# 项目名称：TMDb电影数据分析

## 项目目的：

通过对TMDb的电影数据进行分析，得出电影相关问题的答案

## 项目过程：

1、观察原始数据并提出问题 2、载入数据 3、评估数据，包括 每个数据集中的样本数 每个数据集中的列数 每个数据集中重复的行数 列的数据类型 具有缺失值的特征 每个数据集中特征的非空唯一值的数量 这些唯一值都是什么，以及每个的计数 4、整理数据 5、数据清理 6、根据问题进行分析 7、可视化数据 8、整理数据图 9、得出结论

In [33]:

```
#导入数据准备进行分析
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

df = pd.read_csv('tmdb-movies.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
id                      10866 non-null int64
imdb_id                 10856 non-null object
popularity              10866 non-null float64
budget                  10866 non-null int64
revenue                 10866 non-null int64
original_title          10866 non-null object
cast                    10790 non-null object
homepage                2936 non-null object
director                10822 non-null object
tagline                 8042 non-null object
keywords                9373 non-null object
overview                10862 non-null object
runtime                 10866 non-null int64
genres                  10843 non-null object
production_companies    9836 non-null object
release_date            10866 non-null object
vote_count              10866 non-null int64
vote_average            10866 non-null float64
release_year            10866 non-null int64
budget_adj              10866 non-null float64
revenue_adj             10866 non-null float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

In [34]:

```
df.head()
```

Out[34]:

| | id | imdb_id | popularity | budget | revenue | original_title | cast | |
|---|---|---|---|---|---|---|---|---|
| **0** | 135397 | tt0369610 | 32.985763 | 150000000 | 1513528810 | Jurassic World | Chris Pratt\|Bryce Dallas Howard\|Irrfan Khan\|Vi... | |
| **1** | 76341 | tt1392190 | 28.419936 | 150000000 | 378436354 | Mad Max: Fury Road | Tom Hardy\|Charlize Theron\|Hugh Keays-Byrne\|Nic... | |
| **2** | 262500 | tt2908446 | 13.112507 | 110000000 | 295238201 | Insurgent | Shailene Woodley\|Theo James\|Kate Winslet\|Ansel... | http:// |
| **3** | 140607 | tt2488496 | 11.173104 | 200000000 | 2068178225 | Star Wars: The Force Awakens | Harrison Ford\|Mark Hamill\|Carrie Fisher\|Adam D... | |
| **4** | 168259 | tt2820852 | 9.335014 | 190000000 | 1506249360 | Furious 7 | Vin Diesel\|Paul Walker\|Jason Statham\|Michelle ... | |

5 rows × 21 columns

提问：1、如果投资预算不大，有可能拍出票房不错的电影吗？2、哪一年的电影的质量最高？电影的质量是上升还是在下降？3、评分越高的电影是否越受欢迎（播放量大）

根据需要研究的问题，得知不需要的数据

为"cast"、"keywords"、"homepage"、"director"、"tagline"、"overview"、"genres"、"production_companies

并且id与imdb_id并不能在分析过程中起到标示的作用，因此首先要将这几个数据删掉。

In [35]:

```
#删掉不需要的数据
df.drop(['id' ,'imdb_id' , 'cast' , 'keywords' , 'homepage' , 'director' , 'tagli
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 11 columns):
popularity        10866 non-null float64
budget            10866 non-null int64
revenue           10866 non-null int64
original_title    10866 non-null object
runtime           10866 non-null int64
release_date      10866 non-null object
vote_count        10866 non-null int64
vote_average      10866 non-null float64
release_year      10866 non-null int64
budget_adj        10866 non-null float64
revenue_adj       10866 non-null float64
dtypes: float64(4), int64(5), object(2)
memory usage: 933.9+ KB
```

可以看到新的数据表中没有缺失值。考虑到本数据表内容有预算、收入、时间、标签、播放量等可重复出现的数据，因此在删除重复行之后只对电影名称进行查重与过滤。

In [36]:

```
#找到数据中的重复数据个数
df['original_title'].duplicated().sum()
```

Out[36]:

```
295
```

重建删除重复行之后的表格

In [37]:

```
#删除重复数据
df.drop_duplicates('original_title' , inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10571 entries, 0 to 10865
Data columns (total 11 columns):
popularity        10571 non-null float64
budget            10571 non-null int64
revenue           10571 non-null int64
original_title    10571 non-null object
runtime           10571 non-null int64
release_date      10571 non-null object
vote_count        10571 non-null int64
vote_average      10571 non-null float64
release_year      10571 non-null int64
budget_adj        10571 non-null float64
revenue_adj       10571 non-null float64
dtypes: float64(4), int64(5), object(2)
memory usage: 991.0+ KB
```

问题1:投资预算与收入是否相关？此问题将通过对budget_adj和revenue_adj两组数据的研究进行分析。

In [38]:

```
#找出两组数据的最大值
df['budget_adj'].max()
```

Out[38]:

```
425000000.0
```

In [39]:

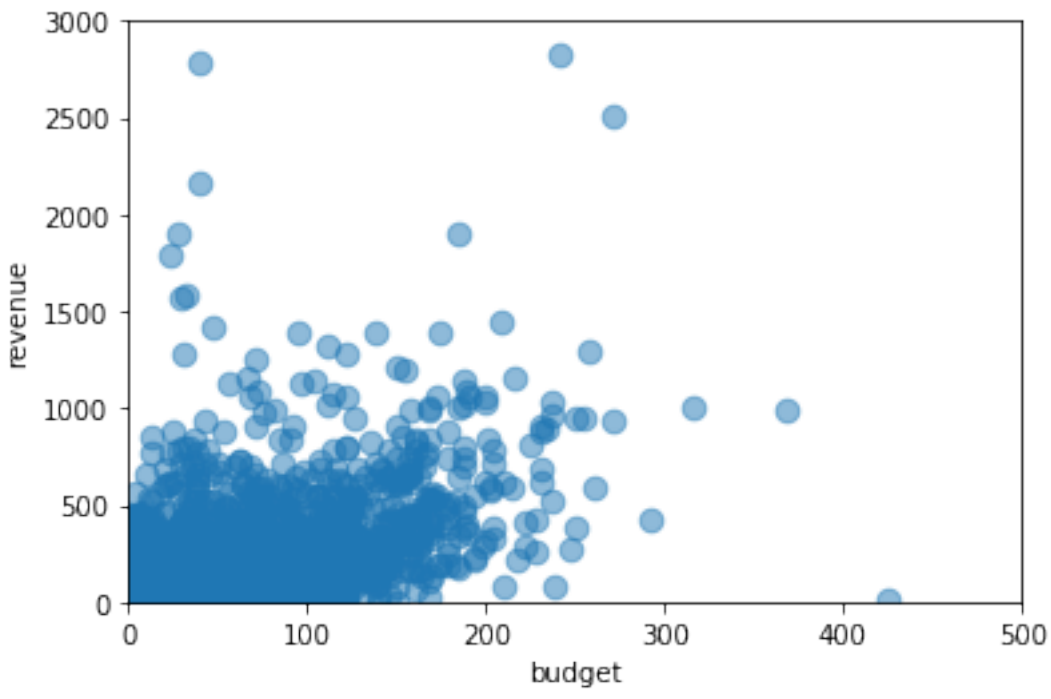
```
df['revenue_adj'].max()
```

Out[39]:

```
2827123750.41189
```

In [40]:

```
#根据散点图找到两组数据的相关性。这里由于每个数据的数字太大，因此将小数点向左移动六位之后再计算
%matplotlib inline
x = (df['budget_adj'] / 1000000).tolist()
y = (df['revenue_adj'] / 1000000).tolist()
plt.scatter(x , y , s = 75 , alpha = 0.5)
plt.xlim((0 , 500))
plt.ylim(0 , 3000)
plt.xlabel('budget')
plt.ylabel('revenue')
plt.show()
```




可以从图中看到两组数据有一定的相关性，但相关性并不强。也就是说明，即使投资预算不大，也有可能排出收入很高的电影。

问题2:哪一年的电影平质量最高？这里我们根据评分来判断电影质量。根据烂番茄与豆瓣的评分机制大概可以知道，评分比票房更能反映出电影的质量。

In [59]:

```
#查看网站的电影年代分布
year = df['release_year'].unique()
year = year[np.argsort(year)]
year
```

Out[59]:

```
array([1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1
970,
       1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1
981,
       1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1
992,
       1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2
003,
       2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2
014,
       2015])
```

In [60]:

```
#根据年代计算平均值
df_avg = df.groupby('release_year').mean()
df_avg
```

Out[60]:

| release_year | popularity | budget | revenue | runtime | vote_count | vote_average | |
|---|---|---|---|---|---|---|---|
| **1960** | 0.368835 | 3.320000e+05 | 2.120200e+06 | 106.560000 | 35.360000 | 6.176000 | 2 |
| **1961** | 0.442715 | 1.640607e+06 | 1.199001e+07 | 121.357143 | 83.428571 | 6.346429 | 1 |
| **1962** | 0.430630 | 1.742142e+06 | 7.433788e+06 | 123.965517 | 72.827586 | 6.279310 | 1 |
| **1963** | 0.510663 | 2.528672e+06 | 6.087134e+06 | 112.862069 | 86.655172 | 6.300000 | 1 |
| **1964** | 0.415003 | 9.630039e+05 | 8.316629e+06 | 108.512195 | 74.926829 | 6.202439 | 6 |
| **1965** | 0.314955 | 1.653092e+06 | 1.018706e+07 | 115.852941 | 49.823529 | 6.173529 | 1 |
| **1966** | 0.311006 | 1.337548e+06 | 2.017540e+06 | 106.047619 | 31.047619 | 6.135714 | 8 |
| **1967** | 0.380738 | 1.877291e+06 | 1.374989e+07 | 104.171429 | 57.028571 | 6.277143 | 1 |
| **1968** | 0.450295 | 1.675400e+06 | 5.655489e+06 | 106.485714 | 91.314286 | 6.328571 | 1 |
| **1969** | 0.426495 | 1.404303e+06 | 7.656903e+06 | 105.866667 | 54.500000 | 5.926667 | 8 |
| **1970** | 0.349693 | 3.255563e+06 | 1.436467e+07 | 112.923077 | 50.769231 | 6.358974 | 1 |
| **1971** | 0.448349 | 1.250940e+06 | 7.790744e+06 | 106.480000 | 96.120000 | 6.400000 | 6 |
| **1972** | 0.471532 | 7.953429e+05 | 1.003814e+07 | 102.028571 | 147.542857 | 6.520000 | 4 |
| **1973** | 0.485700 | 1.051400e+06 | 2.536010e+07 | 104.638298 | 90.085106 | 6.734043 | 5 |
| **1974** | 0.472522 | 1.677857e+06 | 1.769044e+07 | 105.190476 | 119.547619 | 6.426190 | 7 |
| **1975** | 0.559355 | 1.443051e+06 | 2.378179e+07 | 107.256410 | 153.205128 | 6.387179 | 5 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1976** | 0.439639 | 2.223810e+06 | 1.556353e+07 | 109.738095 | 91.380952 | 6.309524 | 8 |
| **1977** | 0.630454 | 2.885357e+06 | 3.893898e+07 | 106.928571 | 143.232143 | 6.167857 | 1 |
| **1978** | 0.397090 | 3.475373e+06 | 2.109796e+07 | 110.254237 | 67.305085 | 6.096610 | 1 |
| **1979** | 0.602322 | 4.354981e+06 | 2.948153e+07 | 111.113208 | 162.622642 | 6.333962 | 1 |
| **1980** | 0.462031 | 4.822857e+06 | 2.296447e+07 | 107.600000 | 120.400000 | 6.141429 | 1 |
| **1981** | 0.455705 | 4.514104e+06 | 2.110136e+07 | 106.506667 | 94.840000 | 6.200000 | 1 |
| **1982** | 0.510683 | 4.567040e+06 | 2.890198e+07 | 103.105263 | 127.539474 | 6.218421 | 1 |
| **1983** | 0.547092 | 6.570980e+06 | 2.907409e+07 | 103.063291 | 125.012658 | 6.007595 | 1 |
| **1984** | 0.577267 | 7.323835e+06 | 2.460611e+07 | 104.144330 | 140.639175 | 5.974227 | 1 |
| **1985** | 0.574204 | 6.773530e+06 | 2.667951e+07 | 113.847619 | 127.990476 | 6.164762 | 1 |
| **1986** | 0.511902 | 6.004600e+06 | 2.578616e+07 | 100.060345 | 116.163793 | 5.992241 | 1 |
| **1987** | 0.509459 | 5.749626e+06 | 2.869794e+07 | 101.559322 | 119.644068 | 6.109322 | 1 |
| **1988** | 0.473934 | 6.601057e+06 | 2.668886e+07 | 101.250000 | 103.057143 | 5.954286 | 1 |
| **1989** | 0.569994 | 8.241652e+06 | 3.927678e+07 | 105.068702 | 141.305344 | 6.077099 | 1 |
| **1990** | 0.527021 | 9.814814e+06 | 3.909186e+07 | 105.560976 | 163.170732 | 5.992683 | 1 |
| **1991** | 0.472386 | 1.098630e+07 | 3.290847e+07 | 105.193798 | 114.751938 | 5.993023 | 1 |
| **1992** | 0.585725 | 1.097453e+07 | 4.604739e+07 | 107.102362 | 147.472441 | 6.081102 | 1 |
| **1993** | 0.546987 | 9.881562e+06 | 3.841838e+07 | 106.802326 | 137.656977 | 6.041860 | 1 |
| **1994** | 0.670794 | 1.189125e+07 | 3.837046e+07 | 106.763736 | 205.741758 | 5.941758 | 1 |
| **1995** | 0.715818 | 1.654665e+07 | 5.327455e+07 | 107.698225 | 204.621302 | 6.063314 | 2 |
| **1996** | 0.619823 | 1.859759e+07 | 4.323558e+07 | 104.494737 | 133.094737 | 5.866842 | 2 |
| **1997** | 0.693460 | 2.461482e+07 | 5.532803e+07 | 106.702128 | 212.973404 | 5.985638 | 3 |
| **1998** | 0.626033 | 2.137707e+07 | 4.492105e+07 | 105.126263 | 193.075758 | 5.978283 | 2 |
| **1999** | 0.651675 | 2.559011e+07 | 5.121183e+07 | 108.648402 | 242.255708 | 6.030594 | 3 |
| **2000** | 0.532895 | 2.523739e+07 | 4.733101e+07 | 103.445946 | 191.342342 | 5.881982 | 3 |
| **2001** | 0.706399 | 2.346002e+07 | 5.610913e+07 | 108.133891 | 263.108787 | 5.893305 | 2 |
| **2002** | 0.705637 | 2.217104e+07 | 5.450290e+07 | 103.988506 | 237.091954 | 5.966667 | 2 |
| **2003** | 0.721360 | 2.223963e+07 | 5.421750e+07 | 100.713262 | 245.046595 | 5.939068 | 2 |
| **2004** | 0.729237 | 2.389713e+07 | 5.563304e+07 | 105.293333 | 259.726667 | 5.991333 | 2 |
| **2005** | 0.627479 | 2.018390e+07 | 4.528920e+07 | 103.008451 | 200.442254 | 5.865634 | 2 |
| **2006** | 0.610985 | 1.809019e+07 | 4.026923e+07 | 101.676692 | 189.000000 | 5.947870 | 1 |
| **2007** | 0.593825 | 1.748350e+07 | 4.511560e+07 | 100.063830 | 206.581560 | 5.959102 | 1 |
| **2008** | 0.579733 | 1.552792e+07 | 3.868240e+07 | 100.198783 | 203.200811 | 5.928195 | 1 |
| **2009** | 0.601515 | 1.624874e+07 | 4.200541e+07 | 97.263757 | 226.017078 | 5.851803 | 1 |
| **2010** | 0.645780 | 1.920364e+07 | 4.518017e+07 | 98.152263 | 267.174897 | 5.993210 | 1 |
| **2011** | 0.679055 | 1.685818e+07 | 4.453345e+07 | 97.889098 | 253.526316 | 5.958835 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **2012** | 0.608736 | 1.414372e+07 | 4.216825e+07 | 98.124786 | 313.627350 | 5.795556 |
| **2013** | 0.620137 | 1.393072e+07 | 3.594434e+07 | 96.057011 | 322.332820 | 5.881202 |
| **2014** | 0.889351 | 1.135242e+07 | 3.485838e+07 | 98.358166 | 295.424069 | 5.923782 |
| **2015** | 1.030657 | 1.207718e+07 | 4.254762e+07 | 96.375199 | 290.019078 | 5.885692 |

In [61]:

```
#根据柱状图得出最受欢迎电影是哪一年上映的
#获取平均分最大值与最小值
df_avg['vote_average'].max()
```

Out[61]:

```
6.7340425531914905
```

In [62]:

```
df_avg['vote_average'].min()
```

Out[62]:

```
5.795555555555555
```

In [63]:

```
df_avg['vote_average']
```

Out[63]:

```
release_year
1960    6.176000
1961    6.346429
1962    6.279310
1963    6.300000
1964    6.202439
1965    6.173529
1966    6.135714
1967    6.277143
1968    6.328571
1969    5.926667
1970    6.358974
1971    6.400000
1972    6.520000
1973    6.734043
1974    6.426190
1975    6.387179
1976    6.309524
1977    6.167857
1978    6.096610
1979    6.333962
1980    6.141429
1981    6.200000
1982    6.218421
1983    6.007595
1984    5.974227
1985    6.164762
```

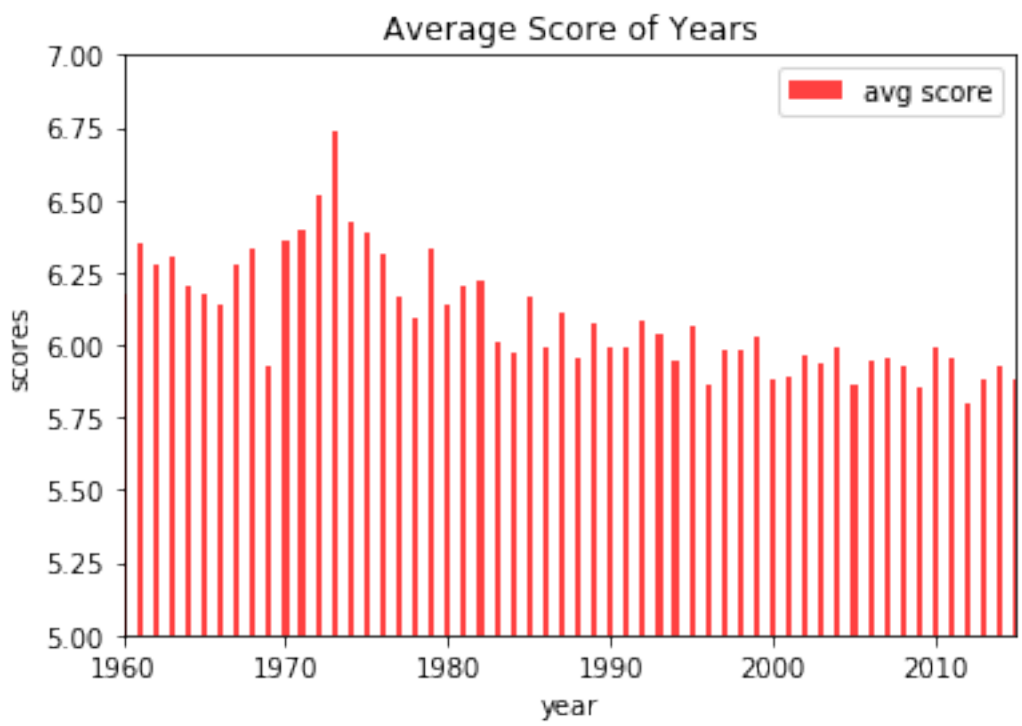
```
1986    5.992241
1987    6.109322
1988    5.954286
1989    6.077099
1990    5.992683
1991    5.993023
1992    6.081102
1993    6.041860
1994    5.941758
1995    6.063314
1996    5.866842
1997    5.985638
1998    5.978283
1999    6.030594
2000    5.881982
2001    5.893305
2002    5.966667
2003    5.939068
2004    5.991333
2005    5.865634
2006    5.947870
2007    5.959102
2008    5.928195
2009    5.851803
2010    5.993210
2011    5.958835
2012    5.795556
2013    5.881202
2014    5.923782
2015    5.885692
Name: vote_average, dtype: float64
```

In [64]:

```
#得出两组数据的柱状图
width = 0.35
score = df_avg['vote_average']
plt.bar(year , score , width , color = 'r' , alpha = 0.75 , label = 'avg score')
plt.xlabel('year')
plt.ylabel('scores')
plt.xlim(1960 , 2015)
plt.ylim(5 , 7)
plt.title('Average Score of Years')
plt.legend()
```

Out[64]:

```
<matplotlib.legend.Legend at 0x1227847f0>
```




In [65]:

```
df_avg['vote_average']
```

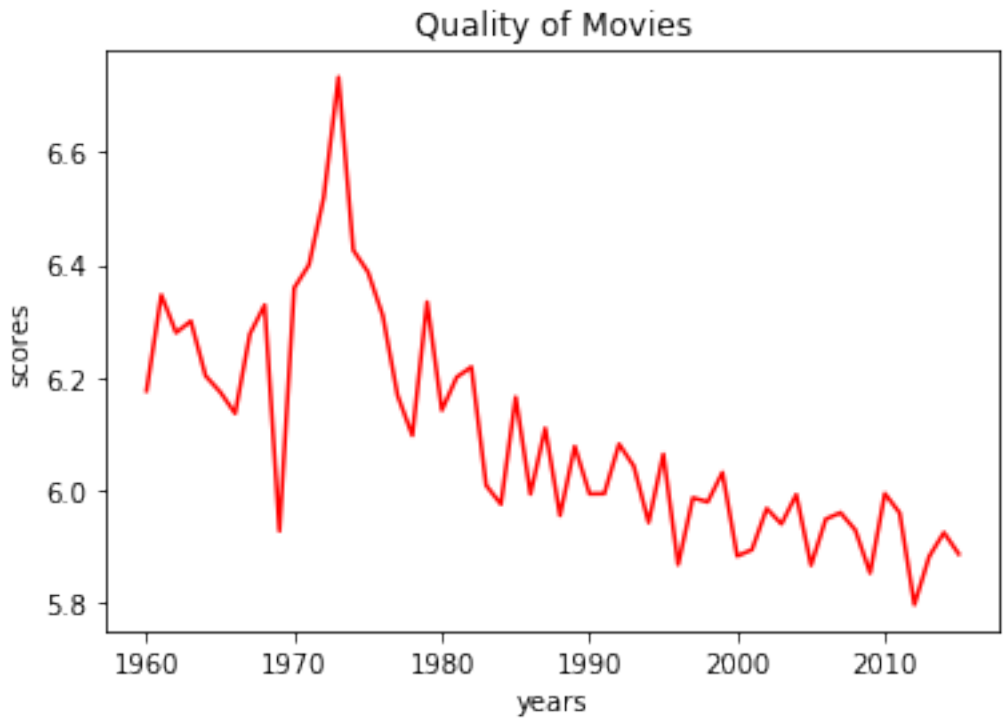
Out[65]:

```
release_year
1960    6.176000
1961    6.346429
1962    6.279310
1963    6.300000
1964    6.202439
1965    6.173529
1966    6.135714
1967    6.277143
1968    6.328571
1969    5.926667
1970    6.358974
1971    6.400000
1972    6.520000
1973    6.734043
1974    6.426190
1975    6.387179
```

```
1976    6.309524
1977    6.167857
1978    6.096610
1979    6.333962
1980    6.141429
1981    6.200000
1982    6.218421
1983    6.007595
1984    5.974227
1985    6.164762
1986    5.992241
1987    6.109322
1988    5.954286
1989    6.077099
1990    5.992683
1991    5.993023
1992    6.081102
1993    6.041860
1994    5.941758
1995    6.063314
1996    5.866842
1997    5.985638
1998    5.978283
1999    6.030594
2000    5.881982
2001    5.893305
2002    5.966667
2003    5.939068
2004    5.991333
2005    5.865634
2006    5.947870
2007    5.959102
2008    5.928195
2009    5.851803
2010    5.993210
2011    5.958835
2012    5.795556
2013    5.881202
2014    5.923782
2015    5.885692
Name: vote_average, dtype: float64
```

In [66]:

```
#得出两组数据的折线图
plt.plot(score , 'r' , label = 'scores changes')
plt.xlabel('years')
plt.ylabel('scores')
plt.title('Quality of Movies')
plt.show()
```




回答：从以上两图可以看出，电影质量最高的一年是在1973年，这年的电影平均分最高。而电影的质量在这几年虽然有些摆动的厉害，但总体而言是在下降的。

问题3:评分越高的电影是否越受欢迎?

In [46]:

```
#找到两组数据的最大值
df['popularity'].max()
```

Out[46]:

```
32.985763
```

In [44]:

```
df['vote_average'].max()
```

Out[44]:

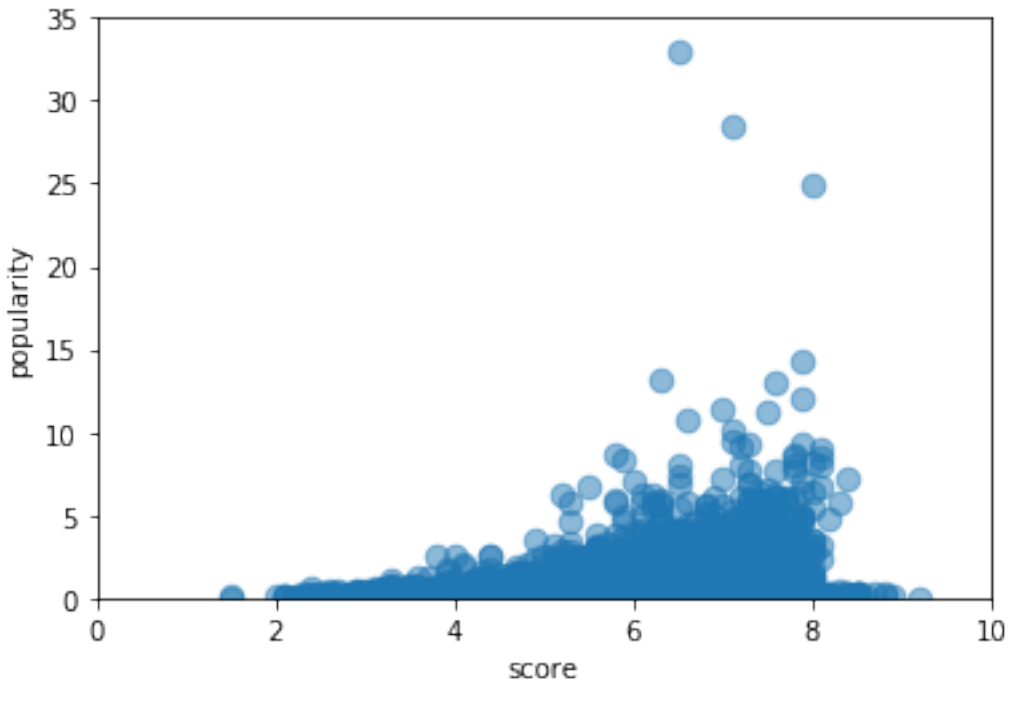
```
9.2
```

In [ ]:

```
df['popuarity']
```

In [48]:

```
#为了研究两组数据的相关性，这里还用散点图来表示
x = df['vote_average']
y = df['popularity']
plt.scatter(x , y , s = 75 , alpha = 0.5)
plt.xlim(0 , 10)
plt.ylim(0 , 35)
plt.xlabel('score')
plt.ylabel('popularity')
plt.show()
```




回答：从上图可以看出，电影的评分与播放量具有一定的相关性，大部分情况下评分越高的电影播放量越高。但是还有一种情况，可以看到得分超过8分的电影播放量非常低。受到数据的限制，我们无法知道这些电影是由于参与评分的人数少导致评分过高，还是因为这些是小众文艺电影所以虽然质量不错但是受众群不广。

结论：根据对imdb所给出的电影数据分析，我们可以知道：电影的质量可能会受到预算的影响，但是就算是小成本电影，依然可以得到市场的认可，收获很高的票房；电影的质量并不是稳定的，经常在一些时间段内出现剧烈的摆动，并且总体而言质量在下降；一般情况下，评分高的电影会得到更的播放量，但是一些评分特别高的电影，却播放量非常少，究竟是什么原因使得这些高分电影的播放量低，在数据中暂时还不能体现，而评分和播放量两者之间具体的关系，暂时也不能知道。