爬取并分析猫眼电影票房数据

一、背景

目前电影已经是中国人的一种主要的娱乐休闲方式。特别是今年，有着《流浪地球》和《复仇者联盟4-终局之战》等优秀电影。本篇通过爬取猫眼电影19年数据，分析电影市场。

二、数据准备

- **爬虫环境筛选**

  通过查询发现，猫眼提供了两种方式查看当日票房信息，为专业版票房榜和常规版票房榜。通过对比发现，常规版票房在爬取过程中，存在一定的难度，比如有字体反扒技术，需要解析字体。对历史数据的爬取也不是很友好。因此我们选取更为友好的专业版票房榜。

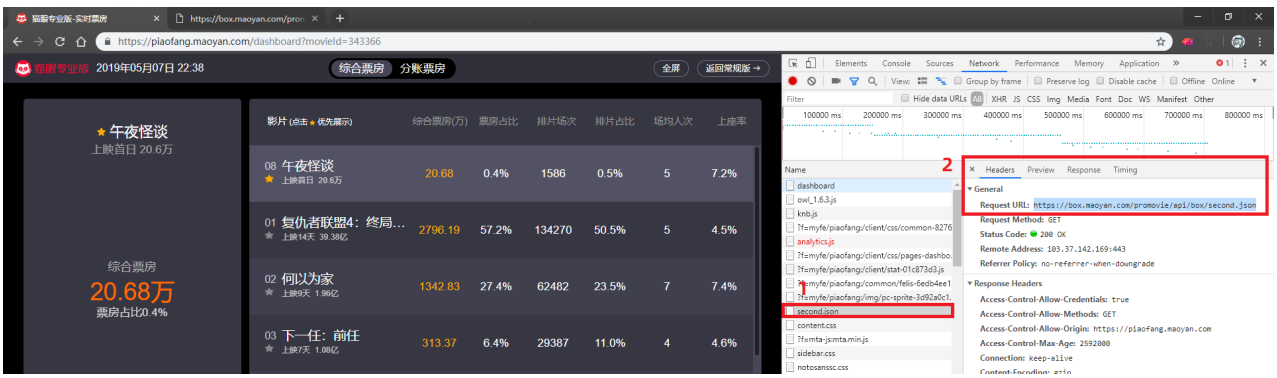
- **爬虫过程分析**

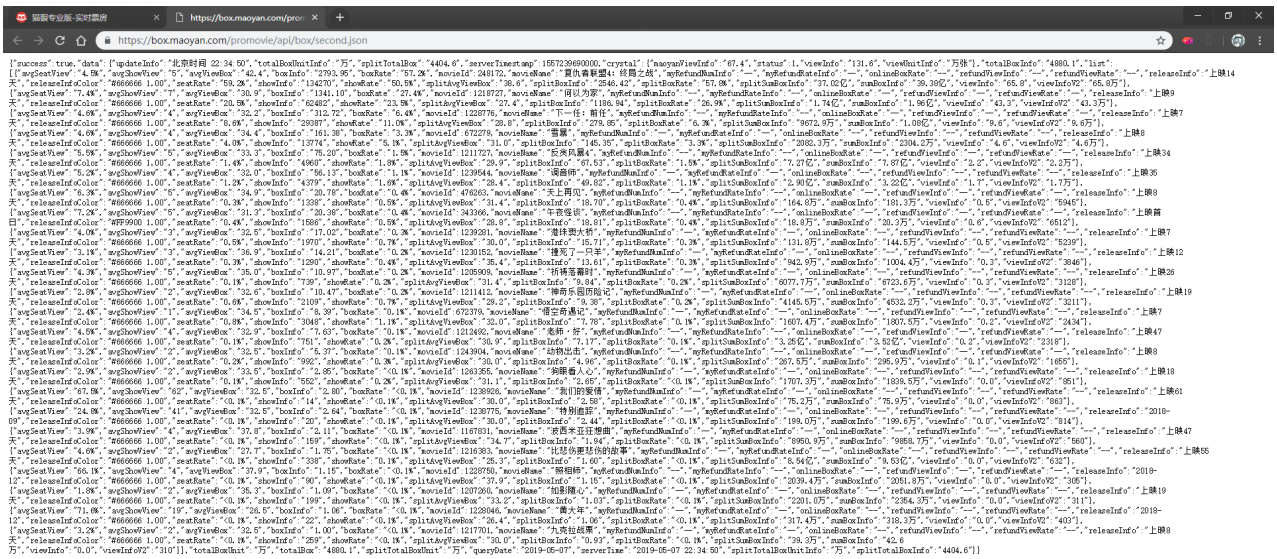
  当前，专业版票房的 URL 为：猫眼专业版 `https://piaofang.maoyan.com/dashboard` 。

  补充详细的 `headers` 信息后，先使用 `request` 模板直接爬取该网站，再通过 `BS4.BeautifulSoup` 解析后发现，票房信息是放在动态 `json` 中的，直接爬取网页无法获取票房信息。

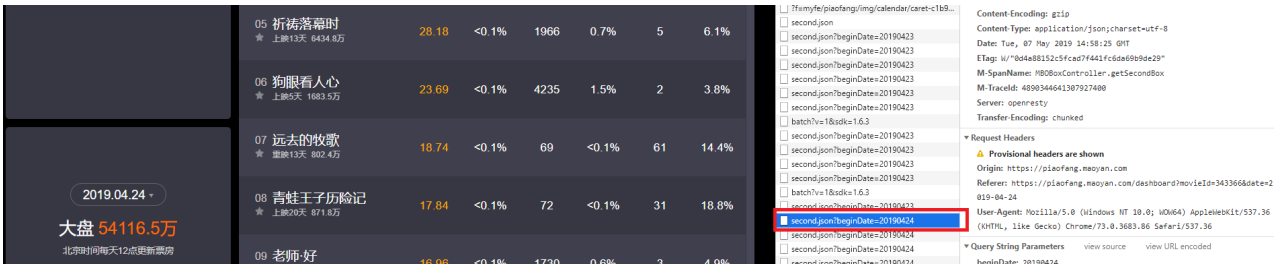
  于是我们通过浏览器检查网页，选择 `Network` ，刷新网页，对当前加载项内容进行分析。

  选择*second.json*这个文件，它的 `response` 中正好包含了票房榜的全部数据。同时他的 `Request Url` 指向的是： `https://box.maoyan.com/promovie/api/box/second.json` 。进入这个地址，果然藏得就是当天的票房信息，还是 `Json` 格式的。图片如下：





由于获取的json文件不包含日期字段，无法获取其他时间的票房信息。我们先退回到票房榜页面。页面左下角有一个时间日期选项，我们先清空下右侧信息（按两次 F12 键），选取到 4 月 23（复联 4 上映时间）。发现有一个包含有日期的*second.json*。



点击`json`文件，跳转到一个地址为https://box.maoyan.com/promovie/api/box/second.json?beginDate=20190423的地址。正好是我们需要的 4 月 23 号信息。我们随便调整下日期，发现数据也随着日期进行变化。此时我们就得到了应该爬取的地址了。

测试一下

```
date = datetime.datetime(2019,4,23)
url = "https://box.maoyan.com/promovie/api/box/second.json?beginDate="
date_url = url + date.strftime('%Y%m%d')
data = requests.get(date_url,headers=headers)
soup = BeautifulSoup(data.text,'lxml')
###注意，原网页是有<html><p>这两个标签的，我们使用bs将其过滤掉
print(soup.p.string)
with open("%s.txt" %(date.strftime('%Y%m%d')),'w+',encoding='utf8') as fp:
    fp.write(str(soup.p.string))
```

查看*20190423.txt*，输出的正好是刚刚看到的`Json`信息。自此我们爬取数据成功。

通过循环我们可以爬取自*20190101*至*20190507*的每日数据。

在这里，为了考虑到多次测试需要，我们使用的是`w+`方式写入文本，这样在写入时，就不用删除之前的文件，新的信息会直接覆盖掉之前的信息。

- **源码展示：**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import requests
import time
import datetime
from bs4 import BeautifulSoup
"""
----------------------------------------------------
   File Name:     DownData.py
   Author:        fynn-PC
   date:          2019/5/6 22:26
   Software:      PyCharm
----------------------------------------------------
"""
'''
此部分用于下载数据
'''


class downData():

    def __init__(self):
        '''
        :param year: 爬虫开始年份
        :param month: 爬虫开始月份
        :param day: 爬虫开始天数
        '''
        self.headers = {
            "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
",
            "Accept-Encoding": "gzip, deflate, br",
            "Accept-Language": "zh-CN,zh;q=0.8",
            "Cache-Control": "max-age=0",
            "Connection": "keep-alive",
            "Upgrade-Insecure-Requests": "1",
            "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36',
        }
        self.year = 2019
        self.month = 1
        self.day = 1

    def run(self):
        year = self.year
        month = self.month
        day = self.day
```

```
        beginDate = datetime.datetime(self.year, self.month, self.day)
        while beginDate.strftime('%Y%m%d') < time.strftime("%Y%m%d",
time.localtime()):
            self._download(beginDate.strftime('%Y%m%d'))
            #休息两秒
            print(f"已完成{beginDate.strftime('%Y%m%d')}文件的下载。")
            time.sleep(2)
            beginDate += datetime.timedelta(days=1)

    def _download(self, date):
        '''
        内部函数
        :param date: 内部参数，爬取当天的日期
        :return: 保存数据
        '''
        url = f"https://box.maoyan.com/promovie/api/box/second.json?beginDate={date}"
        data = requests.get(url, headers=self.headers)
        soup = BeautifulSoup(data.text, 'lxml')
        #将数据保存到与该程序同目录的data文件夹下，并按照时间进行命名
        with open(f"data/{date}.txt", 'w+', encoding='utf8') as fp:
            fp.write(str(soup.p.string))


if __name__ == '__main__':
    year = 2019
    month = 1
    day = 1
    d = downData()
    d.year = year
    d.month = month
    d.day = day
    d.run()
    print("end")
```

三、MySql数据存储与读取

- **读取文本：**

  由于在下载数据环节，我们已经将数据以文本形式存储在本地的*/data*文件夹，所以当前我们需要将数据解析出来，在存储到**SQL**中，以便于我们后续做分析时，可以随用随取。

  通过导入`Json`这个库，我们实现了对文本内容的解析。在解析中，需要注意的是，原文本包含了较多的信息，其中包括有些key的值也是一个字典，我们需要将这些内容清理出来，合并或新建一个新的字典类型，再进行下一步处理。

  如下：

```
fileList = []
for root, dir, files in os.walk(os.path.join(os.getcwd(), 'data')):
    fileList = files
```

```
with open(f'data/{file}', 'r', encoding='utf8') as fp:
    tmp = json.loads(fp.read())
curBox_tmp = tmp['data'].pop('list')
dayBox = pd.DataFrame(curBox_tmp)
dayBox['queryDate'] = file.split('.')[0]
# 由于crystal字段又是一个字典，在这里进行合并
day_total_box_tmp = tmp['data'].pop('crystal')
day_total_box_tmp.update(tmp['data'])
# 这里由于仅是一条数据，所以需要加一个索引
day_total_box = pd.DataFrame(day_total_box_tmp, index=[i])
```

清理完成后，我们生成了两个字典，分别是每天综合票房信息，以及每天上映电影票房信息。

在存储到**SQL**中时，为了便于后续的数据处理方便，我们先将部分会用到的字典再预处理一次，将其格式转化为标准格式，如`float,datetime`等。这个需要注意的是，在转化`onlineViewInfo`字段时会莫名报错。。。

```
# 格式转换
dayBox['queryDate'] = file.split('.')[0]
dayBox['splitBoxInfo'] = dayBox['splitBoxInfo'].astype(np.float)
dayBox['boxInfo'] = dayBox['boxInfo'].astype(np.float)
dayBox['avgViewBox'] = dayBox['avgViewBox'].astype(np.float)
dayBox['showInfo'] = dayBox['showInfo'].astype(np.float)
dayBox['queryDate'] = pd.to_datetime(dayBox['queryDate'])
day_total_box['maoyanViewInfo'] = day_total_box['maoyanViewInfo'].astype(np.float)
day_total_box['viewInfo'] = day_total_box['viewInfo'].astype(np.float)
# 莫名出错
# day_total_box["onlineViewInfo"] = day_total_box["onlineViewInfo"].astype(np.float)
day_total_box['totalBox'] = day_total_box['totalBox'].astype(np.float)
day_total_box['queryDate'] = pd.to_datetime(day_total_box['queryDate'])
```

通过以上，我们就读取并转化了需要存储的字典类型的数据。

- **SQL存储：**

SQL的存储通过`sqlalchemy`类实现的，由于我这里使用的**MySql**且版本为8.0.15，**SQL**与**python**的连接较为复杂，涉及到密码认证方式。其他版本应该会简单一些。可以通过如下方式查看**SQL**版本：

```
mysql> select version();
+-----------+
| version() |
+-----------+
| 8.0.15    |
+-----------+
1 row in set (0.00 sec)
```

python与**SQL**直接配置连接方式基本都一样，先创建连接，之后就是数据层面的增删改查对接了。这里 8.0 以上版本需要的是连接串后面要加一些内容：

```
conn =
create_engine('mysql+mysqlconnector://root:1231@localhost:3306/self_indcharset=utf8&au
th_plugin=mysql_native_password')
```

数据存储，将`DataFrame`写入数据库：

```
pd.io.sql.to_sql(dayBox, 'daybox', conn,if_exists='append',)
pd.io.sql.to_sql(
    day_total_box,
    'day_total_box',
    conn,
    if_exists='append')
```

使用`if_exists=append`是为了后续再更新数据时，仅调整下日期，就可以直接再数据库中添加数据了。若是覆盖，可以改为`replace`。

- **源码展示：**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
-------------------------------------------------
   File Name:     saveSql.py
   Author:        Fynn-mac
   date:          2019-05-08 16:52
   Software:      PyCharm
-------------------------------------------------
"""

import json
import pandas as pd
import os
from sqlalchemy import create_engine
import numpy as np


def run():
    fileList = []
    for root, dir, files in os.walk(os.path.join(os.getcwd(), 'data')):
        fileList = files

    # 数据库存储
    conn = create_engine(
        'mysql+mysqlconnector://root:1231@localhost:3306/self_ind?
charset=utf8&auth_plugin=mysql_native_password')

    # 也可以循环读取，单次写入
    for i, file in enumerate(fileList):
        # 转为pandas进行处理
        with open(f'data/{file}', 'r', encoding='utf8') as fp:
            tmp = json.loads(fp.read())
        curBox_tmp = tmp['data'].pop('list')
        dayBox = pd.DataFrame(curBox_tmp)
        dayBox['queryDate'] = file.split('.')[0]
        # 由于crystal字段又是一个字典，在这里进行合并
        day_total_box_tmp = tmp['data'].pop('crystal')
        day_total_box_tmp.update(tmp['data'])
```

```
        # 这里由于仅是一条数据，所以需要加一个索引
        day_total_box = pd.DataFrame(day_total_box_tmp, index=[i])
        # 格式转换
        dayBox['queryDate'] = file.split('.')[0]
        dayBox['splitBoxInfo'] = dayBox['splitBoxInfo'].astype(np.float)
        dayBox['boxInfo'] = dayBox['boxInfo'].astype(np.float)
        dayBox['avgViewBox'] = dayBox['avgViewBox'].astype(np.float)
        dayBox['showInfo'] = dayBox['showInfo'].astype(np.float)
        dayBox['queryDate'] = pd.to_datetime(dayBox['queryDate'])
        day_total_box['maoyanViewInfo'] = day_total_box['maoyanViewInfo'].astype(
            np.float)
        day_total_box['viewInfo'] = day_total_box['viewInfo'].astype(np.float)
        # 莫名出错
        # day_total_box["onlineViewInfo"] =
day_total_box["onlineViewInfo"].astype(np.float)
        day_total_box['totalBox'] = day_total_box['totalBox'].astype(np.float)
        day_total_box['queryDate'] = pd.to_datetime(day_total_box['queryDate'])

        pd.io.sql.to_sql(dayBox, 'daybox', conn, if_exists='append',)
        pd.io.sql.to_sql(
            day_total_box,
            'day_total_box',
            conn,
            if_exists='append')
    print('sql save end')
```

四、Pandas分析

- **读取数据**

  依然是通过 sqlalchemy 来读取数据。通过 pandas 自带的查询方式获取两张表的全部信息，并且对上一步无法格式转化的内容，进行格式转化，返回两个DataFrame用于分析

```
def readData():
    conn = create_engine(
        'mysql+mysqlconnector://root:1231@localhost:3306/self_ind?
charset=utf8&auth_plugin=mysql_native_password')
    data_total = pd.read_sql_query(
        'select maoyanViewInfo,onlineViewInfo,viewInfo,totalbox,queryDate from
day_total_box',
        con=conn,
        index_col='queryDate')
    # 由于前期在SQL中存储时，没有转化为数字格式，在这里需要转换下
    data_total['onlineViewInfo'] = data_total['onlineViewInfo'].astype(
        np.float)
    sql = "select movieName,sumBoxInfo from daybox "
    data = pd.read_sql_query(sql, con=conn)
    return data_total, data
```

- **数据分析**

我们爬取的数据中，包含了很多指标信息，但猫眼官网仅展示了部分指标，除匹配这部分内容，余下内容基于整理和理解，基本如下：

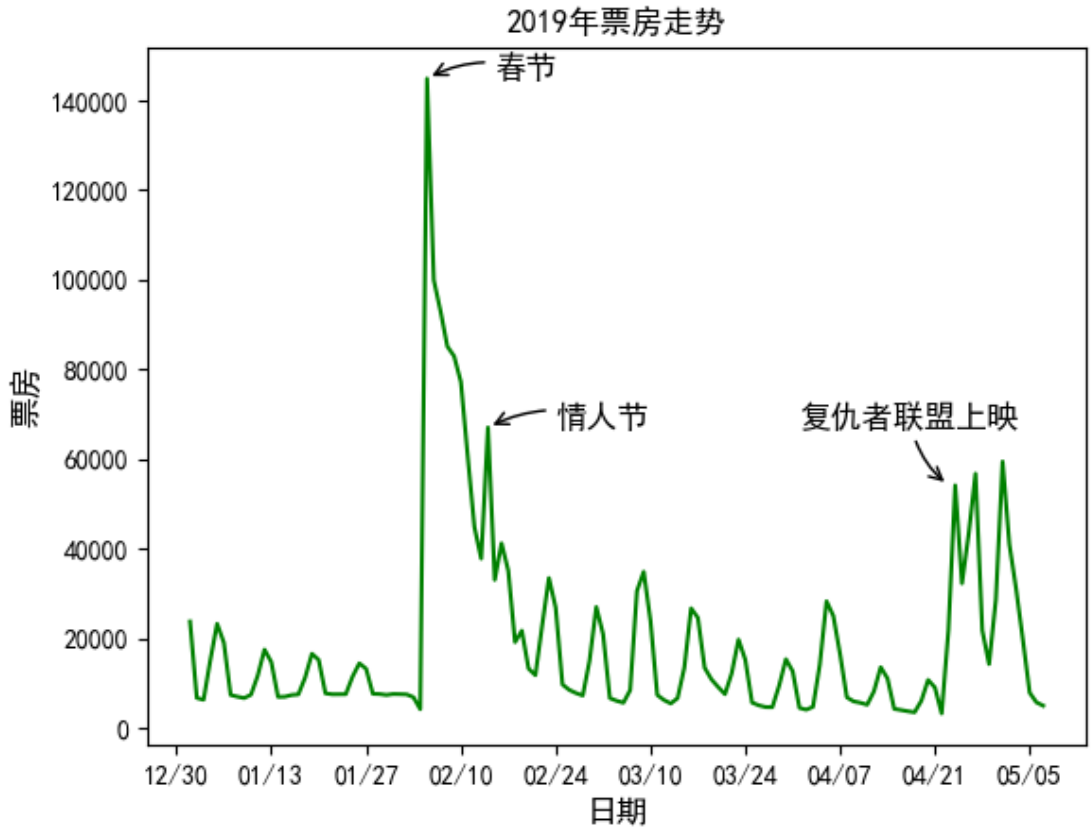
```
# 定义个映射表,页面信信息字段>含义
day_total_box ={
    "splitTotalBoxInfo":"今日实时(分账票房)信息",
    "splitTotalBoxUnitInfo":"分账票房单位信息",
    "totalBox":"今日实时 (综合票房) ",
    "updateInfo":"实时时间信息",
    "list":"信息列表",
    "serverTime":"服务器时间",
    "splitTotalBox":"今日实时(分账票房)",
    "crystal":"crystal",
    "splitTotalBoxUnit":"分账票房单位",
    "totalBoxUnitInfo":"综合票房单位信息",
    "queryDate":"日期",
    "totalBoxUnit":"综合票房单位",
    "serverTimestamp":"服务器的时间戳",
    "totalBoxInfo":"今日实时综合票房信息",
    "maoyanViewInfo":"猫眼观影信息",
    "viewInfo":"全网观影信息"
}
# 定义个映射表,电影条目字段>含义
daybox ={
    "releaseInfoColor":"发布信息颜色",
    "showRate":"排片占比",
    "showInfo":"排片场次",
    "movieId":"电影id号",
    "splitSumBoxInfo":"分账票房总收入",
    "avgSeatView":"上座率",
    "boxInfo":"当日综合票房",
    "splitAvgViewBox":"splitAvgViewBox",
    "boxRate":"票房占比",
    "avgShowView":"场均人次",
    "releaseInfo":"上映天数",
    "splitBoxRate":"分账票房占比",
    "splitBoxInfo":"当日分账票房",
    "avgViewBox":"avgViewBox",
    "seatRate":"seatRate",
    "sumBoxInfo":"综合票房总收入",
    "viewInfo":"viewInfo",
    "movieName":"电影名字",
    "viewInfoV2":"viewInfoV2",
}
```

通过分析上述指标，我们可以：

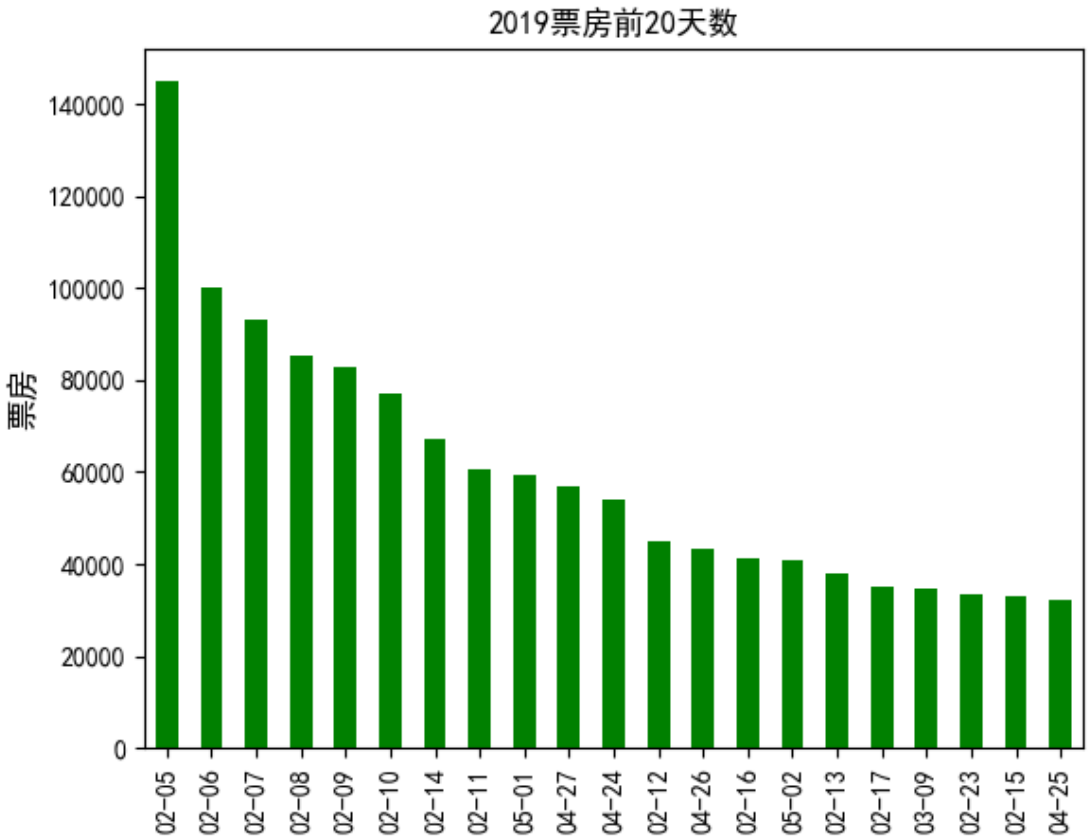
1. 在综合票房中，绘制截至到当前的每日票房趋势图，查看票房走势波动情况，了解电影市场的热点时间。
2. 对总票房进行排序，查看前20的综合票房日期，可以对这些日期进一步进行分析，查看市场热门月份。
3. 通过比对总购票数、在线购票数、猫眼购票数，分析当前的购票渠道
4. 对当前的热门电影进行词云绘制。

- **分析结论**

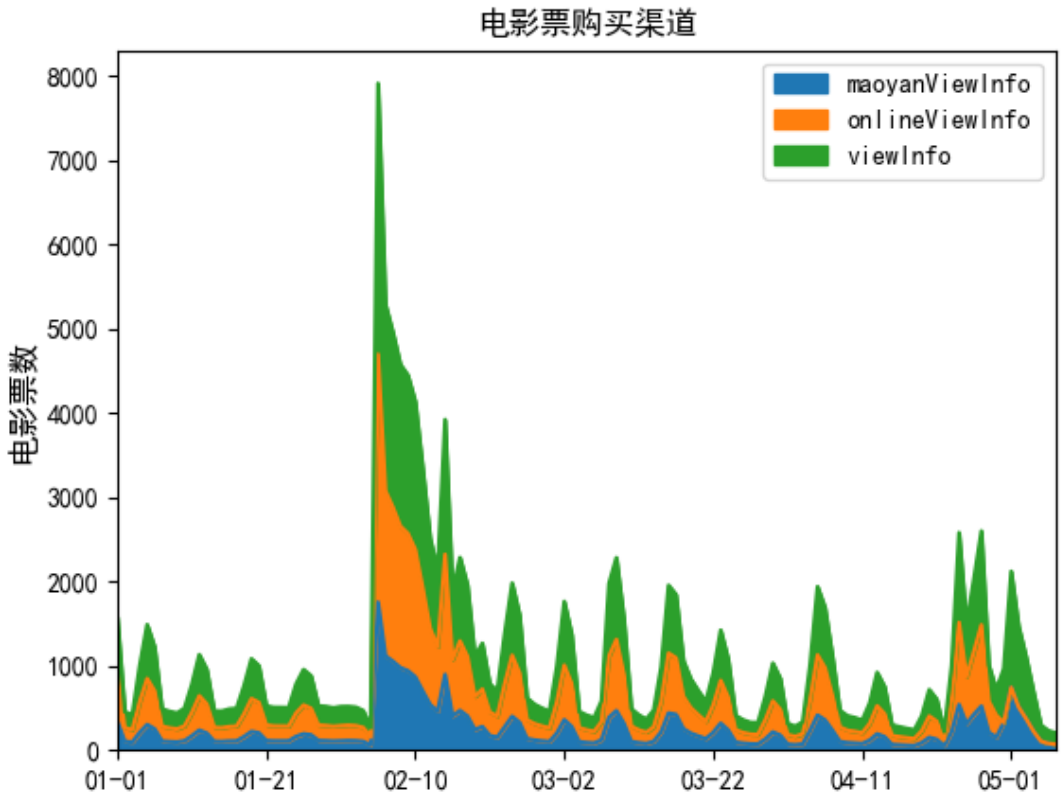
1. 在综合票房走势图中，发现节假日以及热门电影的上映，对电影市场的提升效应非常明显，如春节期间是今年票房市场最火爆时间，情人节也大大提升了市场效应。在漫威热门电影《复仇者联盟4-终局之战》的上映也是一个波峰。比较日常走势，基本是周末电影人数会高于工作日，这也符合大家的996工作现状。

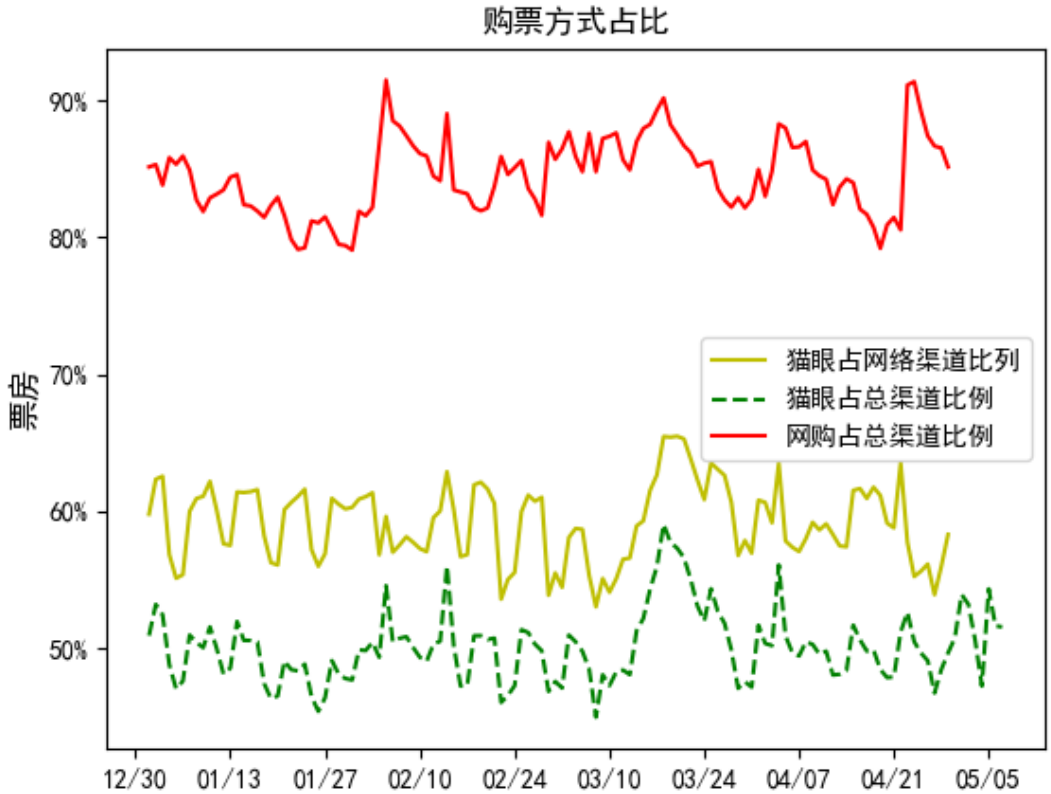


2. 票房榜前20天的排序中，2月占比65%，共计13天，再次验证了上述结论，春节+情人节的组合，是电影市场的黄金时间点。2月5至2月12对应正月初一到初八，2月14是情人节，4月24至4月27是复联4上映的第一个周末。5月1至5月2是劳动节。3月9号在排行榜中是因为，当天是漫威钦定最强英雄惊奇队长的上映第二天，且正好是周六，第一天是周五，引发市场火热，比较尴尬的是，火热的带动效应仅持续了两天。。。另外，4月28号周末没有上榜原因是，当天需要补班。综合来看，电影市场票房多集中在节假日、热门电影上映这些时间段。

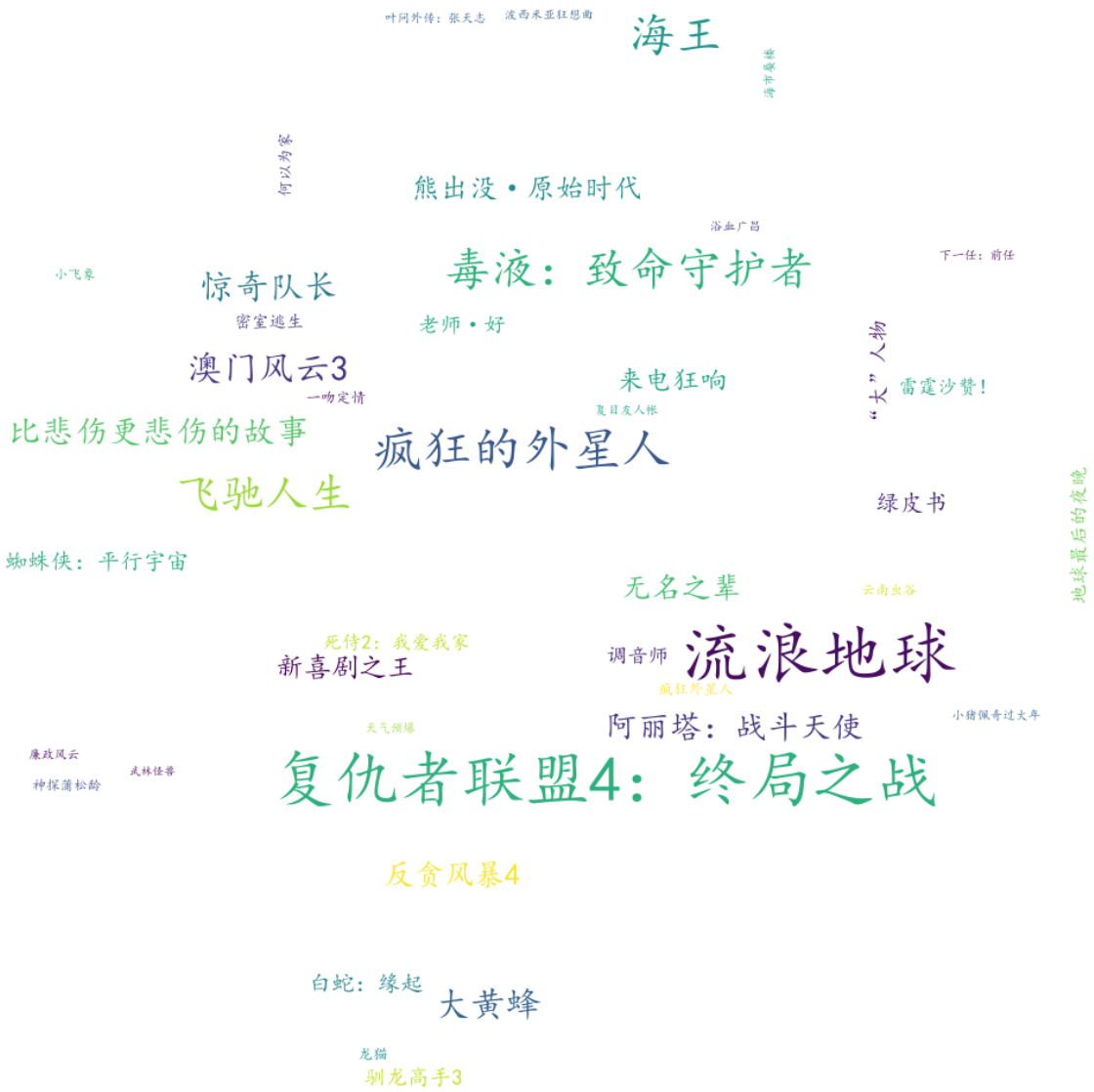


3. 通过对比购票渠道，基本上各渠道的走势基本一致，暂未发现某一时间段下，哪个渠道突然爆发的现象。通过对比猫眼占网络渠道/总渠道的数据，猫眼已经占据了网购渠道的半壁江山，在综合渠道中，也在50%附近浮动。网购已经成为电影票主要的分发渠道。


电影票购买渠道
maoyanViewInfo
onlineViewInfo
viewInfo
电影票数
8000
7000
6000
5000
4000
3000
2000
1000
0
01-01
01-21
02-10
03-02
03-22
04-11
05-01


购票方式占比
猫眼占网络渠道比列
猫眼占总渠道比例
网购占总渠道比例
票房
90%
80%
70%
60%
50%
12/30
01/13
01/27
02/10
02/24
03/10
03/24
04/07
04/21
05/05

4. 热门电影词云绘制。



- **源码展示**

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import numpy as np
import pandas as pd
from matplotlib import ticker as mticker
from matplotlib import dates as mdate
from matplotlib import pyplot as plt
from sqlalchemy import create_engine
from pandas.plotting import register_matplotlib_converters as rmc
from matplotlib.ticker import FuncFormatter
from wordcloud import WordCloud
from PIL import Image

"""
-------------------------------------------------
   File Name:     draw.py
   Author:        fynn-PC
```

```
    date:          2019/5/9 22:51
    Software:      PyCharm
-------------------------------------------------
"""
'''
1、从1月1日到5月7日每日票房曲线
2、从1月1日到5月7日观影人数最多20天
3、总购票数、在线购票、猫眼购票信息对比
4、电影票购买渠道分布
5、词云
'''


def drawWordCloud(data):
    boxInfo = []
    data.drop_duplicates(subset='movieName', keep='last', inplace=True)
    for i in data['sumBoxInfo']:
        if i[-1] == '万':
            boxInfo.append(eval(i[:-1]))
        else:
            boxInfo.append(eval(i[:-1]) * 10000)
    data['boxInfo'] = boxInfo
    tmp = data.sort_values(
        by='boxInfo',
        axis=0,
        ascending=False).drop(
        'sumBoxInfo',
        axis=1)
    t_dict = {}
    for i in range(len(tmp)):
        t_dict[tmp.iloc[i, 0]] = tmp.iloc[i, 1]
    # 加载背景图片
    cloud_mask = np.array(Image.open("img/python.jpg"))
    # 生成wordcloud对象
    wc = WordCloud(background_color="white",
                   font_path="C:\\Windows\\Fonts\\simkai.ttf",
                   mask=cloud_mask,
                   max_words=2000,
                   min_font_size=10,
                   max_font_size=60,
                   random_state=40,)
    wc.generate_from_frequencies(t_dict)
    wc.to_file("img/最热门电影.png")
    plt.imshow(wc, interpolation='bilinear')
    plt.axis('off')
    plt.show()


def drawBuyInfo(data):

    plt.rcParams['font.sans-serif'] = ['SimHei']
    # pandas要求注册mat时使用，消除警告
    rmc(explicit=True)
```

```
    fig = plt.figure()
    font = {'size': 12}
    ax = fig.add_subplot(111)
    ax.set_title('购票方式占比', font)
    ax.set_ylabel('票房', font)
    ax.xaxis.set_major_formatter(mdate.DateFormatter('%m/%d'))
    ax.xaxis.set_major_locator(mticker.MultipleLocator(14))
    ax.plot(
        data.index,
        data['maoyanViewInfo'] /
        data['onlineViewInfo'],
        'y-',
        label='猫眼占网络渠道比列')
    ax.plot(
        data.index,
        data['maoyanViewInfo'] /
        data['viewInfo'],
        'g--',
        label='猫眼占总渠道比例')
    ax.plot(
        data.index,
        data['onlineViewInfo']/
        data['viewInfo'],
        'r-',
        label='网购占总渠道比例')
    plt.legend(loc='best')

    # 用于处理Y轴百分比
    def to_percent(temp, position):
        return '%1.0f' % (100 * temp) + '%'

    plt.gca().yaxis.set_major_formatter(FuncFormatter(to_percent))
    plt.savefig("img/购票方式占比.png")
    plt.show()


def drawViewInfo(data):

    plt.rcParams['font.sans-serif'] = ['SimHei']
    data.index = data.index.strftime('%m-%d')
    tmp = data.drop('totalbox', axis=1)
    font = {'size': 12}
    tmp.plot.area()
    plt.title('电影票购买渠道', font)
    plt.ylabel('电影票数', font)
    plt.savefig("img/电影票购买渠道.png")
    plt.show()


def drawTop20Day(data):
    '''
    绘制票房前20的天数
    :param data:
```

```
    :return: None
    '''
    plt.rcParams['font.sans-serif'] = ['SimHei']
    data.index = data.index.strftime('%m-%d')
    font = {'size': 12}
    data.plot(kind='bar', color='g')
    plt.title('2019票房前20天数', font)
    plt.ylabel('票房', font)
    plt.savefig("img/2019票房前20天数.png")
    plt.show()


def drawBoxInfo(data):
    '''
    绘制2019年票房走势
    :param data:
    :return: None
    '''

    plt.rcParams['font.sans-serif'] = ['SimHei']
    # pandas要求注册mat时使用，消除警告
    rmc(explicit=True)
    fig = plt.figure()
    font = {'size': 12}
    ax = fig.add_subplot(111)
    ax.set_title('2019年票房走势', font)
    ax.set_xlabel('日期', font)
    ax.set_ylabel('票房', font)
    # 设置X轴坐标间距
    # ax.set_xticks(range(0, len(data), 20))
    # ax.set_xticklabels(data.index.strftime('%m-%d')[::20])
    # 设置X轴坐标间距
    ax.xaxis.set_major_formatter(mdate.DateFormatter('%m/%d'))
    ax.xaxis.set_major_locator(mticker.MultipleLocator(14))
    # 根据轴的时间时间属性来设置坐标
    # ax.xaxis.set_major_formatter(mdate.DateFormatter('%m/%d'))
    # ax.set_xticks(pd.date_range('2019-01-01', '2019-05-07', freq='15d'))
    ax.plot(data.index, data['totalbox'], color='g')
    ax.annotate('春节',
                xy=('2019-02-05',
                    data.loc['2019-02-05',
                             'totalbox']),
                xytext=('2019-02-15',
                        data.loc['2019-02-05',
                                 'totalbox']),
                xycoords='data',
                fontsize=12,
                arrowprops=dict(arrowstyle='->',
                                connectionstyle="arc3,rad=.2"))
    ax.annotate('情人节',
                xy=('2019-02-14',
                    data.loc['2019-02-14',
                             'totalbox']),
```

```
                xytext=('2019-02-24',
                        data.loc['2019-02-14',
                                 'totalbox']),
                xycoords='data',
                fontsize=12,
                arrowprops=dict(arrowstyle='->',
                                connectionstyle="arc3,rad=.2"))
    ax.annotate('复仇者联盟上映',
                xy=('2019-04-23',
                    data.loc['2019-04-24',
                             'totalbox']),
                xytext=('2019-04-01',
                        data.loc['2019-02-14',
                                 'totalbox']),
                xycoords='data',
                fontsize=12,
                arrowprops=dict(arrowstyle='->',
                                connectionstyle="arc3,rad=.2"))
    plt.savefig("img/2019年票房走势.png", dip=500)
    plt.show()


def readData():
    conn = create_engine(
        'mysql+mysqlconnector://root:1231@localhost:3306/self_ind?
charset=utf8&auth_plugin=mysql_native_password')
    data_total = pd.read_sql_query(
        'select maoyanViewInfo,onlineViewInfo,viewInfo,totalbox,queryDate from
day_total_box',
        con=conn,
        index_col='queryDate')
    # 由于前期在SQL中存储时，没有转化为数字格式，在这里需要转换下
    data_total['onlineViewInfo'] = data_total['onlineViewInfo'].astype(
        np.float)
    sql = "select movieName,sumBoxInfo from daybox "
    data = pd.read_sql_query(sql, con=conn)
    return data_total, data


def run():
    data_total, data = readData()
    drawBoxInfo(data_total)
    drawTop20Day(data_total['totalbox'].sort_values(ascending=False)[0:21])
    # # 这个函数不能放在最后
    drawBuyInfo(data_total)
    drawViewInfo(data_total)
    drawWordCloud(data)
    print("draw end")
```

五、延展与参考

- 更多的python数据可视化工具
- 词云介绍
- matplotlib绘图