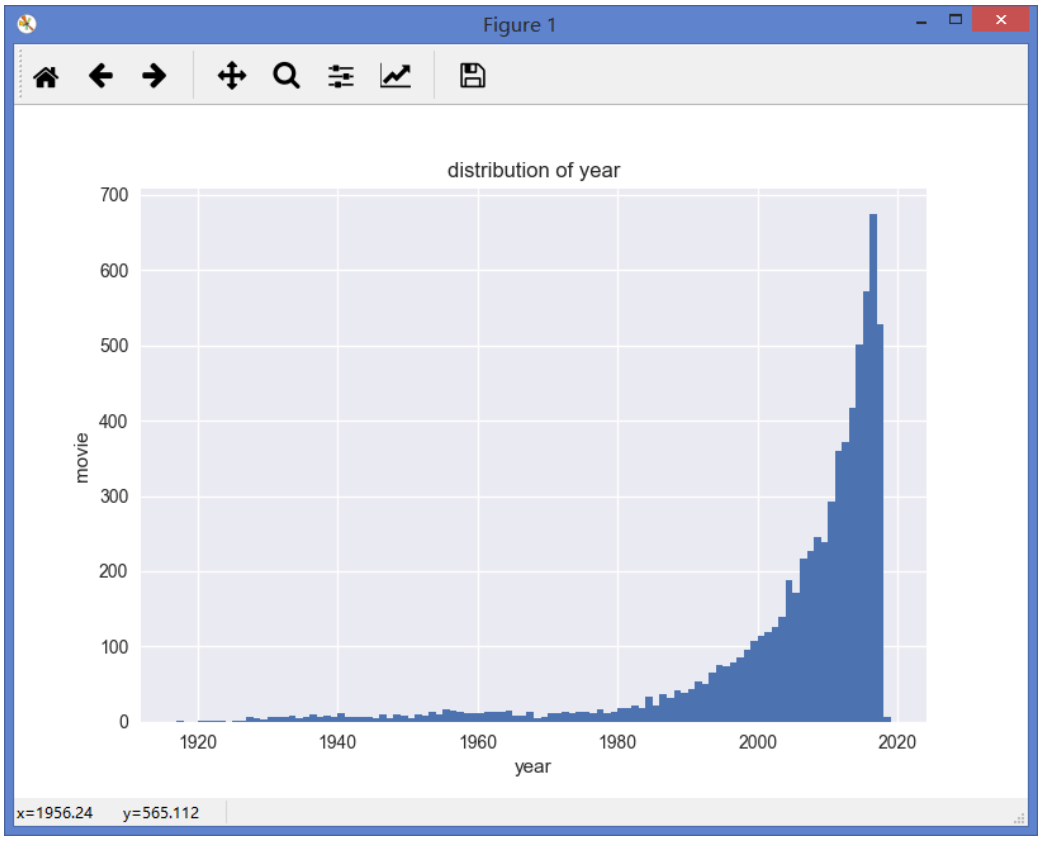
遇到问题：数据太少，且全都有缺失值，所以要先用直接填充的方法（填充平均值）填充另外两个变量再建模预测 score。不能直接用建模的方法填充

③数据可视化

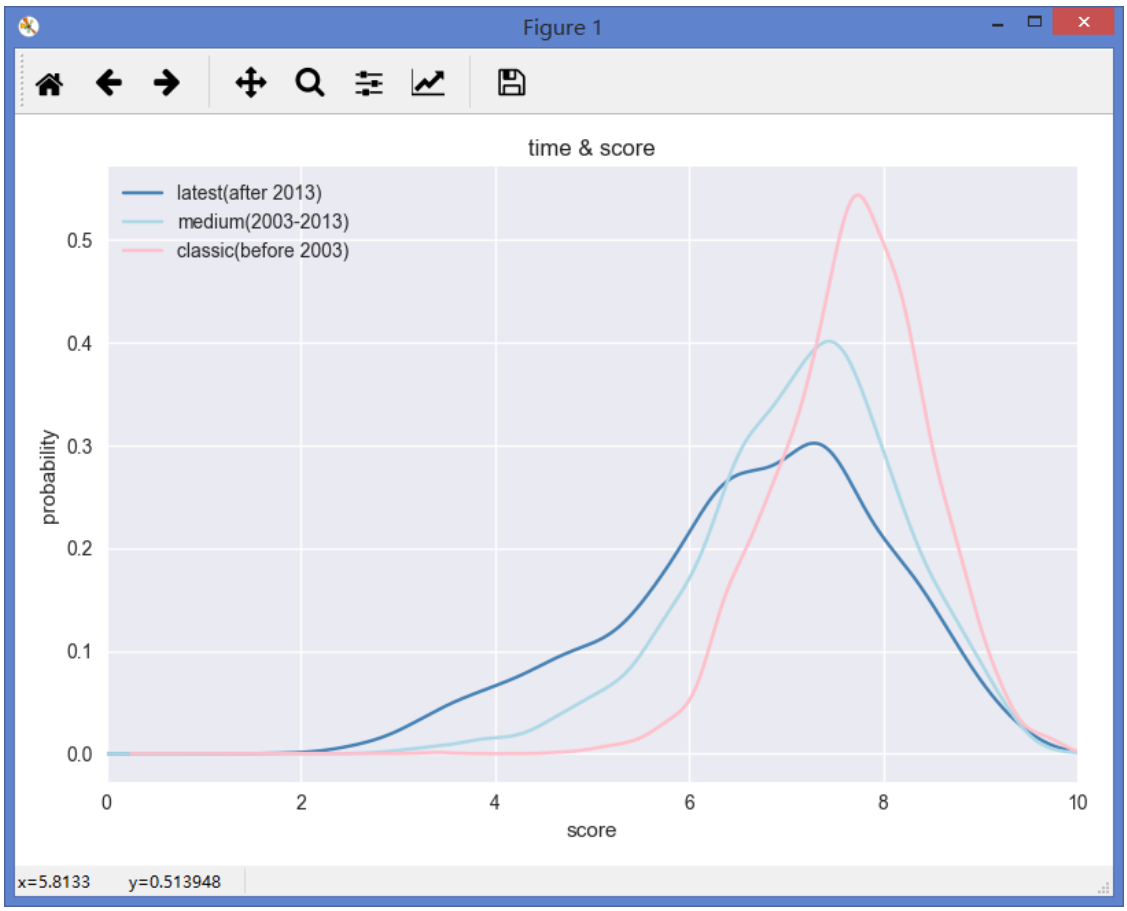
总共 11 张图。

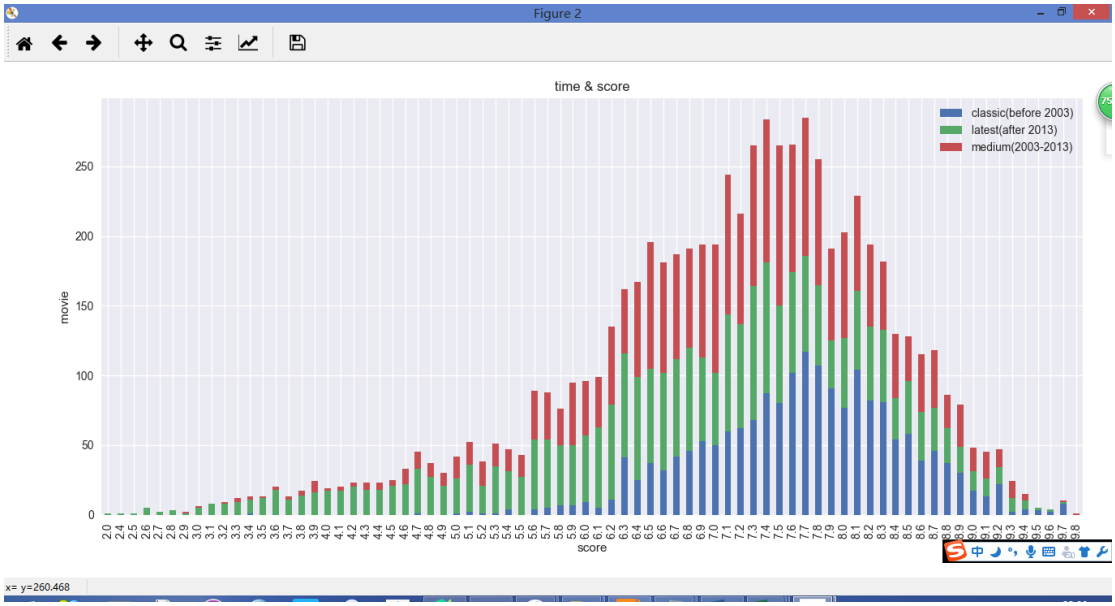


①#了解分数的分布。当然这一点 describe 里也有
可见整体还是呈正态分布的



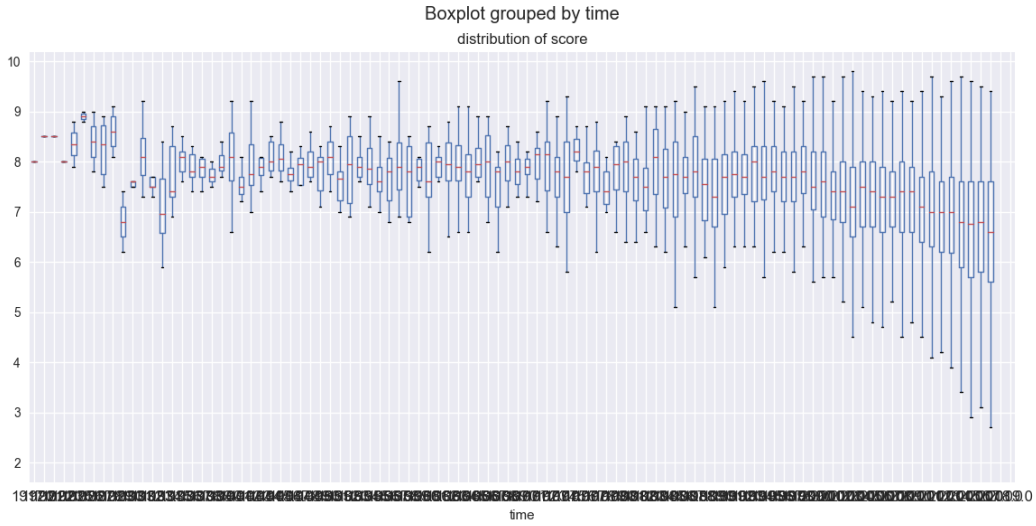
②了解电影制造年份的分布。同理 describe 就有
很明显 80 年代以特别是 2000 年开始呈指数型增长





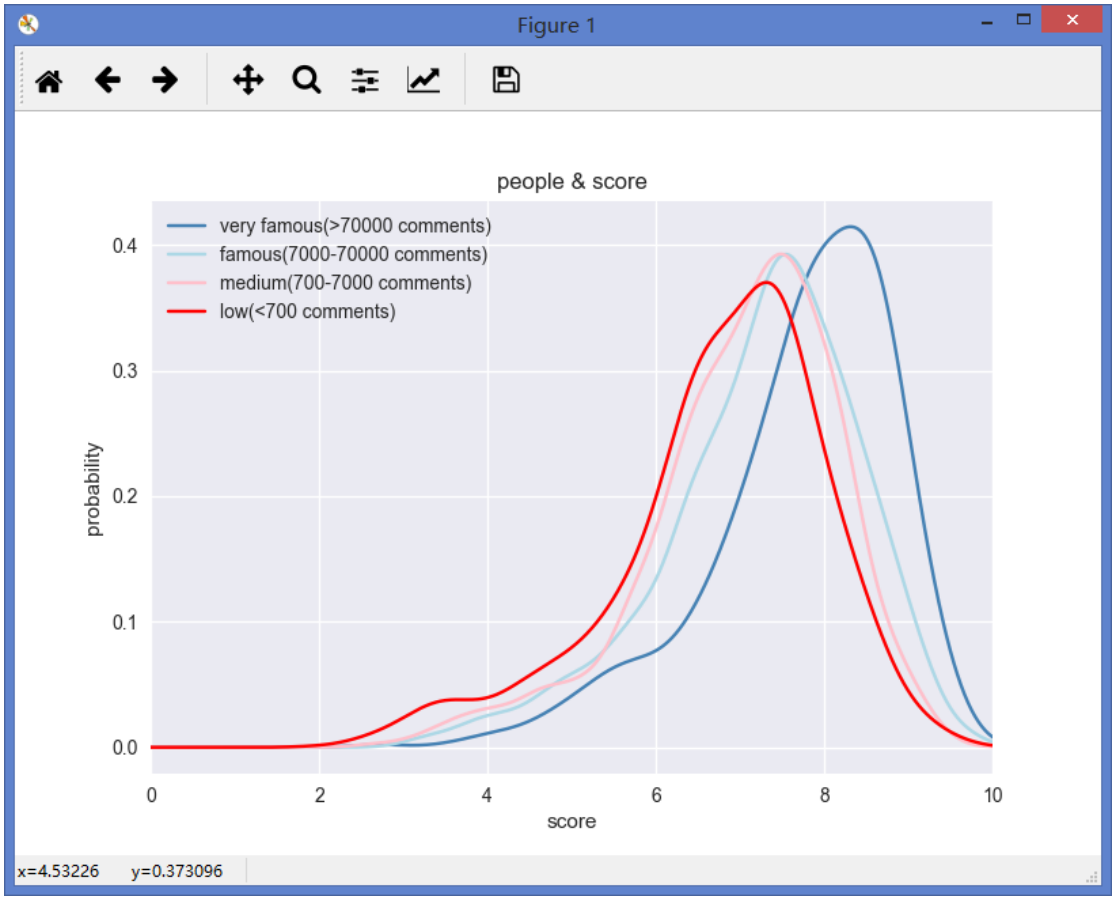
③了解年份与分数的关系（分成 3 组）：
一般来说电影上映年代越久，整体质量越好。2013 年后上映的电影，有重尾分布，且均值不及 2003 年以前的电影。这也说明目前的电影质量不如从前。
④由图二，可知 2013 年后上映的电影低分段重灾区明显！要么好片，要么就大烂片，而且

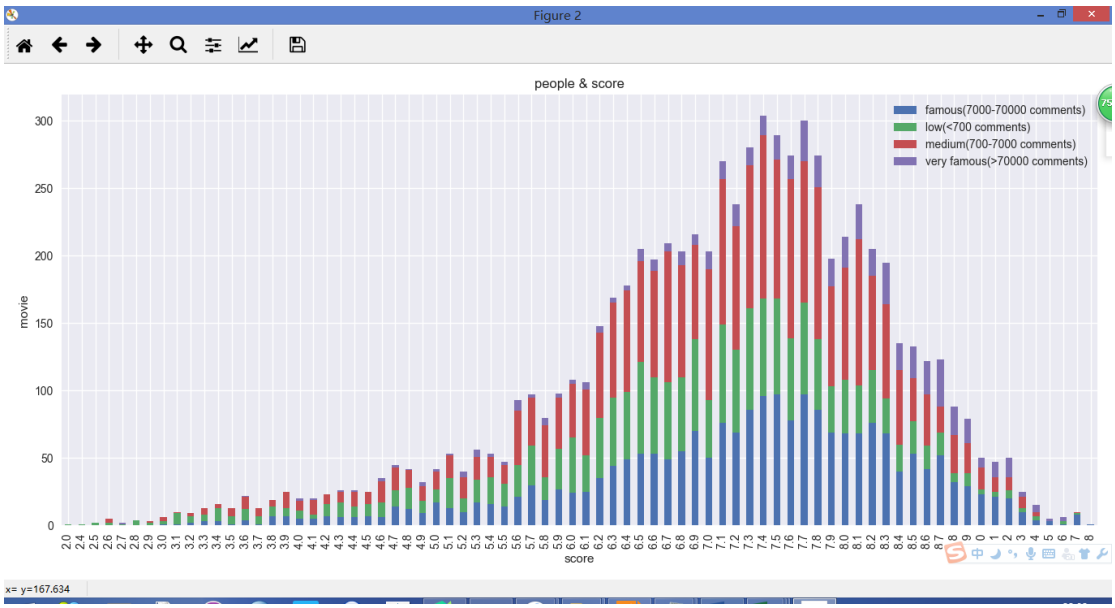
由于数量多，相比之下烂片占比很大



⑤#了解年份与分数的关系，一般来说越新的电影，分数的标准差越大，是异方差的（当然也与样本量有关）

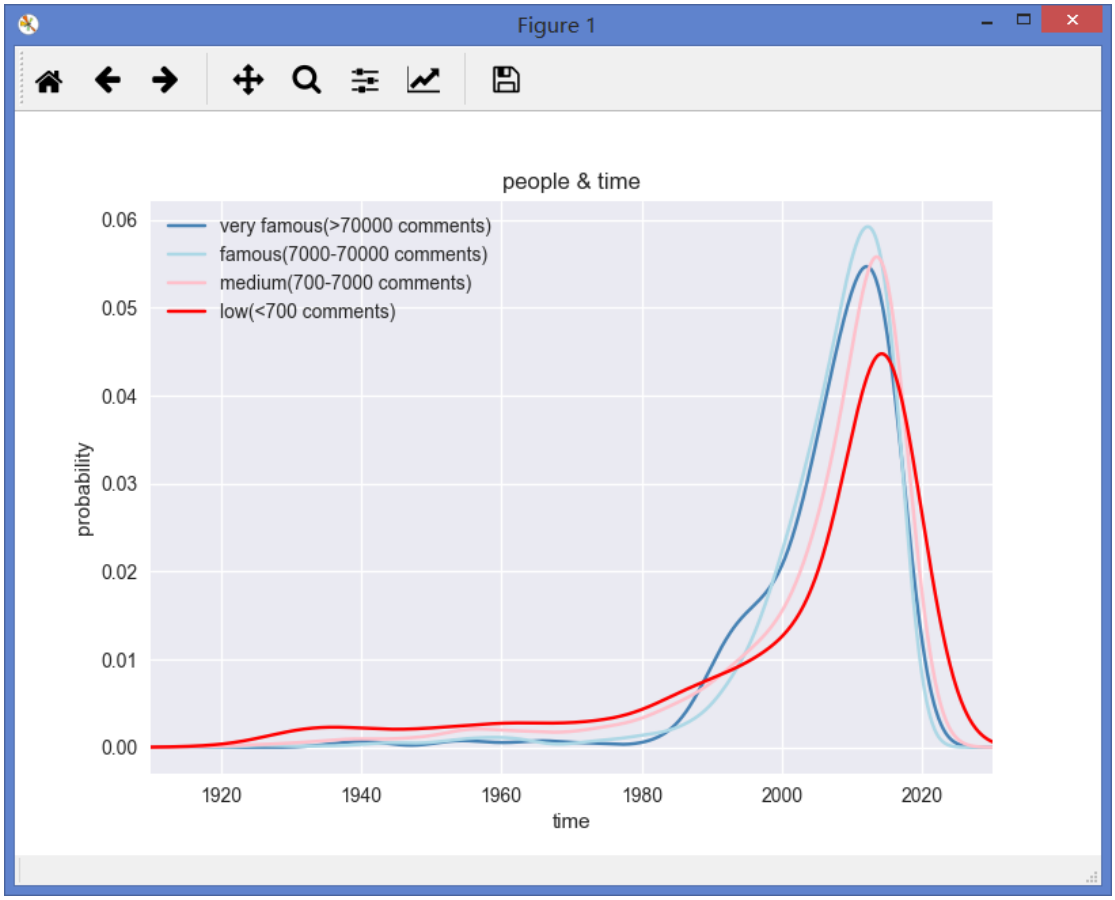
#同样可见，近年电影评分浮动很大。而 1985-2005 这一段时间，整体作品质量都比较上乘

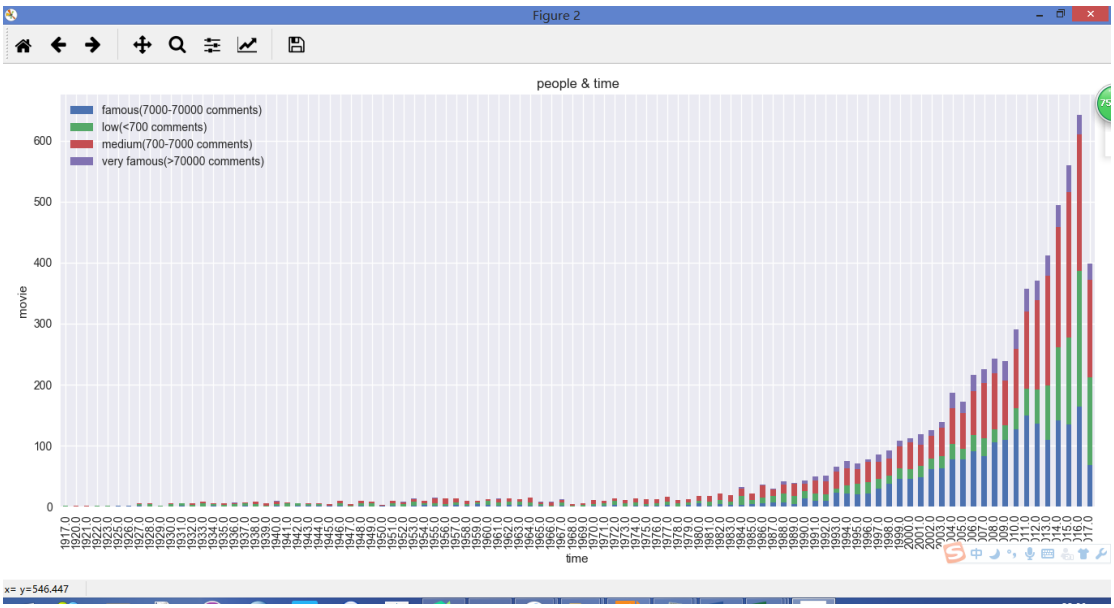




⑥⑦了解评价人数与分数的关系（分成四类）：

一般来说评价人数与分数成正相关，8 分以上的好片热度都不错，评论数量较多。

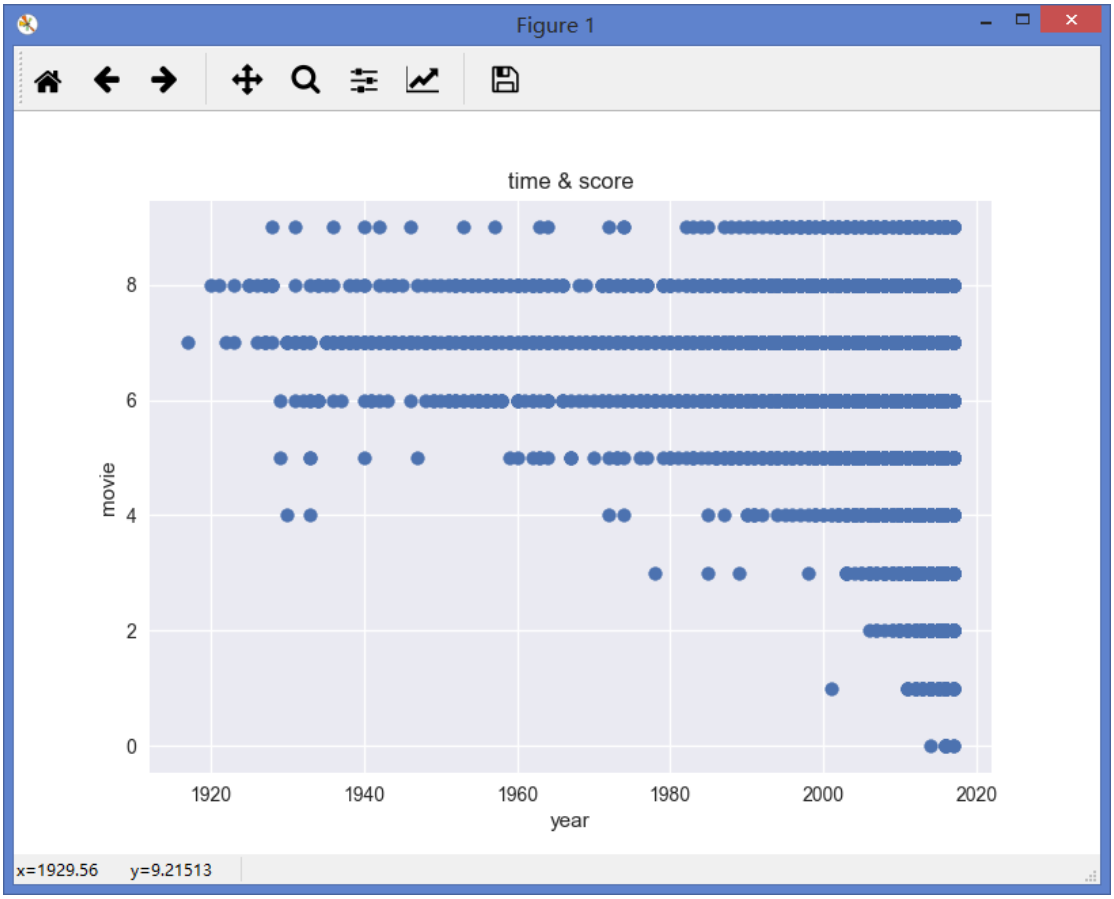




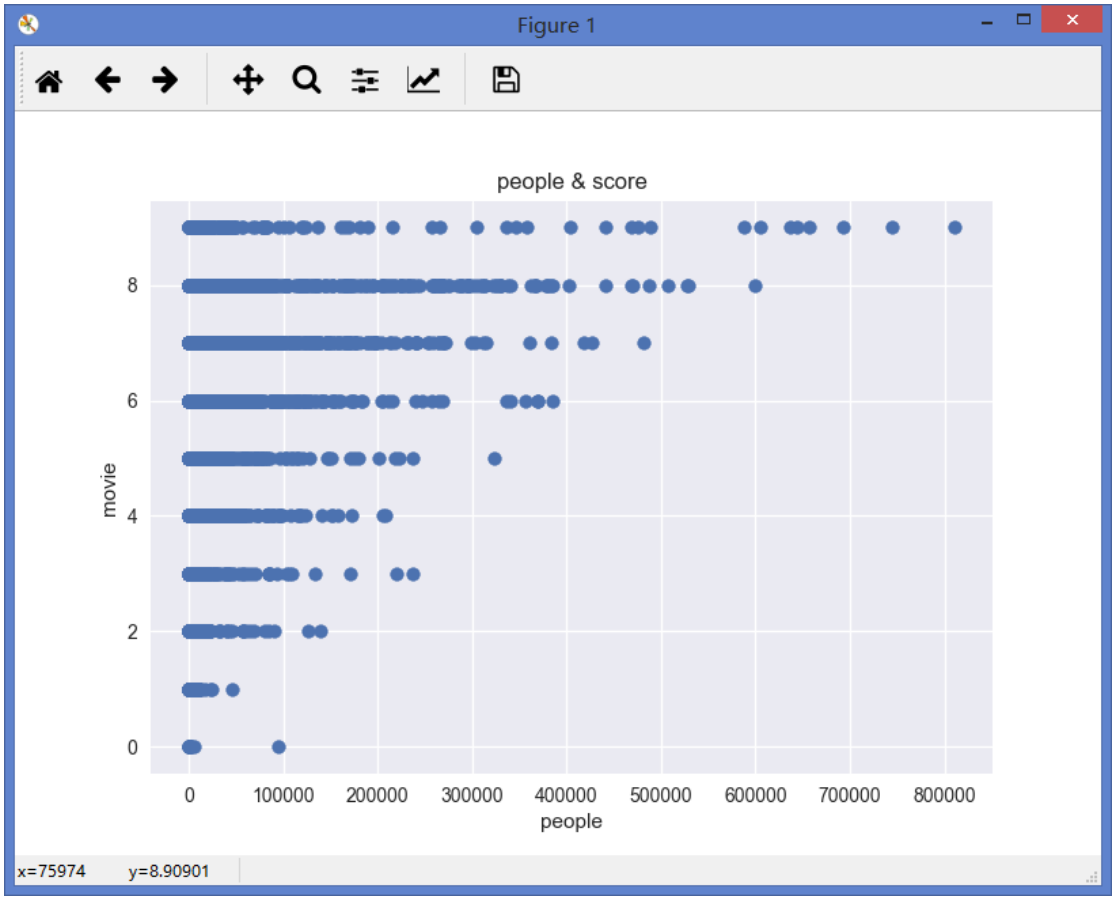
⑧⑨了解评价人数与上映时间的关系：

一般来说评价人数（热度）与上映时间成负相关。

以 1980 年后做说明，2010 年后电影鱼龙混杂小片成群，相比之下低热度小众电影越来越多（可能原因，子题材越来越多，面向观众趋于多样化和小众，热度较低。且质量稂莠不齐）。而 2010 年前的电影大多都较经典，热度较高



10.分组 10 组。老片分数一般很集中在 6-9 分的中高分段，新片则两极分化愈发严重。



11.分组 10 组。越高分的电影评分人数越多（热度越高），呈正相关关系

遇到问题：

①图不够美观，种类不够多
②数据挖掘信息太少
③文字信息没有用到

④总结：

①近年电影评分浮动很大。而 1985-2005 这一段时间，整体作品质量都比较上乘
②老片分数一般很集中在 6-9 分的中高分段，新片则两极分化愈发严重。
③2010 年后电影鱼龙混杂小片成群，相比之下低热度小众电影越来越多（可能原因，子题材越来越多，面向观众趋于多样化和小众，热度较低。且质量稂莠不齐）。而 2010 年前的电影大多都较经典，热度较高。特别 2013 年后上映的电影低分段重灾区明显！要么好片，要么就大烂片，而且由于数量多，相比之下烂片占比很大
④越高分的电影评分人数越多（热度越高），呈正相关关系

所以：

①一般来说主旋律电影会比较保险，特别是以前（85-05 年代）那样的经典电影，可集中参考它们是如何制作的。尤其避免两极分化
②一般来说，评分与热度有一定关系。放映前将口碑热度打起来，扩大宣传不是一件坏事。尽量避免小众、小题材电影

附：（代码）

—

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sp
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsRegressor
pd.options.mode.chained_assignment = None
import codecs
df1 = pd.ExcelFile("豆瓣·电影.xlsx") #读取数据
df = df1.parse('Sheet1')
print(df.shape)
print(df.info())
print(df.describe())

fig=plt.figure()
fig.set(alpha=0.65) #  设置图像透明度，无所谓
df['评分'].hist(bins = 100).plot(kind="bar")#分数的分布
plt.xlim(0,10)
plt.xlabel(u"score")# plots an axis lable
plt.ylabel(u"movie")
plt.title(u"distribution of score")


df.ix[df['上映年份'].isnull(),'上映年份'] = '2010'#处理缺失值，按平均值填充
df.ix[df['评价人数'].isnull(),'评价人数'] = 6319   #处理缺失值，按平均值填充
df.ix[df['评分'].isnull(),'评分'] = 7.2   #处理缺失值，按平均值填充
print(df.info())

fig.set(alpha=0.65)
df['评分'][df['评价人数'] >= 70000].plot(kind='kde',color='steelblue')
df['评分'][df['评价人数'] < 70000][df['评价人数'] >= 7000].plot(kind='kde',color='lightblue')
df['评分'][df['评价人数'] < 7000][df['评价人数'] >=700].plot(kind='kde',color='pink')
df['评分'][df['评价人数'] < 700].plot(kind='kde',color='red')
```

```
plt.legend((u'very famous(>70000 comments)', u'famous(7000-70000 comments)',u'medium(700-7000 comments)',u'low(<700 comments)'),loc='best') # sets our legend for our graph.
plt.xlim(0,10)
plt.xlabel(u"score")# plots an axis lable
plt.ylabel(u"probability")
plt.title(u"people & score")


fig.set(alpha=0.65) # 设置图像透明度，无所谓
plt.scatter(df['评价人数'], df['评分'])
plt.xlabel(u"people")# plots an axis lable
plt.ylabel(u"score")# plots an axis lable
plt.grid(b=True, which='major', axis='y')
plt.legend((u'distribution of comments'),loc='best')

cols = ["评分", "评价人数"]
df[cols].groupby('评分').mean().plot(kind='kde',color='lightblue')
plt.xlabel(u"people")# plots an axis lable
plt.grid(b=True, which='major', axis='y')
plt.legend((u'distribution of score'),loc='best')
plt.show()

df['评分'] = pd.cut(df['评分'], bins=10, labels=False)
fig.set(alpha=0.65) # 设置图像透明度，无所谓
plt.scatter(df['评价人数'], df['评分'])
plt.ylabel(u"movie")                                    # 设定纵坐标名称
plt.xlabel(u"year")# plots an axis lable
plt.grid(b=True, which='major', axis='y')
plt.title(u"time & score")
plt.show()
```

二

①数据采集

```
# -*- coding: utf-8 -*-
import codecs
import csv
import re
import requests
from bs4 import BeautifulSoup
```

```
def getHTML(url):
    r = requests.get(url)
    return r.content

def llist(list1):
    for mm in list1:
        mm = int(mm)
    return mm

def llist2(list2):
    for k in list2:
        k = float(k)
    return k

def parseHTML(html):
    soup = BeautifulSoup(html, 'html.parser')
    genre = soup.title.string
    movie1 = soup.body.find('div', attrs={'id': 'wrapper'})
    movie2 = movie1.find('div', attrs={'class': 'article'})
    movie3 = movie2.find('div', attrs={'class': ''})
    movie4 = movie3.find_all('table', attrs={'width':'100%','class': ''})#非常繁复，有没有再简
化的方法？
    movie_list = []
    for info in movie4:
        name = info.get_text()
        info1 = info.find('a', attrs={'class': ''})
        name1 = info1.get_text()
        if re.search(r'\d+\-\d+', name) != None:
            time1 = re.findall(r'\d{4}\-\d{2}', name)   #  正则提取数字
            year1 = re.findall(r'\d{4}', str(time1))
            year = llist(year1)
        elif re.search(r'\d{4}\-\d{2}', name) == None:
            year = None

        info2 = info.find('div', attrs={'class': 'star clearfix'})
        info3 = info2.get_text()
        if re.match(r'\n\n\d', info3) != None:
            peo1 = re.findall(r'\d{2,6}', info3)
            peo = llist(peo1)
        elif re.match(r'\n\d', info3) == None:
            peo = None
        if re.match(r'\n\n\d', info3) != None:
            score1 = re.findall(r'\d\.\d', info3)
```

```
                score = llist2(score1)   # 正则提出来带[]和引号，要去掉！
            elif re.match(r'\n\d', info3) == None:
                score = None

            movie_list.append([genre,name1,year,score,peo])

        return movie_list   # 修改

def writeCSV(file_name, data_list):
        with codecs.open(file_name, 'w','utf-8') as f:
            writer = csv.writer(f)
            writer.writerow(['genre', "movie","year","score","people"])#write 要有标题就这样
            for data in data_list:
                writer.writerow(data)


data_list = []
try:
        for i in range(205):
            num = str(20*(i))#这里转 str
            url  =  'https://movie.douban.com/tag/%E7%88%B1%E6%83%85?start='+num+'&amp'
#这个 url 的构造规律性太强了！
            html = getHTML(url)
            data_list = data_list + parseHTML(html) # list 拼接
        writeCSV('love.csv',data_list)

except Exception as e:
        print(i)

#
```

②数据预处理

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sp
from scipy.stats import norm
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.neighbors import KNeighborsRegressor
from sklearn import linear_model
```

```
pd.options.mode.chained_assignment = None
import codecs
df = pd.read_csv("score.csv") #读取数据
print(df.shape)
print(df.info())
print(df.describe())
print(df.columns)


df.ix[df.time.isnull(),'time'] = 2010#处理缺失值，按平均值填充
df.ix[df.people.isnull(),'people'] = 3569#处理缺失值，按平均值填充

score = KNeighborsRegressor(n_neighbors=1)#使用 K-means 方法填充分数 #n_neighbors 即 KNN 中的 K 值

# 把数据分为两部分，有缺失的和无缺失的，用无缺失的数据建立模型来判断缺失数据的可能取值
train_score = df[df.score.isnull()==False] #训练集
train_null_score = df[df.score.isnull()==True]
cols = ['time', 'people']
score.fit(train_score[cols], train_score.score)
new_values = score.predict(train_null_score[cols])
train_null_score.ix[:, 'score'] = new_values #填充分数缺失值
score = train_score.append(train_null_score)
df.score = score
print(df.info())
print(score)
```

③画图

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy as sp
from scipy.stats import norm
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
pd.options.mode.chained_assignment = None
import codecs
df = pd.read_csv("score.csv") #读取数据
print(df.shape)
```

```
print(df.info())
print(df.describe())
print(pd.crosstab(df.time, df.score))
fig=plt.figure()

#画图
fig.set(alpha=0.65) # 设置图像透明度，无所谓
df.score.hist(bins = 100).plot(kind="bar")#分数的分布
plt.xlim(0,10)
plt.xlabel(u"score")# plots an axis lable
plt.ylabel(u"movie")
plt.title(u"distribution of score")
plt.show()
plt.savefig('distribution of score.png')     # 保存图片
#了解分数的分布。当然这一点 describe 里也有


fig.set(alpha=0.35) # 设置图像透明度，无所谓
df.time.hist(bins = 102).plot()#分数的分布
plt.xlabel(u"year")# plots an axis lable
plt.title(u"distribution of year") # 标题
plt.ylabel(u"movie")
plt.grid(b=True, which='major', axis='y')
plt.show()
plt.savefig('distribution of year.png')     # 保存图片
#了解电影制造年份的分布。同理 describe 就有




fig.set(alpha=0.65) # 设置图像透明度，无所谓
df.score[df.time >=2013].plot(kind='kde',color='steelblue')
df.score[df.time < 2013][df.time >= 2003].plot(kind='kde',color='lightblue')
df.score[df.time < 2003].plot(kind='kde',color='pink')
plt.xlabel(u"score")# plots an axis lable
plt.ylabel(u"probability")
plt.title(u"time & score")
plt.legend((u'latest(after 2013)', u'medium(2003-2013)',u'classic(before 2003)'),loc='best') #
sets our legend for our graph.
plt.xlim(0,10)
plt.show()
plt.savefig('time & score.png')     # 保存图片

fig.set(alpha=0.65) # 设置图像透明度，无所谓
```

```
aa = df.score[df.time >=2013].value_counts()
bb = df.score[df.time < 2013][df.time >= 2003].value_counts()
cc = df.score[df.time < 2003].value_counts()
df2=pd.DataFrame({u'latest(after 2013)':aa, u'medium(2003-2013)':bb, u'classic(before 2003)':cc})
df2.plot(kind='bar', stacked=True)
plt.title(u"time & score")
plt.xlabel(u"score")
plt.ylabel(u"movie")
plt.show()
plt.savefig('time & score2.png')
#了解年份与分数的关系，一般来说越新越高分
df.boxplot(column='score', by='time')
plt.title(u"distribution of score")
plt.show()
plt.savefig('distribution of score2.png')    # 保存图片
#了解年份与分数的关系，一般来说越新分数的标准差越大，是异方差的（当然也与样本量有关）
#然而可见，1985-2005 这一段时间，整体作品质量都比较上乘


fig.set(alpha=0.65) # 设置图像透明度，无所谓
df.score[df.people >= 70000].plot(kind='kde',color='steelblue')
df.score[df.people < 70000][df.people >= 7000].plot(kind='kde',color='lightblue')
df.score[df.people < 7000][df.people >=700].plot(kind='kde',color='pink')
df.score[df.people < 700].plot(kind='kde',color='red')
plt.legend((u'very famous(>70000 comments)', u'famous(7000-70000 comments)',u'medium(700-7000 comments)',u'low(<700 comments)'),loc='best') # sets our legend for our graph.
plt.xlim(0,10)
plt.xlabel(u"score")# plots an axis lable
plt.ylabel(u"probability")
plt.title(u"people & score")
plt.show()
plt.savefig('people & score.png')    # 保存图片

fig.set(alpha=0.65) # 设置图像透明度，无所谓
aa = df.score[df.people >= 70000].value_counts()
bb = df.score[df.people < 70000][df.people >= 7000].value_counts()
cc = df.score[df.people < 7000][df.people >=700].value_counts()
dd = df.score[df.people < 700].value_counts()
df2=pd.DataFrame({u'very famous(>70000 comments)':aa, u'famous(7000-70000 comments)':bb, u'medium(700-7000 comments)':cc, u'low(<700 comments)':dd})
df2.plot(kind='bar', stacked=True)
```

```
plt.title(u"people & score")
plt.xlabel(u"score")
plt.ylabel(u"movie")
plt.show()
plt.savefig('people & score2.png')    # 保存图片
#了解评价人数与分数的关系，一般来说评价人数与分数成正相关


fig.set(alpha=0.65) # 设置图像透明度，无所谓
df.time[df.people >= 70000].plot(kind='kde',color='steelblue')
df.time[df.people < 70000][df.people >= 7000].plot(kind='kde',color='lightblue')
df.time[df.people < 7000][df.people >=700].plot(kind='kde',color='pink')
df.time[df.people < 700].plot(kind='kde',color='red')
plt.legend((u'very famous(>70000 comments)', u'famous(7000-70000 comments)',u'medium(700-7000 comments)',u'low(<700 comments)'),loc='best') # sets our legend for our graph.
plt.xlim(1910,2030)
plt.xlabel(u"time")# plots an axis lable
plt.ylabel(u"probability")
plt.title(u"people & time")
plt.show()
plt.savefig('people & time.png')    # 保存图片

fig.set(alpha=0.65) # 设置图像透明度，无所谓
aa = df.time[df.people >= 70000].value_counts()
bb = df.time[df.people < 70000][df.people >= 7000].value_counts()
cc = df.time[df.people < 7000][df.people >=700].value_counts()
dd = df.time[df.people < 700].value_counts()
df2=pd.DataFrame({u'very famous(>70000 comments)':aa, u'famous(7000-70000 comments)':bb, u'medium(700-7000 comments)':cc, u'low(<700 comments)':dd})
df2.plot(kind='bar', stacked=True)
plt.title(u"people & time")
plt.xlabel(u"time")
plt.ylabel(u"movie")
plt.show()
plt.savefig('people & time2.png')    # 保存图片
#了解评价人数与年份的关系，一般来说评价人数与分数成正相关


df['score'] = pd.cut(df.score, bins=10, labels=False)
fig.set(alpha=0.65) # 设置图像透明度，无所谓
plt.scatter(df.time, df.score)
plt.ylabel(u"movie")                                # 设定纵坐标名称
plt.xlabel(u"year")# plots an axis lable
```

```
plt.grid(b=True, which='major', axis='y')
plt.title(u"time & score")
plt.show()
plt.savefig('time & score3.png')     # 保存图片
#老片分数一般很集中在 6-9 分的中高分段，新片则两极分化愈发严重。


fig.set(alpha=0.65) # 设置图像透明度，无所谓
plt.scatter(df.people, df.score)
plt.ylabel(u"movie")                                  # 设定纵坐标名称
plt.xlabel(u"people")# plots an axis lable
plt.grid(b=True, which='major', axis='y')
plt.title(u"people & score")
plt.show()
plt.savefig('people & score3.png')     # 保存图片
#越高分的电影评分人数越多（越 famous）
```