原文链接：https://github.com/Alexander-zhoukai/TMDb-data-analysis-report

TMDb 数据分析报告

目　录

第一章　提出问题

1.1 项目背景

本报告数据来源于 Kaggle 平台上的项目 TMDb（The Movie Database），主要是 1916-2017 年百年间美国电影作品，共有 4803 部。

通过对电影数据的分析，利用可视化的方法，发现电影流行的趋势，找到电影投资的方向，为行业新入局者提供参考建议。

本文的重点在于从不同的角度，用数据可视化的方法来分析。未能面面俱到。

1.2 提出问题

本次数据分析的核心任务是：通过历史电影数据的分析，为行业新入局者提供参考建议。细分为以下几个小问题：

问题 1：电影风格随时间变化的趋势；

问题 2：不同风格电影的收益能力；

问题 3：不同风格电影的受欢迎程度；

问题 4：不同风格电影的平均评分；

问题 5：不同风格电影的平均评价次数；

问题 6：比较 Universal Picture 与 Paramount Picture 两家巨头公司的业绩；

问题 7：原创电影和改变电影的对比；

问题 8：电影票房收入与哪些因素最相关。

1.3 分析思维

数据分析常用思维有细分、对比和溯源。

细分方法有横切、纵切和内切。其中横切是指从各个维度的各个点来分析（如产品、渠道、用户、营销等维度里面的各个指标点）；纵切是指通过漏斗分析、动作轨迹分析或者日志来做分析；内切一般是用 RFM 来深入分析。对比指横切的对比、纵切的对比、目标的对比或者时间上对比。溯源是指通过反复的细分，反复的对比，来确定关键点所在。

本项目采用的思维是细分-横切，从各个维度分析以找到关键信息。

1.4 分析方法和工具

本项目采用数据可视化的方法，来呈现各部分的分析结果，回答问题用数说话、用图说话。数据分析过程中使用 Python 编程语言，数据处理使用 pandas 库、numpy 库，可视化需要 matplotlib 库、seaborn 库。使用以上方法、工具能较好地完成本项目，是适合的方法和工具。

1.5 本项目计划

1 月 2 号，完成第一、二章，前期工作：工具安装调试、项目背景和理解数据。

1 月 3、4 号，完成第三、四章，主要是编写代码：完成数据清洗、数据分析和可视化。

1 月 5 号，完成第五、六章，文字部分：整理项目资料，编写输出文档、存档资料。

第二章　理解数据

在 Kaggle 平台上找到 TMDb 项目，下载 2 个原始数据集：tmdb_5000_movies.txt 和 tmdb_5000_credits.txt，前者存放电影的基本信息，有 20 个字段，后者存放演职人员的信息，有 4 个字段。

表 2.1　原始数据集各字段的含义

tmdb_5000_movies		tmdb_5000_credits	
budget	预算	movie_id	编号
genres	风格	title	主题
homepage	主页	cast	演员
id	编号	crew	职员
keywords	关键词		
original_language	原始语言		
original_title	原始标题		
overview	摘要		
popularity	人数		
production_companies	生产公司		
production_countries	生产国家		
release_date	发行日期		
revenue	票房收入		
runtime	时长		
spoken_languages	语言		
status	状态		
tagline	标签		
title	主题		
vote_average	投票平均得分		
vote_count	投票数量		

2.1 导入包

数据分析及可视化常用库：4+n(4)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import json
import warnings
```

```
warnings.filterwarnings('ignore')
plt.rcParams['font.sans-serif'] = ['SimHei']   # 处理中文乱码
```

2.2 导入数据

```
movies = pd.read_csv('D:/DataAnalysis/cases/TMDb1/tmdb_5000_movies.txt')
credits= pd.read_csv('D:/DataAnalysis/cases/TMDb1/tmdb_5000_credits.txt')
```

2.3 理解数据

边看边观察，为后续的处理做准备。

（1）查看数据维度：shape 属性

```
movies.shape
credits.shape
```

（2）查看数据字段：columns 属性

```
movies.columns
credits.columns
```

（3）查看数据统计信息：describe 方法

```
movies.describe()
credits.describe()
```

（4）查看数据框信息：info 方法

```
movies.info()
credits.info()
full.info()
```

（5）展示数据头：head 方法

```
movies.head()
credits.head()
```

小结：movies 表与 credits 有 2 个重复字段，id 和 title，接下来需要处理；movies 表中 homepage、release_date、runtime 和 tagline 四个字段的数据均有缺失，也需要处理。

第三章　数据清洗

原始数据不适宜直接用来做分析，需要进行一系列的清洗，为后面的分析、可视化做好准备。数据清洗主要分为 3 部分：预处理对数据进行删除、填充和合并；特征提取让数据更规整；特征选取已经开始做数据分析的前一步，将这一部分合并到下一章。

3.1 数据预处理

3.1.1 删除行列

credits 中的字段 title， 与 movies 中的字段 title 重复了，删除前者。

方法一：del 命令

```
#删除 title 列
del credits['title']
#删除 homepage,original_title,overview,spoken_languages,status,tagline,movie_id
del  full('homepage')
del  full('original_title')
del  full('overview')
del  full('spoken_languages')
del  full('status')
del  full('tagline')
del  full('movie_id')
```

方法二：drop 方法

```
#删除 title 列
credits.drop('title', axis = 1, inplace = True)，或者：
credits = credits.drop('title', axis = 1)
#删除 homepage,original_title,overview,spoken_languages,status 等 7 列
full.drop(['homepage', 'original_title', 'overview', 'spoken_languages', 'status', 'tagline', 'movie_id'], axis = 1, inplace = True)
```

3.1.2 填充缺失值

movies 中 release_date 字段的数据缺失 1 条，runtime 字段的数据缺失 2 条，需要补充。

方法一：观察 + 查找 + 填充

```
# 填充 release_date 字段的缺失值
```

```
# 先找出 release_date 字段缺失值所在的行、列位置
# 使用了 isnull()和布尔索引
movies['release_date'].isnull()
movies[movies['release_date'].isnull()]
movies.loc[movies['release_date'].isnull()]
movies.loc[4553, 'release_date'] = '2010-06-01'

# 填充 runtime 字段的缺失值
# 先找出 runtime 字段缺失值所在的行、列位置
full.loc[full['runtime'].isnull()]
full.loc[2656, 'runtime'] = 94
full.loc[4140, 'runtime'] = 240
```

方法二：观察 + 快速填充

```
# 填充 release_date 字段的缺失值
movies['release_date'][movies['release_date'].isnull()] = '2010-06-01'
movies['release_date'].fillna( '2010-06-01')

# 填充 runtime 字段的缺失值
full['runtime'].fillna(94, limit = 1, inplace = True)
full['runtime'].fillna(240, limit = 1, inplace = True)
```

3.1.3 合并表格

方法一：append 方法（纵向合并）

方法二：concat 方法（顶级方法用 pd 调用）

（用 axis 参数来控制合并的横纵方向， axis=1 横向合并占内存最少）

```
full = pd.concat([movies, credits], axis = 1)，或者：
full = pd.concat([movies, credits], axis = 0)，或者：
full = pd.concat([movies, credits])
```

方法三：merge 方法（基类调用，或者实例调用）

```
full = pd.merge(movies, credits, how = 'left', left_on='id', right_on='movie_id')
full1 = movies.merge(credits, how = 'left', left_on='id', right_on='movie_id')
```

方法四：join 方法（只能实例调用）（数据有丢失，本项目该法不可取）

```
full = movies.join(credits, how = 'left', on = 'id')
```

3.2 特征提取

3.2.1 解码 json 字符串

方法一：步骤①②，构造增广矩阵，手撕代码可实现。

方法二：另外，也可采用生成伪变量生成伪矩阵 get_dummies(data, prefix = 'string')，也叫 one-hot-encoding

数据中 genres，keywords，production_companies，production_countries，cast，crew 字段均为 json 类型字符串，要解析出其中的关键信息，需分为两步 apply/map：（1）将 json 类型字符串，转换为字典；（2）取出字典里面的关键信息。

```
# 新建一个列表，存放 json 类型的字段
Cols = ['genres', 'keywords', 'production_companies', 'production_countries', 'cast', 'crew']

# 解码 1，将 json 转换为字典
for col in Cols:
    full[col] = full[col].apply(json.loads)

# 解码 2，将字典内键 name 对应的值取出来（方法 1 手撕代码，方法 2 列表解析表达式）
# 方法 1 手撕代码
def getname(x):
    list = []
    for i in x:
        list.append(i['name'])
    return ','.join(list)
for col in Cols[0:4]:
    full[col] = full[col].apply(getname)
# 方法 2 列表解析表达式
def getname(x):
    list = [i['name'] for i in x]
    return ','.join(list)
for col in Cols[0:4]:
    full[col] = full[col].apply(getname)

# 继续解码 2，将字典内键 charactor 主演对应的值取出来
def getcharacter(x):
    list = [i['character'] for i in x]
    return ','.join(list[0:2])
full['cast'] = full['cast'].apply(getcharacter)
```

```
# 继续解码 2，将字典内键 Director 导演对应的值取出来
def getdirector(x):
    list = [i['name'] for i in x if i['job'] == 'Director']
    return ','.join(list)
full['crew'] = full['crew'].apply(getdirector)
```

3.2.2 去重

去重操作应用了：

数据结构：字符串、集合、列表、

字符串：str, split()

集合：union(), remove(), update(), discard()

提取所有的电影风格，该过程需要去重。从在 genres 字段中提出来的数据会重复。

方法一：常规 4 步。注意 union 是顶级方法。注意去空值。

```
genreset = set()
for x in full['genres'].str.split(','):
    genreset = set().union(genreset, x)
genreset.remove('')
genrelist = list(genreset)
```

方法二：少用的 4 步。update 比 union 简单，discard 与 remove 效果一样。

```
genreset = set()
for x in full['genres'].str.split(','):
    genreset.update(x)
genreset.discard('')
genrelist = list(genreset)
```

3.2.3 数字化

方法 1：五合一，df['name'].str.contains('abc').map(lambda x:1 if x else 0)。

方法 2：快捷 pd.get_dummies(data, prefix = 'aaa')

该过程也叫 one-hot-encoding，涉及到的名词有增广矩阵、伪矩阵、哑矩阵、哑变量、伪变量、数字化、二值化、map(字典)。两个方法的差异在于'name'字段的各个数据有 1 个还是多个伪变量。

本项目数据的 genres 列有多个伪变量，适合用方法一来处理。

```
# 新建数据框 df
genre_df = pd.DataFrame()
# 五合一，进行 one-hot-encoding，为 genre_df 的 20 个列赋值
```

```
    for genre in genrelist:
        genre_df[genre] = full['genres'].str.contains(genre).map(lambda x: 1 if x else 0)
```

3.2.4 类型转换

重点提一下时间类型的转换，带格式化的。

数据中 release_date 字段是字符串类型的，先转换为日期类型，再取年份，得整型。

```
    full['release_date'] = pd.to_datetime(full['release_date'], format = '%Y-%m-%d').dt.year
```

3.2.5 重命名列

对处理过的字段 release_date, cast 和 crew，进行重命名。

```
    name_dict = {'release_date': 'year', 'cast': 'actor', 'crew': 'director'}
    full.rename(columns = name_dict, inplace = True)
```

3.3 特征选取

在分析每个小问题之前，有一个重要的步骤，是通过选取特征构造出合适的数据框，以便高效地进行分析并输出可视化图形。

在解决前面提出的若干问题过程中，会频繁地构造数据框，这里有一个小窍门就是：在分析每一个小问题时，选取要分析的数据列，忽略与本问题无关的数据列，或者再添加特定序列，从而构造出一个新数据框，而不是在原始数据上做分析动作，否则后面思维越来越混乱，数据混淆在一起了。需要注意的是，数据复杂度有三个层次，依次是 DataFrame、Series、List/Dict/Index，在构造数据框的时候参考这三个层次，尽量避免构造复杂度高的数据。

3.3.1 构造 Series

方法 1：在原始数据框中**截取**；

方法 2：使用 Series 定义，List/Dict-->Series。

3.3.2 构造 DataFrame

方法 1：在原始数据框中**截取**；

方法 2：使用 DataFrame 定义，Series/List/Dict/Index-->DataFrame;

方法 3：**合并**；

方法 3：**计算**。

3.4 小结数据清洗报告

在数据清洗这一部分，删除了 title、homepage,original_title 等 8 列，填充了 runtime, release_date 列的缺失值，合并了 movies 和 credits 两个表格，对 6 个 json 类数据进行解码提出关键信息，对 genres 列做了 one-hot 编码，对 release_date 做了类型转换，并对 3 个列进行了重命名，最终得到的数据框 full，4308 行，16 列，数据框 genre_df，4308 行，20 列。

第四章　数据分析及可视化

对 1.2 中的九个问题逐个分析。

数据可视化是对每一个问题，每一个特定的数据框进行可视化，发现数据背后的规律和真相。常用图形有散点图、柱状图、直方图、折线图、饼图、箱线图和词云图；较为常用的图形有：小提琴图。

4.1 电影风格随时间变化的趋势

4.1.1 构造数据框

```
#1、构造数据框（截取+合并）
genre_df['year'] = full['year']
# 各种类型电影的数量随时间变化的趋势 genre_by_year
genre_by_year = genre_df.groupby('year').sum()
# 各种类型电影的总数 genreSum_by_year
# 升序
genreSum_by_year = genre_by_year.sum().sort_values()
# 降序
genreSum_by_year = genre_by_year.sum().sort_values(ascending = False)
```

4.1.2 可视化

```
# 可视化 genreSum_by_year
from pylab import rcParams
params =  {'legend.fontsize': 12,
'legend.handlelength': 10}
rcParams.update(params)


fig = plt.figure(figsize=(20, 5))
ax = plt.subplot(1, 1, 1)
ax = genreSum_by_year.plot(kind = 'bar')
plt.title('Film genre by year', fontsize = 18)
plt.xlabel('genre', fontsize = 18)
plt.ylabel('amount', fontsize = 18)
fig.savefig('Film genre by year.png')


# 可视化 genre_by_year
```

```
from pylab import rcParams
params =  {'legend.fontsize': 10,
'legend.handlelength': 3}
rcParams.update(params)

# genre_by_year = genre_by_year.loc[1960:2020, :]


fig = plt.figure(figsize=(20, 12))
plt.xlabel('Year', fontsize = 10)
plt.ylabel('Amount', fontsize = 10)
plt.xticks(range(1920, 2030, 5))
plt.title('Film amount by year', fontsize = 10)
plt.plot(genre_by_year)
plt.legend(genre_by_year, loc = 'best')
# plt.legend(genre_by_year, loc = 'best', ncol = 2)
fig.savefig('Film amount by year_8.png')
 # genre_by_year.plot() #理解为默认设置，不同于上面设定了各个参数
```

4.1.3 分析结果

（1）从上世纪 90 年代开始，整个电影市场各种类型的电影数量呈现爆发式增长；

（2）其中 Drama、Comedy、Thriller、Romance、Adventure 这五类类电影增长最快。

4.2 不同风格电影的收益能力

4.2.1 构造数据框

```
# 构造数据框(定义+合并+截取+计算)
full['profit'] = full['revenue'] - full['budget']
profit_df = pd.DataFrame()
profit_df = pd.concat([genre_df.iloc[:, :-1], full['profit']], axis = 1)
# 各种类型电影的收益 profit_by_genre
# 构造 Series，保存 profit，其 index 为电影类型 genrelist。（定义+计算）
# 此处的计算是重点，要理解数据框结构，掌握计算逻辑，灵活运用。
profit_by_genre = pd.Series(index = genrelist)
for gen in genrelist:
profit_by_genre.loc[gen] = profit_df.loc[:, [gen,
'profit']].groupby(gen).sum().loc[1, 'profit']
# 排序，升序
profit_by_genre = profit_by_genre.sort_values()
```

4.2.2 可视化

```
    fig = plt.figure(figsize=(20, 12))
    plt.xlabel('Profit', fontsize = 12)
    plt.ylabel('Genre', fontsize = 12)
    plt.title('Profit by Genre', fontsize = 12)
    profit_by_genre.plot(kind = 'barh')
    fig.savefig('Profit by Genre_1.png')

    # 构造分析 budget 的数据框并可视化
    budget_df = pd.DataFrame()
    budget_df = pd.concat([genre_df.iloc[:, :-1], full['budget']], axis = 1)
    budget_by_genre = pd.Series(index = genrelist)
    for gen in genrelist:
    budget_by_genre.loc[gen] = budget_df.loc[:, [gen,
'budget']].groupby(gen).sum().loc[1, 'budget']
    budget_by_genre = budget_by_genre.sort_values()

    fig = plt.figure(figsize=(20, 12))
    plt.xlabel('Budget', fontsize = 15)
    plt.ylabel('Genre', fontsize = 15)
    plt.title('Budget by Genre', fontsize = 15)
    profit_by_genre.plot(kind = 'barh')
    fig.savefig('Budget by Genre_1.png')
```

4.2.3 分析结果

可以看出 Adventure、Action、Comedy、Drama 和 Thriller 电影收益最高。

4.3 不同风格电影的受欢迎程度

4.3.1 构造数据框 popu_by_genre

```
    popu_df = pd.DataFrame()
    popu_df = pd.concat([genre_df.iloc[:, :-1], full['popularity']], axis = 1)
    popu_by_genre = pd.Series(index = genrelist)
    for gen in genrelist:
    popu_by_genre.loc[gen] = popu_df.loc[:, [gen,
'popularity']].groupby(gen).mean().loc[1, 'popularity']
    popu_by_genre.sort_values(inplace=True)
```

4.3.2 可视化

```
fig = plt.figure(figsize=(20, 12))
plt.xlabel('Mean of popularity', fontsize = 15)
plt.ylabel('Genre', fontsize = 15)
plt.title('Mean of popularity', fontsize = 15)
popu_by_genre.plot(kind = 'barh')
fig.savefig('popularity_by_genre_1.png')
```

4.3.3 分析结果

可以看出，Adventure、Animation 最受欢迎

4.4 不同风格电影的平均评分

4.4.1 构造数据框 vote_avg_by_genre

```
vote_avg_df = pd.DataFrame()
vote_avg_df = pd.concat([genre_df.iloc[:, :-1], full['vote_average']], axis = 1)
vote_avg_by_genre = pd.Series(index = genrelist)
for gen in genrelist:
vote_avg_by_genre.loc[gen] = vote_avg_df.loc[:, [gen,
'vote_average']].groupby(gen).mean().loc[1, 'vote_average']
vote_avg_by_genre.sort_values(inplace=True) #排序、升序

full['popularity'].corr(full['vote_average']) #相关性值为： 0.27
# 可以看出，电影的平均受欢迎程度与平均评分的相关性很低
```

4.4.2 可视化

```
fig = plt.figure(figsize=(20, 12))
plt.xlabel('vote_average', fontsize = 15)
plt.ylabel('Genre', fontsize = 15)
plt.xlim(5, 7, 0.5)
# plt.xticks(range(5, 7, 1)) # 给效果不够好。
plt.title('vote avg by genre', fontsize = 15)
vote_avg_by_genre.plot(kind = 'barh')
fig.savefig('vote_avg_by_genre_3.png')
```

4.4.3 分析结果

不同类型电影的平均评分，数据很接近。没有显著的差异，最低值与最高值的差距不到 1

```
# 另外，也可以可视化全部平均分，用频率分布直方图 sns.distplot(data, bins=)
fig = plt.figure(figsize=(20, 12))
plt.xlabel('vote_average', fontsize = 12)
```

```
    plt.ylabel('distributio of vote_avg', fontsize = 12)
    plt.xticks(range(11))
    plt.title('vote avg by genre', fontsize = 12)
    sns.distplot(full['vote_average'], bins = 30)
    fig.savefig('distributio of vote_avg.png')
    # 分析结果
    # 不同类型电影的平均评分，在 5-8 之间
```

4.5 不同类型电影的平均评分次数

4.5.1 构造数据框 vote_count_df、vote_count_avg_by_genre

```
    vote_count_df = pd.DataFrame()
    vote_count_df = pd.concat([genre_df.iloc[:, :-1], full['vote_count']], axis = 1)
    vote_count_avg_by_genre = pd.Series(index = genrelist)
    for gen in genrelist:
    vote_count_avg_by_genre.loc[gen] = vote_count_df.loc[:, [gen,
'vote_count']].groupby(gen).mean().loc[1, 'vote_count']

    vote_avg_by_genre.sort_values(inplace=True)
```

4.5.2 可视化

```
    fig = plt.figure(figsize=(20, 12))
    plt.xlabel('amount', fontsize = 15)
    plt.ylabel('Genre', fontsize = 15)
    plt.title('vote_count_avg_by_genre', fontsize = 15)
    vote_count_avg_by_genre.plot(kind = 'barh')
    fig.savefig('vote_count_avg_by_genre_1.png')
```

4.5.3 分析结果

可以看出，Adventure、Science Fiction 两类电影获得的评价平均次数是最多的。

4.6 比较 Universal Picture 与 Paramount Picture 两家巨头公司的业绩

4.6.1 构造数据框

```
    # 构造两公司业绩数据框 revenue_by_company
    # 公司列表 company
    company_list = ['Paramount Pictures', 'Universal Pictures']
    company_df = pd.DataFrame()
    for company in company_list:
    company_df[company] =
full['production_companies'].str.contains(company).apply(lambda x: 1 if x else 0)
```

```
    # company_df = pd.merge([company_df, genre_df.loc[:, :-1], full['revenue']], axis
= 1)
    company_df = pd.concat([company_df, full['revenue']], axis = 1)

    revenue_by_company = pd.Series(index = company_list)
    for company in company_list:
    revenue_by_company.loc[company] = company_df.loc[:, [company,
'revenue']].groupby(company).sum().loc[1, 'revenue']
```

4.6.2 可视化

```
    fig = plt.figure(figsize=(20, 12))
    plt.xlabel('amount', fontsize = 15)
    plt.ylabel('Genre', fontsize = 15)
    plt.title('Paramount vs Universal ', fontsize = 15)
    revenue_by_company.plot(kind = 'barh')
    fig.savefig('revenue_by_company_1.png')
```

4.6.3 分析结果

Universal Pictures 公司的票房收入高于 Paramount Pictures 公司

4.7 原创电影和改变电影的对比

该问题继续细分为原创电影和改编电影数量的对比、利润的对比

4.7.1 构造数据框

```
    # 原创电影和改编电影数量的对比
    # 原创的电影：original - based on = false
    # 改编来的电影：recompose - based on = true

    original_recompose_list = ['original', 'recompose']
    original_recompose_df = pd.DataFrame()
    original_recompose_df['type'] = full['keywords'].str.contains('based
on').apply(lambda x: 1 if x else 0)

    # original_vs_recompose = pd.DataFrame(index = original_recompose_list, columns =
['count', 'budget', 'revenue', 'profit'])
    original_vs_recompose = pd.Series(index = original_recompose_list )
    original_vs_recompose['original'] =
original_recompose_df.groupby('type').type.count().loc[0]
```

```
    original_vs_recompose['recompose'] =
original_recompose_df.groupby('type').type.count().loc[1]
```

4.7.2 可视化

```
    # 可视化
    fig = plt.figure(figsize=(10, 8))
    plt.xlabel('company', fontsize = 10)
    plt.ylabel('count', fontsize = 10)
    plt.title('Paramount vs Universal ', fontsize = 10)
    original_vs_recompose.plot(kind = 'bar')
    fig.savefig('original_vs_recompose_1.png')
```

4.7.3 分析结果

```
    # 原创电影的数量远多于改编电影

    # 补充
    # 原创电影和改变电影利润的对比
    # 构造数据框
    prof_original_recompose = pd.Series(index = original_recompose_list)
    prof_original_recompose['original'] =
original_recompose_df.groupby('type').profit.sum().loc[0]
    prof_original_recompose['recompose'] =
original_recompose_df.groupby('type').profit.sum().loc[1]
    # 可视化
    fig = plt.figure(figsize=(10, 8))
    plt.xlabel('company', fontsize = 10)
    plt.ylabel('count', fontsize = 10)
    plt.title('profit Paramount vs Universal ', fontsize = 10)
    prof_original_recompose.plot(kind = 'bar')
    fig.savefig('profit Paramount vs Universal_1.png')
    # 分析结果
    # 原创电影的利润远远多于改编电影
```

4.8 电影票房收入与哪些因素最相关

4.8.1 构造数据框

```
    full[['budget', 'popularity', 'revenue', 'runtime', 'vote_average',
'vote_count']].corr()
```

```
    full[['budget', 'popularity', 'revenue', 'runtime', 'vote_average',
'vote_count']].corr().iloc[2]
    # 票房收入与预算、受欢迎程度、评价次数三个指标相关性较强
    revenue_df = full[['popularity', 'vote_count', 'budget', 'revenue']]
```

4.8.2 可视化

```
    # 三个：散点图+线性回归线
    fig = plt.figure(figsize = (15, 5))

    ax1 = plt.subplot(1, 3, 1)
    ax1 = sns.regplot(x='popularity', y='revenue', data = revenue_df)
    ax1.text(400, 3e9, 'r=0.64', fontsize=12)
    plt.xlabel('popularity', fontsize=12)
    plt.ylabel('revenue', fontsize=12)
    plt.title('revenue by popularity', fontsize=15)

    ax2 = plt.subplot(1, 3, 2)
    ax2 = sns.regplot(x='vote_count', y='revenue', data = revenue_df, color='g')
    ax2.text(5800, 2.1e9, 'r=0.78', fontsize=12)
    plt.xlabel('vote_count', fontsize=12)
    plt.ylabel('revenue', fontsize=12)
    plt.title('revenue by vote_count', fontsize=15)

    ax3 = plt.subplot(1, 3, 3)
    ax3 = sns.regplot(x='budget', y='revenue', data = revenue_df, color='r')
    ax3.text(1.6e8, 2.1e9, 'r=0.73', fontsize=12)
    plt.xlabel('budget', fontsize=12)
    plt.ylabel('revenue', fontsize=12)
    plt.title('revenue by budget', fontsize=15)

    fig.savefig('revenue.png')
```

4.8.3 分析结果

票房收入与预算、受欢迎程度、评价次数三个指标相关性较强。

4.9 分析结论

（1）从上世纪 90 年代开始，整个电影市场各种类型的电影数量呈现爆发式增长；

（2）其中 Drama、Comedy、Thriller、Romance、Adventure 这五类类电影增长最快；

（3）可以看出 Adventure、Action、Comedy、Drama 和 Thriller 电影收益最高；

（4）不同类型电影的平均评分，数据很接近，没有显著的差异，最低值与最高值的差距不到 1， 不同类型电影的平均评分，在 5-8 之间。

（5）Adventure、Science Fiction 两类电影获得的评价平均次数是最多的；

（6）Universal Pictures 公司的票房收入高于 Paramount Pictures 公司；

（7）原创电影的数量、利润远多于改编电影；

（8）票房收入与预算、受欢迎程度、评价次数三个指标相关性较强。

第五章　项目回顾与总结

本项目是我在学习用 Python 进行数据分析的过程中，做的一个练习项目，按照最常见的典型步骤——提出问题、理解数据、数据清洗、数据分析及可视化、项目报告——对 TMDb 做数据分析。

数据集来源于 Kaggle 平台上的经典项目 TMDb（The Movie Database），数据集共两个文档：tmdb_5000_movies 和 tmdb_5000_credits。由于现在 Kaggle 官方网站无法注册，导致无法下载数据集，所以本项目数据集并不是在 Kaggle 上下载的，而是查阅了网上很多对 TMDb 进行数据分析文章的附件里下载的，两个数据集都是 txt 格式的。

对这些数据进行分析的目的是发现电影流行的趋势，找到电影投资的方向，为行业新入局者提供参考建议。分析思路采用了细分-横切，从各个维度分析得到关键信息，分析方法采用了可视化。使用的主要软件工具有 Python 编程语言、pandas 数据分析库。

在数据清洗过程中，尽量采用了多种方法来完成，体会了各自的差异和优劣，并加深印象，重难点在特征提取这一节，灵活运用各种方法才能让脚本更高效简洁。

数据分析及可视化的过程，重点在与构造合适的数据框，涉及到 pandas 中重要的分组和聚合。构造数据框的操作也是多样灵活的，需要多多联系、思考和总结。可视化操作按照一套基本固定的程序来实现就好。

本次数据分析项目所做的工作，还存在着不足的地方，下一步需要继续补充并掌握以下几个方面的内容：

（1）对各种常用分析思维做综述（介绍、特点、对比）；

（2）对各种常用方法和工具做综述（介绍、特点、对比）；

（3）对数据清洗的常用方法、函数，要进一步熟练并掌握；

（4）对于有次坐标轴的图形如何调用函数、设置参数。

（5）使用词云图