探索电影数据集

在这个项目中，你将尝试使用所学的知识，使用 NumPy、Pandas、matplotlib、seaborn 库中的函数，来对电影数据集进行探索。

下载数据集： TMDb电影数据 (https://s3.cn-north-1.amazonaws.com.cn/static-documents/nd101/explore+dataset/tmdb-movies.csv)

数据集各列名称的含义：

列名称	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage	director	tagline	keywords	overview
含义	编号	IMDB 编号	知名度	预算	票房	名称	主演	网站	导演	宣传词	关键词	简介

请注意，你需要提交该报告导出的 .html、.ipynb 以及 .py 文件。

第一节 数据的导入与处理

在这一部分，你需要编写代码，使用 Pandas 读取数据，并进行预处理。

任务1.1： 导入库以及数据

1. 载入需要的库 NumPy、Pandas、matplotlib、seaborn。
2. 利用 Pandas 库，读取 tmdb-movies.csv 中的数据，保存为 movie_data。

提示：记得使用 notebook 中的魔法指令 %matplotlib inline，否则会导致你接下来无法打印出图像。

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

%matplotlib inline
```

In [2]:
```
movie_data = pd.read_csv('tmdb-movies.csv')
```

任务1.2: 了解数据

你会接触到各种各样的数据表，因此在读取之后，我们有必要通过一些简单的方法，来了解我们数据表是什么样子的。

1. 获取数据表的行列，并打印。
2. 使用 .head()、.tail()、.sample() 方法，观察、了解数据表的情况。
3. 使用 .dtypes 属性，来查看各列数据的数据类型。
4. 使用 isnull() 配合 .any() 等方法，来查看各列是否存在空值。
5. 使用 .describe() 方法，看看数据表中数值型的数据是怎么分布的。

通关提示： 把上面的每个方法都要运行一遍

- .head(n=5)、.tail(n=5)、.sample(n=1)都可以传入相应的数量
- .dtypes查看数据类型有助于我们对数据做修改、赋值等处理
- isnull()配合.any()、 .sum()是检查Nan值的常用方法
- .describe() 会返回各个数值类型列的分位数、平均值、最大最小值等信息。分位数的计算可以使用series.quantile()方法。
  https://www.cnblogs.com/gispathfinder/p/5770091.html (https://www.cnblogs.com/gispathfinder/p/5770091.html)

In [3]:
```
# .head()
```

In [ ]:
```
# .tril()
```

In [ ]:
```
# .sample()
```

In [5]:
```
#.dtypes
```

In [ ]:
```
#.isnull() .any()
```

In [6]:
```
# .describe()
movie_data.describe()
```

Out[6]:

	id	popularity	budget	revenue	runtime	vote_count	v
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.000000	1
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389748	5
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619058	0
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.000000	1
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.000000	5
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.000000	6
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750000	6
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000	9

In [11]:
```
movie_data.id.quantile([0.33, 0.75])
```

Out[11]:
```
0.33     12473.9
0.75     75610.0
Name: id, dtype: float64
```

任务1.3: 清理数据

在真实的工作场景中，数据处理往往是最为费时费力的环节。但是幸运的是，我们提供给大家的 tmdb 数据集非常的「干净」，不需要大家做特别多的数据清洗以及处理工作。在这一步中，你的核心的工作主要是对数据表中的空值进行处理。你可以使用 .fillna() 来填补空值，当然也可以使用 .dropna() 来丢弃数据表中包含空值的某些行或者列。

任务：使用适当的方法来清理空值，并将得到的数据保存。

通关必备：

1. 表格中的数据类型分为两大类，字符类型和数值类型，字符类型是object，数值类型包括 int64 和 float64。其中，数值类型的空值已经再数据中用数值0表示，字符类型的数据是Nan,处理的适合可以分开处理。
2. 可以使用movie_data.info()查看各个列的数据类型和数据长度。
3. 使用movie_data.select_dtypes(inlude=)可以根据数据类型赛选数据。
4. 使用replace(0, np.nan)可以把数据0替换为Nan。
5. 使用dropna(subset=[])对指定列进行操作。
6. 使用fillna(value=)把Nan的值填充为指定值。
7. 使用interpolate()对数值类型列的Nan值做插值填充。

In [14]:
```
# 使用movie_data.info()查看各个列的数据类型和数据长度
movie_data.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 [13]:
```
# 查看非数值类型的格列Nan的个数
movie_data.isnull().sum()
```

Out[13]:
```
id                      0
imdb_id                10
popularity              0
budget                  0
revenue                 0
original_title          0
cast                   76
homepage             7930
director               44
tagline              2824
keywords             1493
overview                4
runtime                 0
genres                 23
production_companies  1030
release_date            0
vote_count              0
vote_average            0
release_year            0
budget_adj              0
revenue_adj             0
dtype: int64
```

In [ ]:
```
# 因为imdb_id、overview，genres的包含nan的值较少，
# 所以这里drop掉这些包含non值的行，不会过分影响数据

# 其他列含nan的值填充'Unknown'
```

In [ ]:
```
# 再次查看数值类型的格列Nan的个数， 确认处理结果
```

In [17]:
```
# 根据movie_data.describe()，budget、revenue、budget_adj、revenue_adj
# 这4列数据存在大量的0
print(movie_data[(movie_data.budget == 0)].budget.value_counts())
print(movie_data[(movie_data.revenue == 0)].revenue.value_counts())
```

```
0    5696
Name: budget, dtype: int64
0    6016
Name: revenue, dtype: int64
```

In [20]:
```
# 把0值替换为 nan
movie_data[['budget', 'revenue', 'budget_adj', 'revenue_adj' ]].replace(0, np.nan)
# 把nan做插值
movie_data.interpolate?
```

In [ ]:
```
# 查看是否处理完
movie_data.describe()
```

第二节 根据指定要求读取数据

相比 Excel 等数据分析软件，Pandas 的一大特长在于，能够轻松地基于复杂的逻辑选择合适的数据。因此，如何根据指定的要求，从数据表当获取适当的数据，是使用 Pandas 中非常重要的技能，也是本节重点考察大家的内容。

任务2.1: 简单读取

1. 读取数据表中名为 id、popularity、budget、runtime、vote_average 列的数据。
2. 读取数据表中前1～20行以及48、49行的数据。
3. 读取数据表中第50～60行的 popularity 那一列的数据。

要求：每一个语句只能用一行代码实现。

通关必备：

1. 读取多列 movie_data[]中传入一个list.
2. 读取多行movie_data.iloc[]中传入一个list
3. 某行某列movie['popularity'].iloc[]先选列再选行

In [ ]:
```
movie_data[[]]
```

In [22]:
```
movie_data.iloc[[1,2,3, 48, 49]]
```

Out[22]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	http://ww
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	http://ww
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	http://ww episod...
48	265208	tt2231253	2.932340	30000000	0	Wild Card	Jason Statham|Michael Angarano|Milo Ventimigli...	NaN
49	254320	tt3464902	2.885126	4000000	9064511	The Lobster	Colin Farrell|Rachel Weisz|LÃ©a Seydoux|John C...	http://tick

5 rows × 21 columns

In [ ]:
```
movie['popularity'].iloc[[]]
movie.iloc[[]].popularity
```

任务2.2: 逻辑读取（Logical Indexing）

1. 读取数据表中 `popularity` **大于5** 的所有数据。
2. 读取数据表中 `popularity` **大于5** 的所有数据且**发行年份在1996年之后**的所有数据。

提示：Pandas 中的逻辑运算符如 `&`、`|`，分别代表且以及或。

要求：请使用 Logical Indexing实现。

通关必备

条件筛选：`movie_data[(movie_data['popularity'] > 5) & (movie_data['popularity'] < 7)]`

任务2.3: 分组读取

1. 对 `release_year` 进行分组，使用 `.agg` (http://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.core.groupby.DataFrameGroupBy.agg.html) 获得 `revenue` 的均值。
2. 对 `director` 进行分组，使用 `.agg` (http://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.core.groupby.DataFrameGroupBy.agg.html) 获得 `popularity` 的均值，从高到低排列。

要求：使用 `Groupby` 命令实现。

通关必备

1. .agg()可以传入一个函数名，比如'sum'按列计算和，'mean'按列计算平均值。
2. .agg()返回一个DataFrame， 然后再选取'revenue'列
3. 使用`sort_values('列名', ascending=False)`进行排序，ascending参数为True是升序，False是降序

In [28]:
```
movie_data.groupby('release_year').agg('sum')
```

Out[28]:

	id	popularity	budget	revenue	runtime	vote_count	vote_avera
release_year							
1960	502889	14.685834	22056948	145005000	3541	2481	202.4
1961	578367	13.107641	46137000	337720188	3702	2405	197.6
1962	544034	14.553069	54722126	215579846	3979	2392	203.0
1963	562904	17.092019	73331500	187404989	3785	2816	215.2
1964	729942	17.321989	39483161	340981782	4587	3137	260.9
1965	614765	11.990529	70205115	458081854	4136	1820	216.8
1966	759644	13.989152	57554800	84736689	4917	1460	281.9
1967	697188	18.846147	100652200	737834637	4198	3102	249.7
1968	726515	18.883866	71939000	264732980	4184	4217	248.8
1969	591357	13.106127	42129087	243957076	3304	1733	184.8
1970	751907	14.009686	126966946	560221969	4594	2011	263.1
1971	1137584	24.646153	75997000	404910610	5925	5081	353.1
1972	704684	17.962162	36279254	494730171	4078	5433	261.4
1973	997893	27.195419	65190783	1223981102	5694	5173	368.7
1974	910835	21.047978	76970000	812539818	4964	5207	300.6
1975	760791	22.855561	56279000	957489966	4724	6088	281.2
1976	744888	22.302266	122150000	801005600	5138	4720	298.0
1977	1420531	35.704048	161580000	2180583159	6105	8031	350.4
1978	1411200	26.865433	208997011	1369779659	7155	4898	398.5
1979	1086718	33.590762	254814000	1684794913	6385	8830	359.6
1980	1639820	38.660886	362500000	1768662387	8385	10898	480.5
1981	1560246	36.277875	373757786	1774606236	8681	7480	505.6
1982	1403773	44.011589	437795002	2458443852	8418	11385	505.6
1983	1400668	43.315574	519107412	2307529320	8240	9886	477.7
1984	1611152	62.038521	729211964	2635524418	10864	15371	630.4
1985	1629950	63.662354	748720637	2875772392	12463	13761	673.3
1986	1886604	61.655002	704533613	3002778281	12092	13675	726.6
1987	2048079	63.079564	709455811	3462104847	12646	15064	766.1
1988	2707730	67.430667	925348000	3739550845	14760	14576	865.0
1989	2629704	77.221155	1079656360	5164923718	14355	18657	831.6
1990	1857851	70.716996	1289922066	5315166660	13946	21575	791.2
1991	1905608	66.644959	1466233000	4706599796	13980	17006	799.9
1992	1794978	77.527364	1441765370	6078153217	14235	19447	808.8
1993	2788827	97.375953	1779628653	6955151167	19068	24284	1076.9
1994	3231011	123.063733	2229207032	7095429177	19672	37591	1093.4
1995	2920928	124.376366	2865884377	9156341160	18821	35810	1059.8
1996	3066028	123.372747	3687042051	8311492279	21564	26031	1203.8

	id	popularity	budget	revenue	runtime	vote_count	vote_avera
release_year							
1997	2925074	136.704630	4751086675	10655173234	20449	40985	1149.8
1998	3092866	131.494588	4499660000	9493174938	22063	40042	1253.8
1999	4150239	144.658971	5765235106	11355712579	24385	53447	1351.3
2000	3611297	124.123452	5752700000	10978701012	23558	46206	1335.3
2001	4144137	170.043163	5641944000	13410083139	26144	63058	1426.9
2002	3607313	186.586849	5894640255	14643618528	28426	62904	1588.9
2003	5527642	202.062290	6239857694	15138243542	28291	68425	1666.6
2004	4468149	221.788592	7170340222	16793822618	32347	79200	1838.5
2005	5140666	228.833453	7343284349	16516835108	37378	73336	2135.4
2006	5906687	247.399250	7306185300	16275739385	41487	76083	2424.1
2007	7005822	259.804380	7635569004	19411668670	43980	90391	2612.4
2008	9101362	290.069191	7781262597	19431695138	49739	102562	2941.7
2009	16241077	319.895074	8594084056	22180170559	52261	119689	3121.6
2010	23087632	316.029586	9385001006	21959998545	48079	130149	2935.5
2011	37302676	364.537424	9018153652	23695591578	52878	135439	3217.6
2012	61545950	357.031964	8274084052	24668428824	57607	183539	3410.3
2013	111910769	413.606395	9236038361	24703633017	63293	214486	3875.3
2014	170103095	621.087887	7923990138	24331150183	68832	206262	4144.5
2015	186663306	648.283099	7596547557	26762450518	60620	182422	3702.1

第三节 绘图与可视化

接着你要尝试对你的数据进行图像的绘制以及可视化。这一节最重要的是，你能够选择合适的图像，对特定的可视化目标进行可视化。所谓可视化的目标，是你希望从可视化的过程中，观察到怎样的信息以及变化。例如，观察票房随着时间的变化、哪个导演最受欢迎等。

可视化的目标	可以使用的图像
表示某一属性数据的分布	饼图、直方图、散点图
表示某一属性数据随着某一个变量变化	条形图、折线图、热力图
比较多个属性的数据之间的关系	散点图、小提琴图、堆积条形图、堆积折线图

在这个部分，你需要根据题目中问题，选择适当的可视化图像进行绘制，并进行相应的分析。对于选做题，他们具有一定的难度，你可以尝试挑战一下～

任务3.1： 对 `popularity` 最高的20名电影绘制其 `popularity` 值。

通关必备

1. 首先按照'popularity'对movie_data按降序排序.sort_values('列名', ascending=False)，并取出排在前20的数据，保存为popularity_movies。
2. 针对popularity_movies，选择适合的图形，图中坐标轴，一个是original_title，一个是popularity。
3. 如果画条形图，使用sb.barplot()，或者pandas自带的api,plot.barh(), 又或者使用matplotlib直接画图

In [ ]:
```
# 按popularity排序，取出前20
popularity_movies = movie_data.sort_values('popularity', ascending=False)[][:20]
```

In [ ]:
```
# 画图
```

任务3.2：分析电影净利润（票房-成本）随着年份变化的情况，并简单进行分析。

通关必备：

1. profit = revenue - budge
2. 可以求出每年利润的平均值(mean)再画图, 还可以求出标准差作为误差(sem)
3. 使用plt.errorbar()画图，或者使用pandas api画图 .plot(x = y_means.index, yerr=y_sems)

In [ ]:
```
# 计算利润 movie_data['revenue'] - movie_data['budget']
```

In [ ]:
```
# 计算平均值和标准差
y_means = movie_data.groupby('release_year')['profit'].mean()
y_sems = movie_data.groupby('release_year')['profit'].sem()
```

In [ ]:
```
# 使用plt画图
```

分析:

[选做]任务3.3：选择最多产的10位导演（电影数量最多的），绘制他们排行前3的三部电影的票房情况，并简要进行分析。

通关必备：

1. 原来数据中单部电影多名导演是用'|'隔开的， 会需要处理包含'|'的director数据
2. 讲director数据分成两类，第一类分为带分割符的，第二类分为带分割符的， 把第一类数据融入第二类数据中。比如，如果导演数据是'Kyle Balda|Pierre Coffin',这需要把其分为Kyle Balda和Coffin做处理。
3. 使用value_counts来快速统计每个导演的作品数量。
4. 根据统计结果找到排名前十的导演。
5. 然后找到排名前十导演的前3部最受欢迎的电影。
6. 画图，建议使用sb.barplot(data=target_data, x='original_title', y='revenue_adj', hue='director')

In [39]:
```
movie_data.dropna(inplace=True)
```

In [45]:
```
select_split_director = movie_data.director.str.contains('\|')
split_director = movie_data[select_split_director].director.str.split('|')
```

In [46]:
```
split_director
```

Out[46]:
```
8                              [Kyle Balda, Pierre Coffin]
11                         [Lana Wachowski, Lilly Wachowski]
64                               [Glenn Ficarra, John Requa]
132                          [Mark Burton, Richard Starzack]
143                          [Gerado Olivares, Otmar Penker]
214                                  [Ryan Fleck, Anna Boden]
239                            [Toby Genkel, Sean McCormack]
241                               [Jill Bauer, Ronna Gradus]
242                             [Travis Cluff, Chris Lofing]
255                                [Tom Gianas, Ross Shuman]
397                               [Ben Blaine, Chris Blaine]
596                          [Robert Gordon, Morgan Neville]
631                               [Joe Russo, Anthony Russo]
632                           [Chad Stahelski, David Leitch]
635                               [Don Hall, Chris Williams]
666                          [Phil Lord, Christopher Miller]
676                          [Phil Lord, Christopher Miller]
710                        [Frank Miller, Robert Rodriguez]
893                    [Alastair Fothergill, Keith Scholey]
954                                [Andrew Erwin, Jon Erwin]
969                            [Ishi Rudell, Jayson Thiessen]
975                        [Jason Friedberg, Aaron Seltzer]
990                          [Kevin Kolsch, Dennis Widmyer]
996                            [James D. Rolfe, Kevin Finn]
1232                         [Michael Hewitt, Dermot Lavery]
1336                  [John Lounsbery, Wolfgang Reitherman]
1390                               [Bob Peterson, Pete Docter]
1402                               [Ron Clements, John Musker]
1411                         [Phil Lord, Christopher Miller]
1423                           [Mark Neveldine, Brian Taylor]
                                       ...
6590                                 [David Bowers, Sam Fell]
6648                               [Anthony Russo, Joe Russo]
6649                         [Jonathan Dayton, Valerie Faris]
6671          [Roger Allers, Jill Culton, Anthony Stacchi]
6778            [Paul Bolger, Yvette Kaplan, Greg Tiernan]
6881                 [Mat Whitecross, Michael Winterbottom]
6977         [Andrew Adamson, Kelly Asbury, Conrad Vernon]
7046                               [Will Finn, John Sanford]
7389                                [Brad Bird, Jan Pinkava]
7411                               [Chris Miller, Raman Hui]
7415                         [Steve Hickner, Simon J. Smith]
7431                           [Colin Strause, Greg Strause]
7466                               [Ash Brannon, Chris Buck]
7573                         [David Nerlich, Andrew Traucki]
7598                      [Bernie Goldmann, Melisa Wallack]
7630                   [Alastair Fothergill, Mark Linfield]
7644                         [Patrick Archibald, Jay Oliva]
7647                    [Luis Piedrahita, Rodrigo SopeÃ±a]
7680                           [Lauren Montgomery, Bruce Timm]
8243                               [Ron Clements, John Musker]
8378                [Ted Berman, Art Stevens, Richard Rich]
8977                         [John Lasseter, Andrew Stanton]
8985                        [Bobby Farrelly, Peter Farrelly]
9316                               [Gary Trousdale, Kirk Wise]
9421     [Eleanor Coppola, Fax Bahr, George Hickenlooper]
9631                                [Peter Harris, Eric Till]
9708                                [Y.K. Kim, Woo-sang Park]
9807                             [Terry Gilliam, Terry Jones]
10213                             [Peter Clifton, Joe Massot]
10328          [John Carpenter, Tobe Hooper, Larry Sulkis]
Name: director, Length: 175, dtype: object
```

In [47]:
```
non_split_director = movie_data[~select_split_director].director.value_counts()
```

In [48]:
```
non_split_director
```

Out[48]:
```
John Carpenter          17
Steven Soderbergh       11
Steven Spielberg        11
Clint Eastwood           8
Peter Jackson            8
Robert Zemeckis          8
Ridley Scott             8
Paul W.S. Anderson       7
Christopher Nolan        7
David Fincher            7
Ron Howard               7
Martin Scorsese          7
Quentin Tarantino        6
Michael Bay              6
Francis Ford Coppola     6
Francis Lawrence         6
Martin Campbell          5
Brian Robbins            5
Guy Ritchie              5
George Lucas             5
Gore Verbinski           5
Kevin Smith              5
Rob Zombie               5
James Wan                5
Marc Forster             5
Chris Columbus           5
Peter Berg               5
John Glen                5
Dennis Dugan             5
Darren Aronofsky         5
                        ..
Ben Ketai                1
Michael Lehmann          1
J Blakeson               1
Bradley King             1
Rodrigo GarcÃa          1
Ed Decter                1
Joey Ansah               1
John Lasseter            1
Joshua Michael Stern     1
Julie Anne Robinson      1
Jonas Elmer              1
Dermot Mulroney          1
Mike Figgis              1
Patty Jenkins            1
Steve Carr               1
Alan Poul                1
Tod Williams             1
Blair Erickson           1
Damon Gameau             1
Jared Hess               1
Mary Agnes Donoghue      1
Billy Corben             1
Spencer Susser           1
David Soren              1
Julia Leigh              1
Robbie Pickering         1
Mel Brooks               1
Kristin Hanggi           1
Josh Trank               1
Leigh Whannell           1
Name: director, Length: 1187, dtype: int64
```

In [49]:
```
for director in split_director:
    for d in director:
        if d in non_split_director:
            non_split_director[d] += 1
        else:
            non_split_director[d] = 1
```

Out[49]:
```
John Carpenter          18
Steven Soderbergh       11
Steven Spielberg        11
Clint Eastwood           8
Peter Jackson            8
Robert Zemeckis          8
Ridley Scott             8
Paul W.S. Anderson       7
Christopher Nolan        7
David Fincher            7
Ron Howard               7
Martin Scorsese          7
Quentin Tarantino        7
Michael Bay              6
Francis Ford Coppola     6
Francis Lawrence         6
Martin Campbell          5
Brian Robbins            5
Guy Ritchie              5
George Lucas             5
Gore Verbinski           5
Kevin Smith              5
Rob Zombie               5
James Wan                5
Marc Forster             5
Chris Columbus           5
Peter Berg               5
John Glen                5
Dennis Dugan             5
Darren Aronofsky         5
                        ..
Yvette Kaplan            1
Greg Tiernan             1
Mat Whitecross           1
Will Finn                1
John Sanford             1
Jan Pinkava              1
Chris Miller             1
Simon J. Smith           1
David Nerlich            1
Bernie Goldmann          1
Melisa Wallack           1
Mark Linfield            1
Patrick Archibald        1
Luis Piedrahita          1
Rodrigo SopeÃ±a          1
Bruce Timm               1
Ted Berman               1
Art Stevens              1
Richard Rich             1
Eleanor Coppola          1
Fax Bahr                 1
Peter Harris             1
Eric Till                1
Y.K. Kim                 1
Woo-sang Park            1
Terry Jones              1
Peter Clifton            1
Joe Massot               1
Tobe Hooper              1
Larry Sulkis             1
Name: director, Length: 1444, dtype: int64
```

In [ ]:
```
# 再次排序
non_split_director
```

In [50]:
```
non_split_director[:10]
```

Out[50]:
```
John Carpenter        18
Steven Soderbergh     11
Steven Spielberg      11
Clint Eastwood         8
Peter Jackson          8
Robert Zemeckis        8
Ridley Scott           8
Paul W.S. Anderson     7
Christopher Nolan      7
David Fincher          7
Name: director, dtype: int64
```

In [51]:
```
# 排行前十的director

top10_director = pd.Series(non_split_director[:10].index, name='director').reset_index()
top10_director
```

Out[51]:

	index	director
0	0	John Carpenter
1	1	Steven Soderbergh
2	2	Steven Spielberg
3	3	Clint Eastwood
4	4	Peter Jackson
5	5	Robert Zemeckis
6	6	Ridley Scott
7	7	Paul W.S. Anderson
8	8	Christopher Nolan
9	9	David Fincher

In [53]:
```
sort_data = movie_data.merge(top10_director, on='director').sort_values(['index', 'revenue_adj'],ascending=[True, False])
```

In [55]:
```
target_data = sort_data.groupby('director').head(3)
target_data
```

Out[55]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	
73	948	tt0077651	1.198849	300000	70000000	Halloween	Donald Pleasence|Jamie Lee Curtis|P.J. Soles|N...	http
64	1103	tt0082340	1.003963	6000000	50000000	Escape from New York	Kurt Russell|Lee Van Cleef|Ernest Borgnine|Don...	http
60	9663	tt0088172	0.637446	22000000	28744356	Starman	Jeff Bridges|Karen Allen|Charles Martin Smith|...	http
48	161	tt0240772	2.642249	85000000	450717150	Ocean's Eleven	George Clooney|Brad Pitt|Matt Damon|Andy GarcÃ...	http ele
54	163	tt0349903	2.175284	110000000	362744280	Ocean's Twelve	George Clooney|Brad Pitt|Catherine Zeta-Jones|...	http
55	298	tt0496806	1.414350	85000000	311312624	Ocean's Thirteen	George Clooney|Brad Pitt|Matt Damon|Al Pacino|...	http
89	578	tt0073195	2.563191	7000000	470654000	Jaws	Roy Scheider|Robert Shaw|Richard Dreyfuss|Lorr...	http
87	601	tt0083866	2.900556	10500000	792910554	E.T. the Extra-Terrestrial	Henry Thomas|Drew Barrymore|Robert MacNaughton...	http
91	329	tt0107290	2.204926	63000000	920100000	Jurassic Park	Sam Neill|Laura Dern|Jeff Goldblum|Richard Att...	http
38	190859	tt2179136	3.863074	58800000	542307423	American Sniper	Bradley Cooper|Sienna Miller|Kyle Gallner|Cole...	http

	id	imdb_id	popularity	budget	revenue	original_title	cast	
40	13223	tt1205489	2.245306	33000000	269958228	Gran Torino	Clint Eastwood|Christopher Carley|Bee Vang|Ahn...	http
44	70	tt0405159	1.648554	30000000	216763646	Million Dollar Baby	Clint Eastwood|Hilary Swank|Morgan Freeman|Jay...	http
29	122	tt0167260	7.122455	94000000	1118888979	The Lord of the Rings: The Return of the King	Elijah Wood|Ian McKellen|Viggo Mortensen|Liv T...	http
26	121	tt0167261	8.095275	79000000	926287400	The Lord of the Rings: The Two Towers	Elijah Wood|Ian McKellen|Viggo Mortensen|Liv T...	http
25	120	tt0120737	8.575419	93000000	871368364	The Lord of the Rings: The Fellowship of the Ring	Elijah Wood|Ian McKellen|Viggo Mortensen|Liv T...	http
11	105	tt0088763	6.095293	19000000	381109762	Back to the Future	Michael J. Fox|Christopher Lloyd|Lea Thompson|...	http
14	165	tt0096874	2.566875	40000000	332000000	Back to the Future Part II	Michael J. Fox|Christopher Lloyd|Lea Thompson|...	http
15	196	tt0099088	2.441201	40000000	244527583	Back to the Future Part III	Michael J. Fox|Christopher Lloyd|Mary Steenbur...	http
0	286217	tt3659388	7.667400	108000000	595380321	The Martian	Matt Damon|Jessica Chastain|Kristen Wiig|Jeff ...	http

	id	imdb_id	popularity	budget	revenue	original_title	cast	
3	70981	tt1446714	4.008188	130000000	403170142	Prometheus	Noomi Rapace|Michael Fassbender|Charlize Thero...	http
6	348	tt0078748	4.935897	11000000	104931801	Alien	Sigourney Weaver|Tom Skerritt|Veronica Cartwri...	http
74	35791	tt1220634	0.131526	60000000	296221663	Resident Evil: Afterlife	Milla Jovovich|Wentworth Miller|Ali Larter|Kim...	http
78	71679	tt1855325	0.260624	65000000	240159255	Resident Evil: Retribution	Milla Jovovich|Sienna Guillory|Michelle Rodrig...	http
79	395	tt0370263	1.348704	70000000	171183863	AVP: Alien vs. Predator	Sanaa Lathan|Raoul Bova|Ewen Bremner|Colin Sal...	http
19	49026	tt1345836	6.591277	250000000	1081041287	The Dark Knight Rises	Christian Bale|Michael Caine|Gary Oldman|Anne ...	http
18	155	tt0468569	8.466668	185000000	1001921825	The Dark Knight	Christian Bale|Michael Caine|Heath Ledger|Aaro...	http
17	27205	tt1375666	9.363643	160000000	825500000	Inception	Leonardo DiCaprio|Joseph Gordon-Levitt|Ellen P...	http
37	807	tt0114369	4.765359	33000000	327311859	Se7en	Brad Pitt|Morgan Freeman|Gwyneth Paltrow|John ...	http
31	210577	tt2267998	6.438727	61000000	369330363	Gone Girl	Ben Affleck|Rosamund Pike|Carrie Coon|Neil Pat...	http

	id	imdb_id	popularity	budget	revenue	original_title	cast	
34	4922	tt0421715	2.537342	150000000	333932083	The Curious Case of Benjamin Button	Cate Blanchett|Brad Pitt|Tilda Swinton|Julia O...	http

30 rows × 22 columns

分析:

[选做]任务3.4：分析1968年~2015年六月电影的数量的变化。

通关必备：

1. 使用pd.to_datetime()把movie_data['release_date']转换成datetime格式。
2. movie_data['release_date'].dt.year可以访问年份，movie_data['release_date'].dt.month可以访问月份
3. 设置年份筛选条件，可以使用 .dt.year.between() 进行范围筛选。
4. 设置月份筛选条件， 可以使用 .dt.month ==进行特定值筛选。
5. 根据前两个筛选条件，对movie_data筛选数据，并按年份统计结果。可以使用 .release_year.value_counts().sort_index()。
6. 对结果进行画图。

In [ ]:
```
# movie_data['release_date']转化成datetime格式
movie_data['release_date'] =
```

In [ ]:
```
# 筛选条件1 年份
select_years =
```

In [ ]:
```
# 筛选条件2 六月
select_month =
```

In [ ]:
```
# 筛选数据并画图
ret = movie_data[select_years & select_month]
ret.plot()
```

分析:

[选做]任务3.5：分析1968年~2015年六月电影 Comedy 和 Drama 两类电影的数量的变化。

通关必备：

1. 使用pd.to_datetime()把movie_data['release_date']转换成datetime格式。
2. movie_data['release_date'].dt.year可以访问年份，movie_data['release_date'].dt.month可以访问月份
3. 设置年份筛选条件，可以使用 .dt.year.between() 进行范围筛选。
4. 设置月份筛选条件， 可以使用 .dt.month ==进行特定值筛选。
5. 如果上一步的任务已经完成，可以直接使用上一步的筛选条件。
6. 设置电影类型筛选Comedy，由于电影类型是包含分割符的，一个电影可以是多种类型，所以可以使用 movie_data.genres.str.contains('Comedy')来筛选。
7. 设置电影类型筛选Drama，.str.contains('Drama')
8. 根据前3个筛选条件，对movie_data筛选数据，并按年份统计结果。可以使用 .release_year.value_counts().sort_index()。
9. 对结果进行画图,图中包含两个类型的数据，Comedy和Drama。

In [ ]:
```
# 筛选条件1 年份
select_years =
```

In [ ]:
```
# 筛选条件2 六月
select_month =
```

In [ ]:
```
# # 筛选条件3 类型 Comedy
select_comedy =
```

In [ ]:
```
# # 筛选条件3 类型 Drama
select_drama =
```

In [ ]:
```
# 筛选出Comedy数据
ret_comedy =
# 筛选出Drama数据
ret_drama =
```

In [ ]:
```
# 画图, 自由发挥吧
```

分析：

注意: 当你写完了所有的代码，并且回答了所有的问题。你就可以把你的 iPython Notebook 导出成 HTML 文件。你可以在菜单栏，这样导出**File -> Download as -> HTML (.html)、Python (.py)** 把导出的 HTML、python文件 和这个 iPython notebook 一起提交给审阅者。