Project 3: Hollywood Movie Data Analysis

Author: Karen Bevis

Table of Contents

Introduction

>

Welcome to this exploration of Hollywood movies! Here are the steps we'll be taking:

Step 1: Question. What do we want to figure out with this data? We will sift through over 10,000 movie titles in order to discover valuable relationships between variables such as revenues (determined by box office sales), budget, genres, and viewer popularity. We are especially curious about:

- What properties are associated with specific years and decades?
  - Over time, what genres are most popular?
  - Over time, has movie length (runtime) increased or decreased?
- What properties are associated with higher revenues?
  - Are specific genres associated with higher revenues?
  - Are movies with higher revenues more popular?
  - Does a larger budget correlate to higher revenues?
  - Do movies with higher revenues make more profits?

We will be directing our analysis towards finding answers to these questions, and in the process hopefully stumble across new insights as well.

Step 2: Data Wrangling. Gather, load, and assess the data. Make modifications, such as adding and replacing information and removing duplicates and extraneous data, to ensure our dataset is clean for analysis.

Step 3: Data Exploration. Augment the data, remove outliers, create better features, and find patterns. This step might lead us back to the first two steps, questioning and wrangling.

Step 4: Conclusions. Lastly we will summarize the relationships we found, make predictions, and present our findings visually.

Notes:

- Popularity is a value that gets updated daily and takes a number of things into account like site views, number of user ratings/watchlist/favourite additions and release date.
- Revenues and budgets are in US dollars.

Let's begin!

In [1]:
```
# import all libraries we'll want to use
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline
```

Data Wrangling

Gather, Assess, Clean

> In this step, we will gather and our data, a csv spreadsheet provided by Udacity, then load it into a dataframe to assess its quality. We will be looking for missing or errant, and problems in quality and/or structure. We will be removing extraneous data and making modifications, such as replacing information and removing duplicates, to ensure our dataset is trim and clean for analysis.

General Properties

In [2]:
```
# Gather data: load data and print out a few lines. Inspect datatypes an
d look for missing/ errant data.

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

Out[2]:

	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...	h
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	h
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	h
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	h e
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	h

5 rows × 21 columns

There is lots of great information here and also much that we are not concerned with. There are a few columns that have multiple strings of information that should be separated out. There's no data on profitability, I'll want to add that column.

In [3]:
```
# Assess number of rows and columns of dataset
df.shape
```

Out[3]:
```
(10866, 21)
```

In [4]:
```
# Assess summary of dataset, including datatypes, and check for missing
 data.
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
```

There is missing row data for many columns, but I plan on removing these columns since they aren't directly relevant to our questions. I'll revisit missing data once the dataset is trimmed. Next we'll assess statistics for the columns.

In [5]:
```
# assess statistics for each column
df.describe()
```

Out[5]:

	id	popularity	budget	revenue	runtime	vote_
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.0
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.0000
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.0000
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.0000
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.00

Notable findings:

- Popularity ranges from 0 - 33, but has an average of .6 (33 could be an outlier?)
- Votes range from 1.5 to 9.2 (probably on a scale of 1-10), with an average of 6
- Budget (usd) ranges from approx. 0 - 425 million (average 17.6 million)
- Revenue (usd) ranges from approx. 0 - 2.8 billion (average 51.4 million)
- Release years range from 1960 - 2015 (average 2001, most were released after 1995)

Next, we'll clean our data.

Data Cleaning

>

Now we'll make modifications to our dataset. First we'll remove extraneous data and duplicates, then add and replace information to ensure our dataset is clean for analysis.

- I'll drop extraneous columns that aren't relevant to our analysis.
- I'm most concerned with budget and revenue rather than those columns adjusted for inflation, but will keep it in the dataset in case I want to compare.
- I am dropping release date since I'm more interested in the release year.
- I'll keep the id here in case I want to merge with another dataset.

In [6]:
```
df.drop(['imdb_id', 'homepage', 'tagline', 'keywords', 'overview', 'prod
uction_companies', 'release_date'], axis=1, inplace=True)
df.head()
```

Out[6]:

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

The correct columns were removed. Next we'll assess if there are any duplicates.

In [7]:
```
sum(df.duplicated())
```

Out[7]: 1

There is only 1 duplicate, so we'll drop that row and perform 2 checks to ensure the duplicates were removed.

In [8]:
```
df.drop_duplicates(inplace=True)
print(sum(df.duplicated()))
print(df.shape)
```

```
0
(10865, 14)
```

There are no longer any duplicates and the dataset now has one less row. Next, I'll assess if any rows have missing values.

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
id                 0
popularity         0
budget             0
revenue            0
original_title     0
cast              76
director          44
runtime            0
genres            23
vote_count         0
vote_average       0
release_year       0
budget_adj         0
revenue_adj        0
dtype: int64
```

Let's view the rows with missing information to assess if it's ok to drop. I'd like to order by runtime to get a sense if these are full feature length films.

In [10]:
```
df[df.isnull().any(axis=1)].sort_values(['runtime'], ascending=True)
```

Out[10]:

	id	popularity	budget	revenue	original_title	cast	
2370	127717	0.081892	0	0	Freshman Father	Britt Irvin|Merrilyn Gann|Barbara Tyson|Anthon...	Michae
2315	48373	0.171615	0	0	Listen to Your Heart	Alexia Rasmussen|Kent Moran|Cybill Shepherd|Sh...	NaN
1241	296370	0.135376	0	0	Dance-Off	Kathryn McCormick|Shane Harper|Finola Hughes|C...	NaN
4883	142563	0.078472	0	0	Fresh Guacamole	NaN	PES
4890	126909	0.083202	0	0	Cousin Ben Troop Screening	Jason Schwartzman	Wes An
7905	13924	0.647261	0	0	The Adventures of AndrÃ© and Wally B.	NaN	Alvy Ra
10754	3171	0.002757	0	0	Bambi Meets Godzilla	NaN	Marv N
10550	13925	0.306425	0	0	Luxo Jr.	NaN	John L
5934	200204	0.067433	0	0	Prada: Candy	Peter Gadiot|Rodolphe Pauly|LÃ©a Seydoux	Wes Anders Coppo
6930	53215	0.076078	0	0	Kiwi!	NaN	Dony F
9251	13928	0.471351	0	0	Knick Knack	NaN	John L
9677	13926	0.253376	0	0	Red's Dream	NaN	John L
2221	48832	0.281852	0	0	Scott Pilgrim vs. the Animation	Michael Cera|Alison Pill|Georgette Perna|Mae W...	NaN
6374	13933	0.392879	0	0	One Man Band	NaN	Mark Andrew Jimene
10434	48784	0.146906	200	0	Six Men Getting Sick	NaN	David I

	id	popularity	budget	revenue	original_title	cast	
4818	116440	0.150035	0	0	Maggie Simpson in The Longest Daycare	NaN	David S
9799	48847	0.175008	0	0	The Amputee	Catherine E. Coulson|David Lynch	David L
6736	13060	0.434986	0	0	Lifted	NaN	Gary R
1177	269711	0.153047	0	0	JohnnyExpress	NaN	Kyungı
9529	13927	0.236514	0	0	Tin Toy	NaN	John L
6530	168891	0.092724	0	0	Saw Rebirth	Whit Anderson|Stan Kirsch|Jeff Shuter|George W...	Jeff Shuter Viney
9755	48714	0.046272	0	0	The Big Shave	NaN	Martin
3902	124277	0.013745	0	0	The Maker	NaN	Christo Kezelo
1899	31160	0.027045	0	0	Alma	NaN	Rodrig
6760	38580	0.371028	0	0	The Little Matchgirl	NaN	Roger
371	345637	0.422901	0	0	Sanjay's Super Team	NaN	Sanjay
600	332479	0.047256	0	0	Star Wars: TIE Fighter	NaN	Paul Jo
3097	10378	0.256180	150000	0	Big Buck Bunny	NaN	Sacha Goede
8824	48617	0.191631	0	0	Father and Daughter	NaN	Michae de Wit
6514	98622	0.128484	0	0	9	NaN	Shane
...	...	...	...	...	...	...	...
6033	238185	0.048587	0	0	Russell Brand: Messiah Complex	Russell Brand	NaN
9564	24348	0.168545	2500000	589244	Powaqqatsi	NaN	Godfre
7650	12172	0.383253	0	0	Encounters at the End of the World	NaN	Werner

	id	popularity	budget	revenue	original_title	cast	
4732	139463	0.235911	0	0	The Scapegoat	Andrew Scott|Jodhi May|Eileen Atkins|Matthew R...	Charle Sturrid
1236	250665	0.093062	0	0	No No: A Dockumentary	NaN	Jeffrey
3907	70845	0.004886	0	0	There's Something Wrong with Aunt Diane	NaN	Liz Ga
1852	133365	0.256703	0	0	The Diary of Anne Frank	Ellie Kendrick|Kate Ashfield|Geoff Breton|Feli...	NaN
424	363869	0.244648	0	0	Belli di papÃ	Diego Abatantuono|Matilde Gioli|Andrea Pisani|...	Guido
1316	245158	0.007622	0	0	Kids for Cash	NaN	Robert
6024	159012	0.080336	0	0	Narco Cultura	NaN	Shaul S
8234	56804	0.028874	0	0	Viaggi di nozze	Carlo Verdone|Claudia Gerini|Veronica Pivetti|...	Carlo V
587	319091	0.062536	0	0	The Hunting Ground	NaN	Kirby D
9307	141859	0.094652	0	0	Goldeneye	Charles Dance|Phyllis Logan|Patrick Ryecart|La...	Don Bo
556	321160	0.100910	0	0	With This Ring	Regina Hall|Jill Scott|Eve|Brooklyn Sudano|Dei...	NaN
6078	376823	0.002647	0	0	Trophy Kids	NaN	Chris E
2853	57892	0.130018	0	0	Vizontele	YÄ±lmaz ErdoÄŸan|Demet Akbag|Altan Erkekli|Cem...	YÄ±lm ErdoÄ
7723	13016	0.197715	7000000	0	Zeitgeist	NaN	Peter J

	id	popularity	budget	revenue	original_title	cast	
3365	22258	0.002475	0	0	Foo Fighters: Live at Wembley Stadium	Dave Grohl|Nate Mendel|Chris Shiflett|Pat Smea...	NaN
4872	269177	0.090552	0	0	Party Bercy	Florence Foresti	NaN
3339	13180	0.067761	0	0	Zeitgeist: Addendum	NaN	Peter J
6043	190940	0.039080	0	0	Bombay Talkies	Aamir Khan|Rani Mukerji|Randeep Hooda|Saqib Sa...	Anurag Kashya Banerj Akhtar
7813	22887	0.065543	6000	6000	Loose Change: Final Cut	NaN	Dylan A
1872	26379	0.091395	3250000	0	Paa	Amitabh Bachchan|Abhishek Bachchan|Vidya Balan...	NaN
465	321109	0.201696	0	0	Bitter Lake	NaN	Adam
3276	15467	0.147657	4180000	11000000	Kismat Konnection	Shahid Kapoor|Vidya Balan|Juhi Chawla|Om Puri|...	NaN
2401	45644	0.067753	0	0	Opeth: In Live Concert At The Royal Albert Hall	Mikael Ã… kerfeldt|Martin "Axe" Axenrot|Martin ...	NaN
3224	20313	0.224721	0	0	John Mayer: Where the Light Is Live in Los Ang...	John Mayer|Steve Jordan|Pino Palladino|David R...	NaN
4547	123024	0.520520	0	0	London 2012 Olympic Opening Ceremony: Isles of...	Queen Elizabeth II|Mike Oldfield|Kenneth Brana...	Danny
4939	168219	0.003183	0	0	The Men Who Built America	NaN	NaN
6181	18729	0.000065	0	0	North and South, Book I	Patrick Swayze|Philip Casnoff|Kirstie Alley|Ge...	NaN

134 rows × 14 columns

Looks like these movies are a combination of shorts and features films. I'm comfortable with removing the rows that have no director, cast, and/or genre. After doing this I'll check the dataset again to ensure there is no missing information (should return False) and the new dataset info.

In [11]:
```
df.dropna(inplace=True)
print(df.isnull().sum().any())
print(df.info())
```

```
False
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10731 entries, 0 to 10865
Data columns (total 14 columns):
id                10731 non-null int64
popularity        10731 non-null float64
budget            10731 non-null int64
revenue           10731 non-null int64
original_title    10731 non-null object
cast              10731 non-null object
director          10731 non-null object
runtime           10731 non-null int64
genres            10731 non-null object
vote_count        10731 non-null int64
vote_average      10731 non-null float64
release_year      10731 non-null int64
budget_adj        10731 non-null float64
revenue_adj       10731 non-null float64
dtypes: float64(4), int64(6), object(4)
memory usage: 1.2+ MB
None
```

There are now 10,731 rows and 14 columns. All columns now have the full amount of rows. If we wanted to replace nulls with mean values: df.fillna(df.mean(), inplace=True)

Additional cleaning checks:

- Do columns need renaming? No (no spaces and all lowercase)
- Do datatypes need converting? No, could convert budget and revenue from float to int but doesn't make a difference so will keep as is.
- Let's review popularity, vote count, and vote average more closely. Remove any outliers?

In [12]:
```
df[['original_title','popularity', 'vote_count', 'vote_average']].sort_values('popularity', ascending=False).head(25)
```

Out[12]:

	original_title	popularity	vote_count	vote_average
0	Jurassic World	32.985763	5562	6.5
1	Mad Max: Fury Road	28.419936	6185	7.1
629	Interstellar	24.949134	6498	8.0
630	Guardians of the Galaxy	14.311205	5612	7.9
2	Insurgent	13.112507	2480	6.3
631	Captain America: The Winter Soldier	12.971027	3848	7.6
1329	Star Wars	12.037933	4428	7.9
632	John Wick	11.422751	2712	7.0
3	Star Wars: The Force Awakens	11.173104	5292	7.5
633	The Hunger Games: Mockingjay - Part 1	10.739009	3590	6.6
634	The Hobbit: The Battle of the Five Armies	10.174599	3110	7.1
1386	Avatar	9.432768	8458	7.1
1919	Inception	9.363643	9767	7.9
4	Furious 7	9.335014	2947	7.3
5	The Revenant	9.110700	3929	7.2
2409	Fight Club	8.947905	5923	8.1
635	Big Hero 6	8.691294	4185	7.8
6	Terminator Genisys	8.654359	2598	5.8
2633	The Lord of the Rings: The Fellowship of the Ring	8.575419	6079	7.8
2875	The Dark Knight	8.466668	8432	8.1
3371	Underworld: Endless War	8.411577	21	5.9
636	The Imitation Game	8.110711	3478	8.0
3911	The Lord of the Rings: The Two Towers	8.095275	5114	7.8
4177	Pulp Fiction	8.093754	5343	8.1
2634	Harry Potter and the Philosopher's Stone	8.021423	4265	7.2

In [13]:
```
df[['original_title','popularity', 'vote_count', 'vote_average']].sort_values('popularity', ascending=False).tail()
```

Out[13]:

	original_title	popularity	vote_count	vote_average
7268	Born into Brothels	0.001117	23	6.4
6961	Khosla Ka Ghosla!	0.001115	10	6.8
6551	Mon petit doigt m'a dit...	0.000973	13	5.7
6080	G.B.F.	0.000620	82	6.1
9977	The Hospital	0.000188	10	6.4

- Doesn't look like there are outliers for popularity, just a wide range of values. Will keep as is.
- For vote average, some only have 10 votes while others have upwards of 10,000. Review the vote_count column while exploring the data, might want to weight averages with higher vote counts since it's a larger sample size.

I'd like to add a profit column so we can create a profitability ratio.

Profit = revenue (aka income) - budget (aka cost or expense)

In [14]:
```
df['profit'] = df['revenue'] - df['budget']
df.head()
```

Out[14]:

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

In [16]:
```
# make sure no negative numbers for profit
df.loc[df['profit'] < 0, 'profit'] = 0
```

Now that we have profit column, we can create a profitability ratio column.

Profitability ratio = (profit/revenue) x 100 = percentage

I'll adjust for non-zero division by adding .0001 to the denominator, revenue. I'll convert this column from float to integer so we have non-decimal values ranging from 1-100.

In [17]:
```
df['profitability_ratio'] = (df['profit'] / (df['revenue'] + .0001)) * 1
00
df['profitability_ratio'] = df['profitability_ratio'].astype(int)

df.sort_values(['profitability_ratio'], ascending=False).tail()
```

Out[17]:

	id	popularity	budget	revenue	original_title	cast	director	ru
5826	178682	0.251933	0	0	The Wizards Return: Alex vs. Alex	Selena Gomez|Jake T. Austin|Jennifer Stone|Mar...	Victor Gonzalez	60
5825	186988	0.257366	0	0	Miele	Jasmine Trinca|Carlo Cecchi|Libero De Rienzo|V...	Valeria Golino	96
5824	207780	0.258887	0	0	Antisocial	Michelle Mylett|Cody Ray Thompson|Adam Christi...	Cody Calahan	90
5823	205891	0.260546	0	0	The Demented	Kayla Ewell|Brittney Alger|Sarah Butler|Michae...	Christopher Roosevelt	92
10865	22293	0.035919	19000	0	Manos: The Hands of Fate	Harold P. Warren|Tom Neyman|John Reynolds|Dian...	Harold P. Warren	74

In [18]:
```
print(df['profitability_ratio'].nunique())
```

```
100
```

Before I removed the negative numbers for profits, the profitability ratio was off: I was seeing 936 rather than a max of 100 unique values. The problem appears to be with the negative numbers so this problem no longer exists, but stil wanted to check that there is a max value of 100 for the ratio, and no downside of replacing the profitability ratio negative number to zero, even if there aren't any.

In [19]:
```
df.loc[df['profitability_ratio'] < 0, 'profitability_ratio'] = 0
print(df['profitability_ratio'].nunique())
```

```
100
```

Great, this columns is now clean. Next I'll create a new column, revenue_rating, to splice the revenue column into groups: low (under a million), mediun (millions), and high (billions).

In [20]:
```
bin_edges = [0, 1e+06, 1e+09, 2.827124e+09]
bin_names = ['under_million', 'millions', 'billions']
df['revenue_rating'] = pd.cut(df['revenue'], bin_edges, labels=bin_names
)
df.head()
```

Out[20]:

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

In [21]:
```
df['revenue_rating'].value_counts()
```

Out[21]:
```
millions         4298
under_million     523
billions           22
Name: revenue_rating, dtype: int64
```

In [22]:
```
df.isnull().sum()
```

Out[22]:
```
id                      0
popularity              0
budget                  0
revenue                 0
original_title          0
cast                    0
director                0
runtime                 0
genres                  0
vote_count              0
vote_average            0
release_year            0
budget_adj              0
revenue_adj             0
profit                  0
profitability_ratio     0
revenue_rating       5888
dtype: int64
```

To clean up the revenue rating, let's make all the rows with null values 0 since those rows have no revenue or budget:

In [23]:
```
df.revenue_rating.fillna('under_million', inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10731 entries, 0 to 10865
Data columns (total 17 columns):
id                    10731 non-null int64
popularity            10731 non-null float64
budget                10731 non-null int64
revenue               10731 non-null int64
original_title        10731 non-null object
cast                  10731 non-null object
director              10731 non-null object
runtime               10731 non-null int64
genres                10731 non-null object
vote_count            10731 non-null int64
vote_average          10731 non-null float64
release_year          10731 non-null int64
budget_adj            10731 non-null float64
revenue_adj           10731 non-null float64
profit                10731 non-null int64
profitability_ratio   10731 non-null int64
revenue_rating        10731 non-null category
dtypes: category(1), float64(4), int64(8), object(4)
memory usage: 1.4+ MB
```

The release years range from 1960 to 2015. I'll create a column for all the decades.

In [24]:
```
bin_edges = [1959, 1970, 1980, 1990, 2000, 2010, 2015]
bin_names = ['sixties', 'seventies', 'eighties', 'nineties', 'two_thousa
nds', 'two_thousand_tens']
df['decades'] = pd.cut(df['release_year'], bin_edges, labels=bin_names)
df.head()
```

Out[24]:

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

Check to make sure both newly created columns look good.

In [25]:
```
df[['release_year', 'decades']].head()
```

Out[25]:

	release_year	decades
0	2015	two_thousand_tens
1	2015	two_thousand_tens
2	2015	two_thousand_tens
3	2015	two_thousand_tens
4	2015	two_thousand_tens

Let's now deal with the columns that have multiple values per cell, starting with genres. I'll create separate dataframes for each: genres, cast, and director, in case we want the original df intact.

In [26]:
```
df['genres'].str.contains('|')
df['genres'].nunique()
```

Out[26]: 2022

I want to split up the genres column cells so we can tally each genre individually. Next I'll remove the 'genres' column (with multiple values) and replace it with a 'genre' column (with single values). Then I'll make sure that there is a new row for each genre (stacked), so there will be multiple rows with the same original_title.

In [27]:
```
df_split_genre = df.copy()

split_genre = df_split_genre['genres'].str.split('|').apply(pd.Series, 1
).stack().reset_index(level=1, drop=True)
split_genre.name = 'genre_split'
df_split_genre = df_split_genre.drop(['genres'], axis=1).join(split_genr
e)
```

The genres now appear to be split up and stacked, let's check to make sure that the new genre column contains only single values.

In [28]:
```
df_split_genre['genre_split'].unique()
```

Out[28]:
```
array(['Action', 'Adventure', 'Science Fiction', 'Thriller', 'Fantasy',
       'Crime', 'Western', 'Drama', 'Family', 'Animation', 'Comedy',
       'Mystery', 'Romance', 'War', 'History', 'Music', 'Horror',
       'Documentary', 'TV Movie', 'Foreign'], dtype=object)
```

Let's check for duplicates and view the info for our new dataset.

In [29]:
```
print(df_split_genre.info())
print(df_split_genre.shape)
print(sum(df_split_genre.duplicated()))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26753 entries, 0 to 10865
Data columns (total 18 columns):
id                    26753 non-null int64
popularity            26753 non-null float64
budget                26753 non-null int64
revenue               26753 non-null int64
original_title        26753 non-null object
cast                  26753 non-null object
director              26753 non-null object
runtime               26753 non-null int64
vote_count            26753 non-null int64
vote_average          26753 non-null float64
release_year          26753 non-null int64
budget_adj            26753 non-null float64
revenue_adj           26753 non-null float64
profit                26753 non-null int64
profitability_ratio   26753 non-null int64
revenue_rating        26753 non-null category
decades               26753 non-null category
genre_split           26753 non-null object
dtypes: category(2), float64(4), int64(8), object(4)
memory usage: 3.5+ MB
None
(26753, 18)
0
```

We now have 26,753 rows (from 10,000) and 14 columns (same), which makes sense, and no duplicate rows. Let's check for any null values.

In [30]:
```
df_split_genre.isnull().sum()
```

Out[30]:
```
id                    0
popularity            0
budget                0
revenue               0
original_title        0
cast                  0
director              0
runtime               0
vote_count            0
vote_average          0
release_year          0
budget_adj            0
revenue_adj           0
profit                0
profitability_ratio   0
revenue_rating        0
decades               0
genre_split           0
dtype: int64
```

Looks good. Now, I'll repeat the process with the other columns with multiple values: cast and director. However, I will create copies of the original df and apply these separately so the processing power is not slowed too much.

In [31]:
```
df_split_cast = df.copy()

split_cast = df_split_cast['cast'].str.split('|').apply(pd.Series, 1).st
ack().reset_index(level=1, drop=True)
split_cast.name = 'cast_split'
df_split_cast = df_split_cast.drop(['cast'], axis=1).join(split_cast)

df_split_director = df.copy()

split_director = df_split_director['director'].str.split('|').apply(pd.S
eries, 1).stack().reset_index(level=1, drop=True)
split_director.name = 'director_split'
df_split_director = df_split_director.drop(['director'], axis=1).join(sp
lit_director)
```

In [32]:
```
df_split_genre.head()
```

Out[32]:

	id	popularity	budget	revenue	original_title	cast	director	run
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow	124
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow	124
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow	124
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	Colin Trevorrow	124
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	George Miller	120

In [33]:
```
df_split_cast.head()
```

Out[33]:

	id	popularity	budget	revenue	original_title	director	runtime	
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adv Fiction|Thı
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adv Fiction|Thı
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adv Fiction|Thı
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adv Fiction|Thı
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adv Fiction|Thı

In [34]:
```
df_split_director.head()
```

Out[34]:

	id	popularity	budget	revenue	original_title	cast	runtime	
0	135397	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	124	Ac Fic
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...	120	Ac Fic
2	262500	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...	119	Ad Fic
3	140607	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	136	Ac Fic
4	168259	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	137	Ac

In [35]:
```
df_split_genre.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26753 entries, 0 to 10865
Data columns (total 18 columns):
id                    26753 non-null int64
popularity            26753 non-null float64
budget                26753 non-null int64
revenue               26753 non-null int64
original_title        26753 non-null object
cast                  26753 non-null object
director              26753 non-null object
runtime               26753 non-null int64
vote_count            26753 non-null int64
vote_average          26753 non-null float64
release_year          26753 non-null int64
budget_adj            26753 non-null float64
revenue_adj           26753 non-null float64
profit                26753 non-null int64
profitability_ratio   26753 non-null int64
revenue_rating        26753 non-null category
decades               26753 non-null category
genre_split           26753 non-null object
dtypes: category(2), float64(4), int64(8), object(4)
memory usage: 3.5+ MB
```

In [36]:
```
df_split_cast.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 52334 entries, 0 to 10865
Data columns (total 18 columns):
id                    52334 non-null int64
popularity            52334 non-null float64
budget                52334 non-null int64
revenue               52334 non-null int64
original_title        52334 non-null object
director              52334 non-null object
runtime               52334 non-null int64
genres                52334 non-null object
vote_count            52334 non-null int64
vote_average          52334 non-null float64
release_year          52334 non-null int64
budget_adj            52334 non-null float64
revenue_adj           52334 non-null float64
profit                52334 non-null int64
profitability_ratio   52334 non-null int64
revenue_rating        52334 non-null category
decades               52334 non-null category
cast_split            52334 non-null object
dtypes: category(2), float64(4), int64(8), object(4)
memory usage: 6.9+ MB
```

In [37]:
```
df_split_director.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11774 entries, 0 to 10865
Data columns (total 18 columns):
id                    11774 non-null int64
popularity            11774 non-null float64
budget                11774 non-null int64
revenue               11774 non-null int64
original_title        11774 non-null object
cast                  11774 non-null object
runtime               11774 non-null int64
genres                11774 non-null object
vote_count            11774 non-null int64
vote_average          11774 non-null float64
release_year          11774 non-null int64
budget_adj            11774 non-null float64
revenue_adj           11774 non-null float64
profit                11774 non-null int64
profitability_ratio   11774 non-null int64
revenue_rating        11774 non-null category
decades               11774 non-null category
director_split        11774 non-null object
dtypes: category(2), float64(4), int64(8), object(4)
memory usage: 1.5+ MB
```

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10731 entries, 0 to 10865
Data columns (total 18 columns):
id                    10731 non-null int64
popularity            10731 non-null float64
budget                10731 non-null int64
revenue               10731 non-null int64
original_title        10731 non-null object
cast                  10731 non-null object
director              10731 non-null object
runtime               10731 non-null int64
genres                10731 non-null object
vote_count            10731 non-null int64
vote_average          10731 non-null float64
release_year          10731 non-null int64
budget_adj            10731 non-null float64
revenue_adj           10731 non-null float64
profit                10731 non-null int64
profitability_ratio   10731 non-null int64
revenue_rating        10731 non-null category
decades               10731 non-null category
dtypes: category(2), float64(4), int64(8), object(4)
memory usage: 1.4+ MB
```

We now have 4 clean dataframes: df, df_split_genre, df_split_cast, and df_split_director. I'll save them below.

Let's move on to exploring and augmenting our data. I'd like to add 2 new categories, profit and profitability ratio in order to compare this to revenue, and a new column that splits up revenue into 3 categories: under a million, millions, and billions. I'll also want to view decades over individual years but will create that filter in the next section.

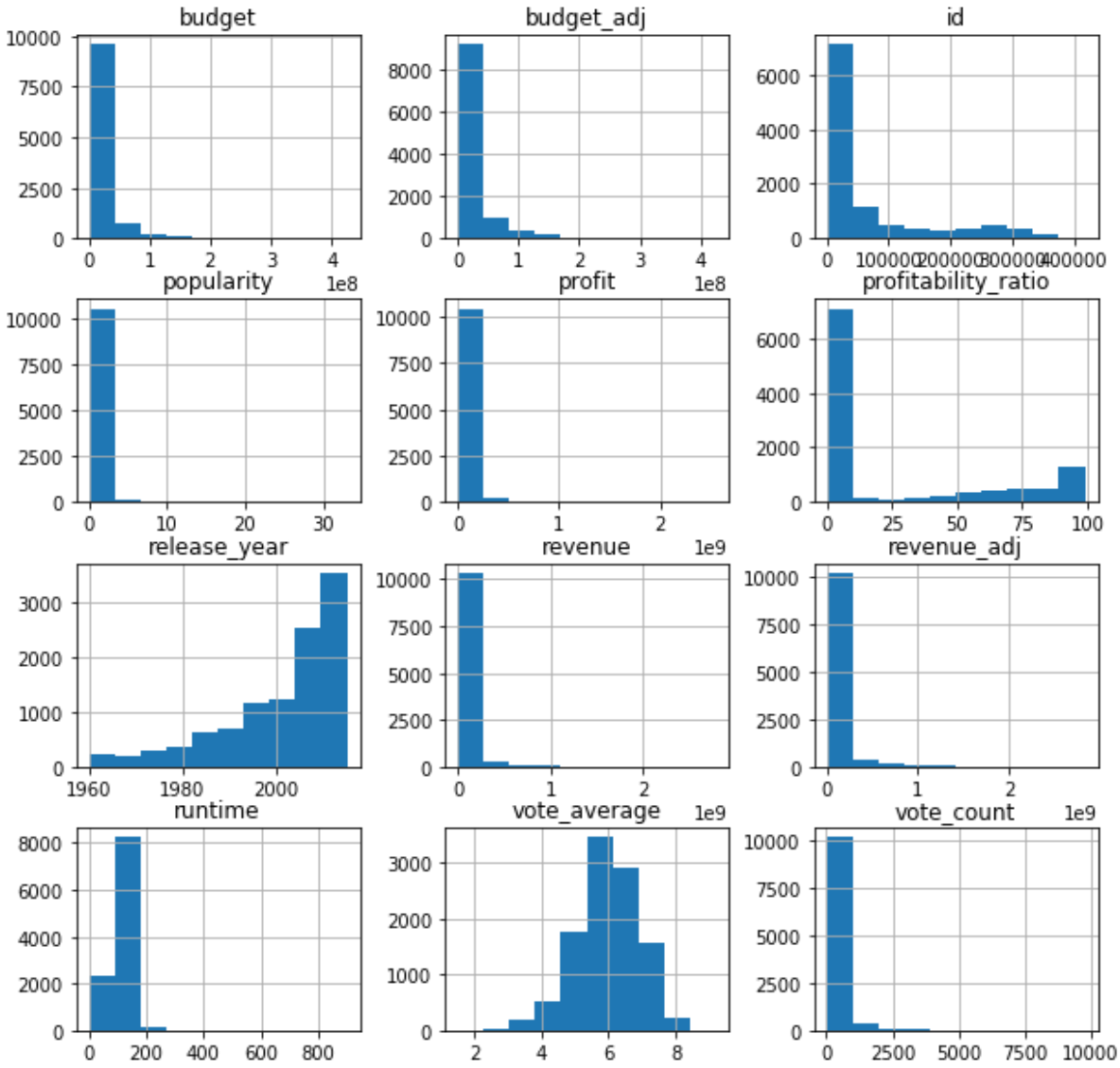
In [39]:
```
df.to_csv('tmbd-movies-clean.csv', index=False)
df_split_genre.to_csv('tmbd-movies-genre.csv', index=False)
df_split_cast.to_csv('tmbd-movies-cast.csv', index=False)
df_split_director.to_csv('tmbd-movies-director.csv', index=False)
```

Exploratory Data Analysis

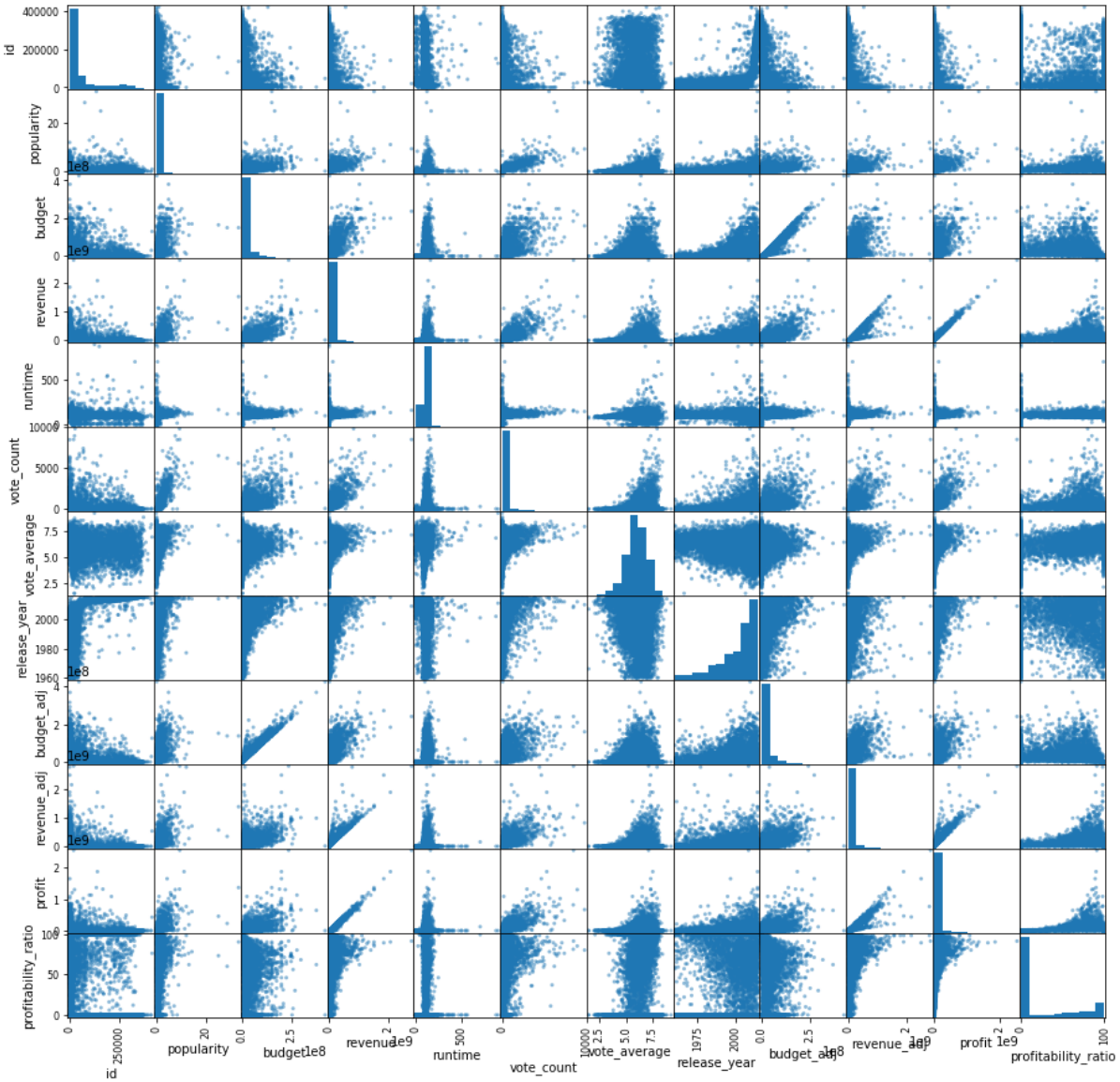
Now that we've trimmed and cleaned our data, let's move on to exploration. In this section, we'll compute statistics and create visualizations with to address our questions.

Let's first view all columns with numerical data with a histogram:

In [40]:
```
df.hist(figsize=(10,10));
```



In [41]:
```
pd.plotting.scatter_matrix(df, figsize=(15, 15));
```

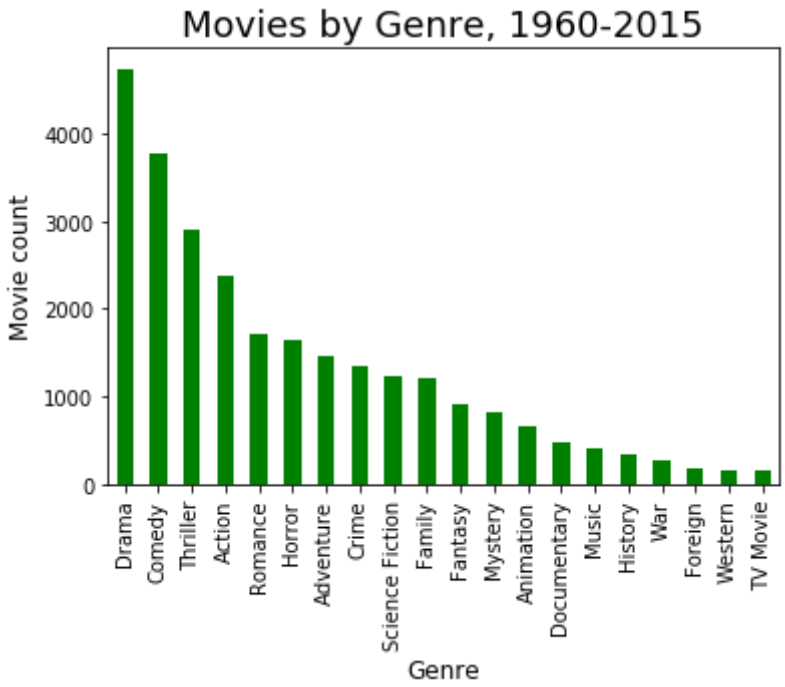


At first glance, revenue and profits are positively correlated.

Movies throughout the years:

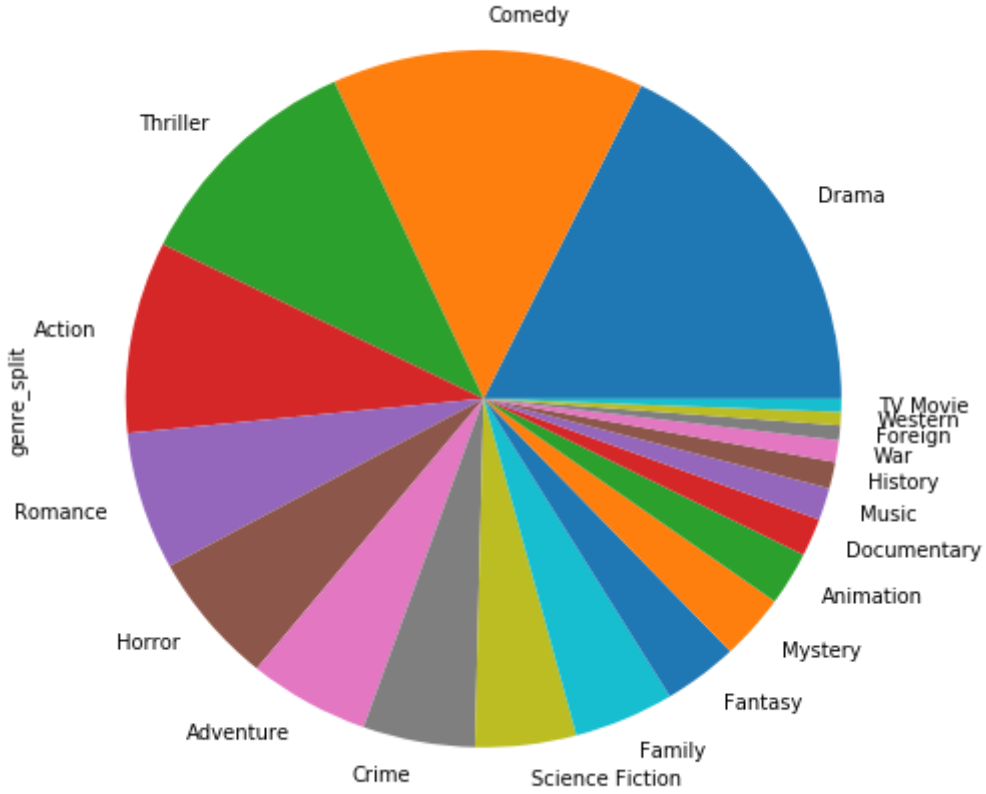
What genres are most popular overall?

In [42]:
```
df_split_genre['genre_split'].value_counts().plot(kind='bar', color='g'
);
plt.title('Movies by Genre, 1960-2015', size=18)
plt.xlabel('Genre', size=12)
plt.ylabel('Movie count', size=12);
```



In [43]:
```
# also view with a pie chart
df_split_genre['genre_split'].value_counts().plot(kind='pie', figsize=(8
,8))
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1a236642b0>



Drama, Comedy, Thriller, and Action are the most popular genres in general. The pie chart is a better visual since we can assess that these top 4 genres make up about 50% of all movies. TV Movies, Westerns, and Foreigns are the least popular genres.

Do these genres hold true throughout the decades?

What genres are most popular throughout the decades?

In [44]:
```
genres_decades = df_split_genre.groupby(['decades'])['genre_split'].value_counts()
genres_decades.groupby(level=0).nlargest(3).reset_index(level=0, drop=True)
```

Out[44]:

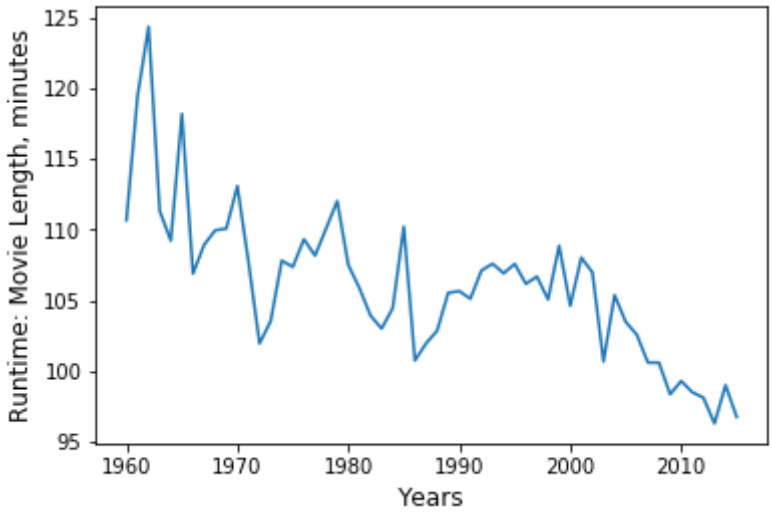
decades	genre_split	
sixties	Drama	186
	Comedy	124
	Action	89
seventies	Drama	251
	Thriller	168
	Action	137
eighties	Comedy	451
	Drama	449
	Action	283
nineties	Drama	902
	Comedy	785
	Thriller	512
two_thousands	Drama	1717
	Comedy	1421
	Thriller	1042
two_thousand_tens	Drama	1241
	Comedy	860
	Thriller	830

Name: genre_split, dtype: int64

> Drama is the most popular genre for every decade except for the 80's which is Comedy. Thanks John Hughes!

Has the average movie length lengthened or shortened over time?

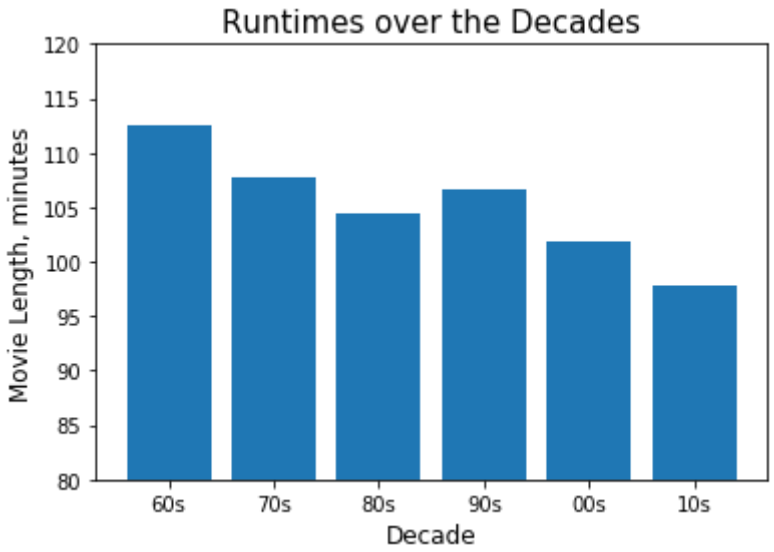
In [45]:
```
runtime = df.groupby('release_year')['runtime'].mean()
plt.plot(runtime)
plt.xlabel('Years', size=12)
plt.ylabel('Runtime: Movie Length, minutes', size=12);
```



Runtime has decreased throughout the years, from 118 minutes in 1960 to 97 minutes in 2015. Let's look at this data for the decades:

In [46]:
```
runtime_decade_locations = [1, 2, 3, 4, 5, 6]
runtime_decade_heights = df.groupby('decades')['runtime'].mean()
print(runtime_decade_heights)
labels = ['60s', '70s', '80s', '90s', '00s', '10s']
plt.bar(runtime_decade_locations, runtime_decade_heights, tick_label=lab
els)
plt.title('Runtimes over the Decades', size=15);
plt.xlabel('Decade', size=12)
plt.ylabel('Movie Length, minutes', size=12)
plt.ylim((80,120));  # view the y-coodinate more closely
```

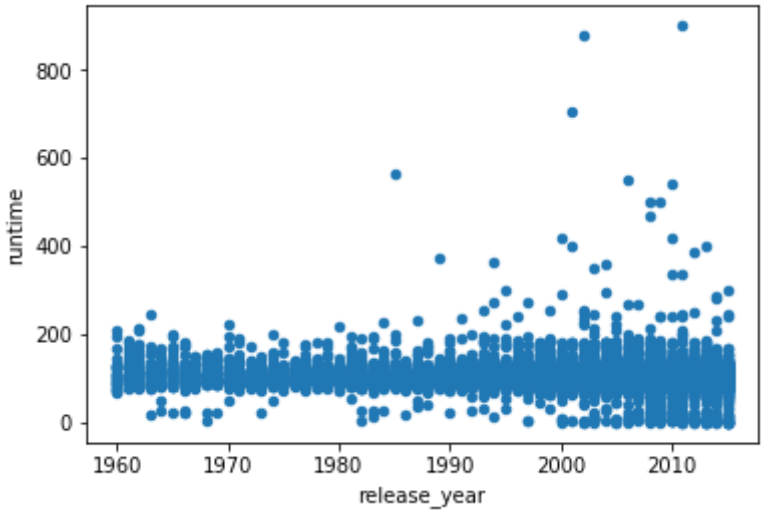
```
decades
sixties              112.535176
seventies            107.734807
eighties             104.360108
nineties             106.566955
two_thousands        101.923729
two_thousand_tens     97.723314
Name: runtime, dtype: float64
```



116 - 98 = 16 minutes / 116 = .16 (a 16% decrease).

Runtime has decreased by 16% from the 1960s through 2010's.

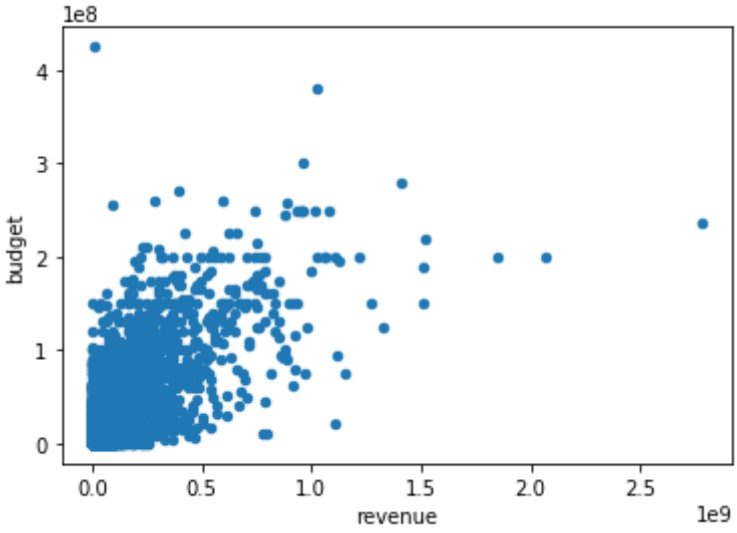
In [47]:
```
df.plot(x='release_year', y='runtime', kind='scatter');
```



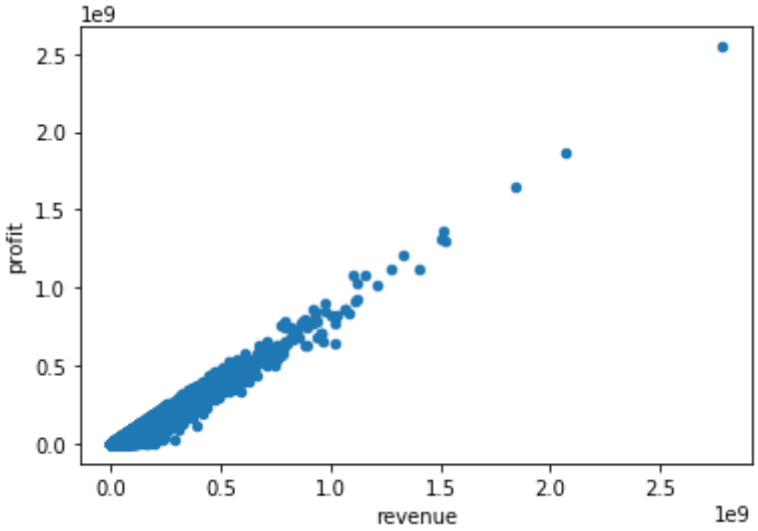
What properties are associated with higher revenues?

General scatter plots of revenue vs budget, profit, and popularity.

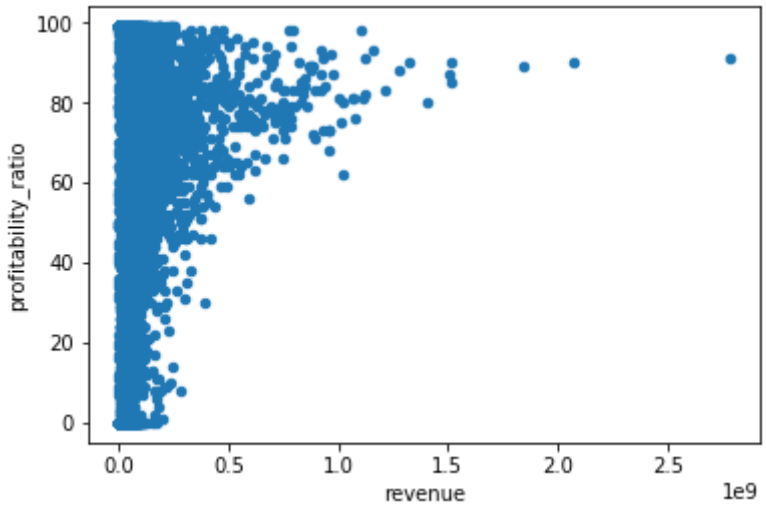
In [48]:
```
df.plot(x='revenue', y='budget', kind='scatter');
```



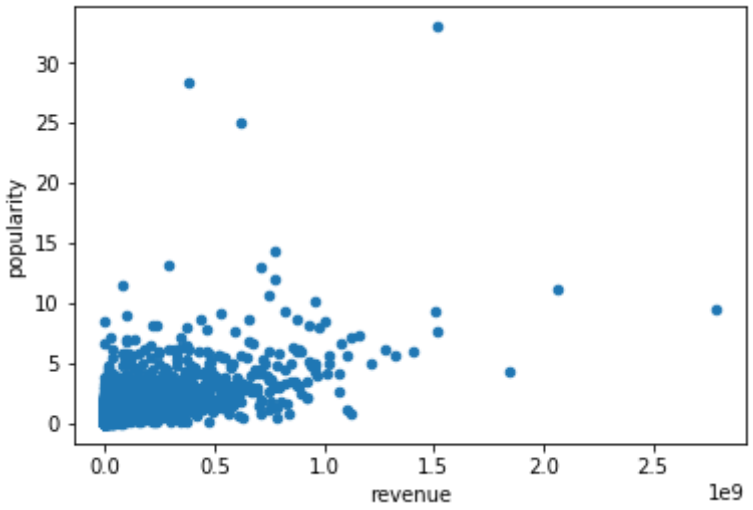
In [49]:
```
df.plot(x='revenue', y='profit', kind='scatter');
```



In [50]:
```
df.plot(x='revenue', y='profitability_ratio', kind='scatter');
```



In [51]:
```
df.plot(x='revenue', y='popularity', kind='scatter');
```

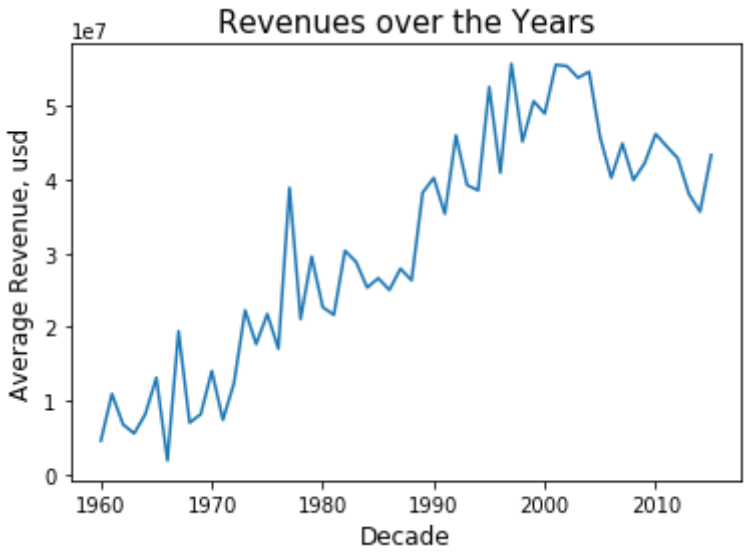


- Revenue and budget have a weak positive correlation.
- Revenue and profit have a strong positive correlation.
- Revenue and profitability have a weak positive correlation.
- Revenue and popularity have positive correlation, movies with higher revenues tend to be more popular.

Have revenues increased over time?

In [52]:
```
rev = df.groupby('release_year')['revenue'].mean()
print(df.groupby('decades')['revenue'].mean())
plt.plot(rev)
plt.title('Revenues over the Years', size=15);
plt.xlabel('Decade', size=12)
plt.ylabel('Average Revenue, usd', size=12);
```

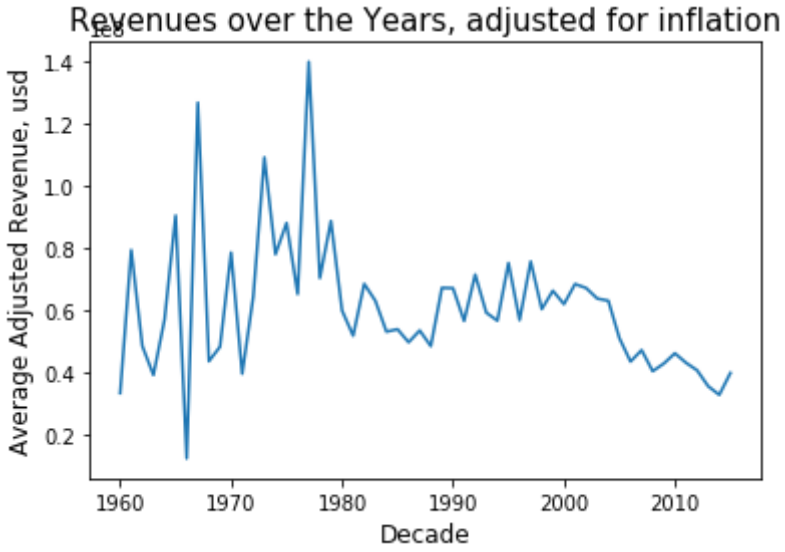
```
decades
sixties              8.985570e+06
seventies            2.154416e+07
eighties             2.954496e+07
nineties             4.578074e+07
two_thousands        4.654393e+07
two_thousand_tens    4.064386e+07
Name: revenue, dtype: float64
```



Revenues over the years have increased 391%, from 11 million to 54 million.

I'd expect the adjusted revenue to stay relatively constant:

In [53]:
```
# first lets look at general revenue throughout the years,
# I'd expect the original revenue to trend towards increasing, while the
 adjusted should stay relatively contant
revenue_adj = df.groupby('release_year')['revenue_adj'].mean()
plt.plot(revenue_adj)
plt.title('Revenues over the Years, adjusted for inflation', size=15);
plt.xlabel('Decade', size=12)
plt.ylabel('Average Adjusted Revenue, usd', size=12);
```



What variables are associated with higher revenues?

Are specific genres associated with higher revenues?

Find the top 100 movies in box office revenues to create filters (new dataframes) for the top grossing and below:

In [54]:
```
df_top_hundred = df.sort_values(by=['revenue'], ascending=False).head(10
0)
df_top_hundred['original_title'].nunique()
```

Out[54]: 100

In [55]:
```
df_top_hundred.describe()
```

Out[55]:

	id	popularity	budget	revenue	runtime	vote_count
count	100.000000	100.000000	1.000000e+02	1.000000e+02	100.000000	100.000000
mean	43314.780000	5.191795	1.542350e+08	8.979584e+08	130.070000	3475.060000
std	55833.708222	4.467371	6.694002e+07	3.189943e+08	25.168625	1839.603382
min	11.000000	0.436803	1.050000e+07	6.118994e+08	88.000000	201.000000
25%	673.750000	2.755044	1.122500e+08	7.097980e+08	110.250000	2245.750000
50%	12299.500000	4.264860	1.500000e+08	8.079637e+08	132.000000	3169.500000
75%	69080.750000	6.067551	2.000000e+08	9.590500e+08	144.250000	4266.000000
max	211672.000000	32.985763	3.800000e+08	2.781506e+09	201.000000	9767.000000

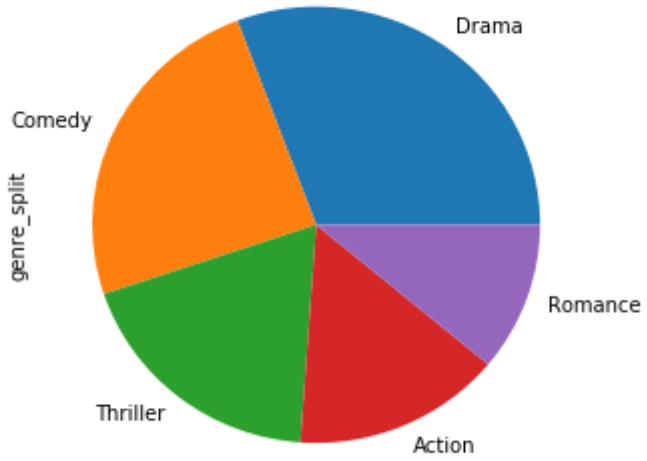
The min revenue for the top 100 movies is 6.118994e+08

Create two filters with the split_genre dataframe: The top 100 grossing movies and below

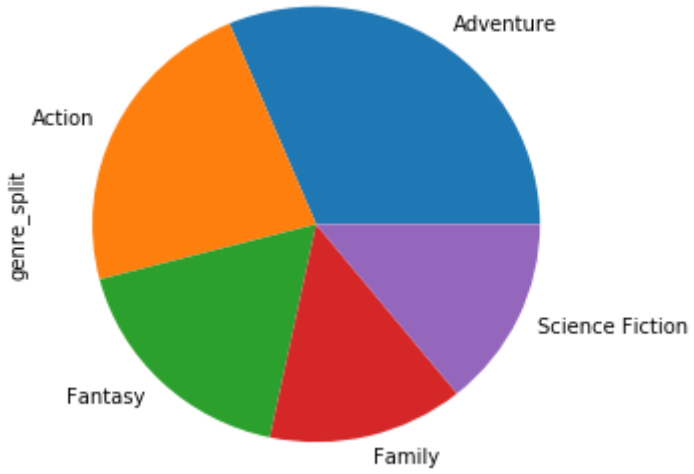
In [56]:
```
below_hundred = df_split_genre.query('revenue < 6.118994e+08')
top_hundred = df_split_genre.query('revenue >= 6.118994e+08')

print(below_hundred['genre_split'].value_counts().nlargest(5))
print(top_hundred['genre_split'].value_counts().nlargest(5))
```

```
Drama       4732
Comedy      3760
Thriller    2884
Action      2323
Romance     1701
Name: genre_split, dtype: int64
Adventure          73
Action             53
Fantasy            41
Family             33
Science Fiction    33
Name: genre_split, dtype: int64
```

In [57]:
```
below_hundred['genre_split'].value_counts().nlargest(5).plot(kind='pie',
 figsize=(5,5));
```



In [58]:
```
top_hundred['genre_split'].value_counts().nlargest(5).plot(kind='pie', f
igsize=(5,5));
```



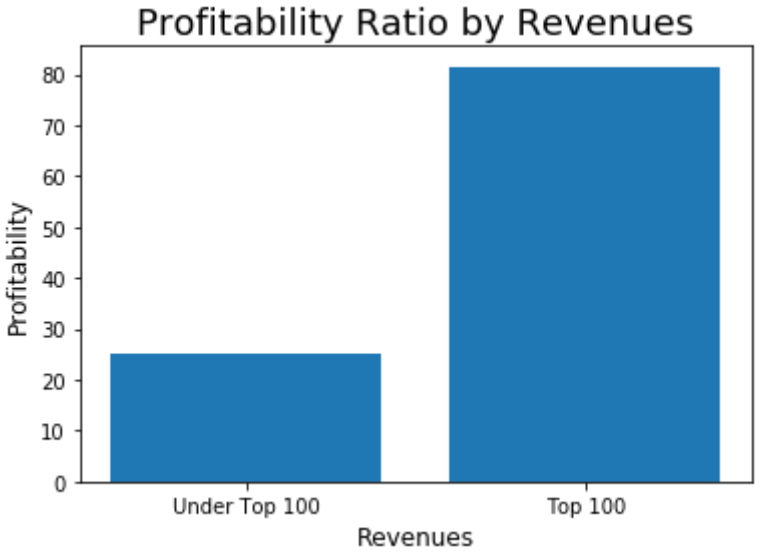
- Movies that make less revenue are Dramas, Comedies, and Thrillers.
- The top 100 revenue producing movies are Adventure, Action, and Fantasy.

Do higher revenue films make more profits?

In [59]:
```
print(below_hundred['popularity'].mean())
print(top_hundred['popularity'].mean())

print(below_hundred['profit'].mean())
print(top_hundred['profit'].mean())

print(below_hundred['profitability_ratio'].mean())
print(top_hundred['profitability_ratio'].mean())
```

```
0.6545952035033356
5.29254594392523
24674963.9686365
754018391.6728972
24.990428268765132
81.70404984423676
```

In [60]:
```
locations = [1, 2]
heights = [below_hundred['profitability_ratio'].mean(), df_top_hundred[
'profitability_ratio'].mean()]
labels = ['Under Top 100', 'Top 100']
plt.bar(locations, heights, tick_label=labels)
plt.title('Profitability Ratio by Revenues', size=18)
plt.xlabel('Revenues', size=12)
plt.ylabel('Profitability', size=12);
```



Yes, movies that do better at the box office have nearly double the profits as those that make less. Movies under the Top 100 have profitability ratio of 25%, while those in the Top 100 is 84%, a 59% increase.

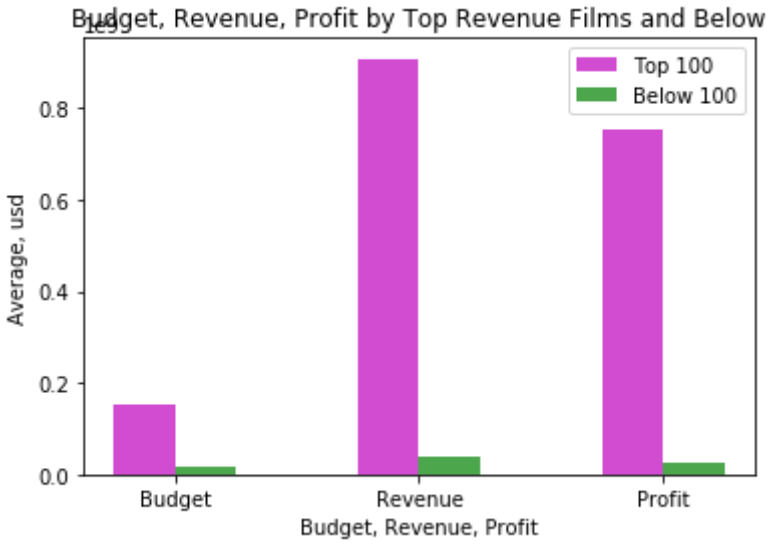
> Top revenue movies have higher budgets and much higher profits.

In [61]:
```
# break it down by the bins:
top_profits = df.groupby('revenue_rating')['profit'].mean()
top_profits
```

Out[61]:
```
revenue_rating
under_million    1.155967e+04
millions         6.497209e+07
billions         1.142942e+09
Name: profit, dtype: float64
```

In [63]:
```
# plot bars
top = top_hundred['budget'].mean(), top_hundred['revenue'].mean(), top_h
undred['profit'].mean()
bottom = below_hundred['budget'].mean(), below_hundred['revenue'].mean
(), below_hundred['profit'].mean()

print(top, bottom)

ind = np.arange(len(top))
width = 0.25
top_bars = plt.bar(ind, top, width, color='m', alpha=.7, label='Top 100'
)
bottom_bars = plt.bar(ind + width, bottom, width, color='g', alpha=.7, l
abel='Below 100')

# title and labels
plt.ylabel('Average, usd')
plt.xlabel('Budget, Revenue, Profit')
plt.title('Budget, Revenue, Profit by Top Revenue Films and Below')
locations = ind + width / 2  # xtick locations
labels = ['Budget', 'Revenue', 'Profit']  # xtick labels
plt.xticks(locations, labels)

# legend
plt.legend();
```

```
(153355140.18691587, 907373531.8598131, 754018391.6728972) (15988437.54
7555992, 37359860.67459897, 24674963.9686365)
```



Movies with higher revenues have correspondingly higher budgets, revenues, and profits.

Other Fun Facts:

Which actors have starred in the most movies?

```
In [64]: df_split_cast['cast_split'].value_counts().head(15)
```

```
Out[64]: Robert De Niro       72
         Samuel L. Jackson    71
         Bruce Willis         62
         Nicolas Cage         61
         Michael Caine        53
         Robin Williams       51
         John Cusack          50
         John Goodman         49
         Morgan Freeman       49
         Susan Sarandon       48
         Liam Neeson          48
         Julianne Moore       47
         Alec Baldwin         47
         Johnny Depp          46
         Tom Hanks            46
         Name: cast_split, dtype: int64
```

Who has directed the most movies?

```
In [65]: df_split_director['director_split'].value_counts().head(15)
```

```
Out[65]: Woody Allen            46
         Clint Eastwood         34
         Steven Spielberg       30
         Martin Scorsese        30
         Steven Soderbergh      23
         Ridley Scott           23
         Ron Howard             22
         Joel Schumacher        21
         Brian De Palma         20
         John Carpenter         19
         David Cronenberg       19
         Francis Ford Coppola   19
         Barry Levinson         19
         Wes Craven             19
         Robert Rodriguez       19
         Name: director_split, dtype: int64
```

What are the 10 most popular movies?

In [66]:
```
df[['popularity', 'original_title']].sort_values(by='popularity', ascending=False).head(10)
```

Out[66]:

	popularity	original_title
0	32.985763	Jurassic World
1	28.419936	Mad Max: Fury Road
629	24.949134	Interstellar
630	14.311205	Guardians of the Galaxy
2	13.112507	Insurgent
631	12.971027	Captain America: The Winter Soldier
1329	12.037933	Star Wars
632	11.422751	John Wick
3	11.173104	Star Wars: The Force Awakens
633	10.739009	The Hunger Games: Mockingjay - Part 1

What are the top 10 movies in revenue?

In [67]:
```
df[['revenue', 'original_title']].sort_values(by='revenue', ascending=False).head(10)
```

Out[67]:

	revenue	original_title
1386	2781505847	Avatar
3	2068178225	Star Wars: The Force Awakens
5231	1845034188	Titanic
4361	1519557910	The Avengers
0	1513528810	Jurassic World
4	1506249360	Furious 7
14	1405035767	Avengers: Age of Ultron
3374	1327817822	Harry Potter and the Deathly Hallows: Part 2
5422	1274219009	Frozen
5425	1215439994	Iron Man 3

What are the 10 most profitable movies?

In [68]:
```
df[['profitability_ratio', 'original_title']].sort_values(by='profitabil
ity_ratio', ascending=False).head(10)
```

Out[68]:

	profitability_ratio	original_title
1559	99	Die PÃ¤pstin
7673	99	Gruz 200
7552	99	The Condemned
1479	99	The Uninvited
7604	99	The Babysitters
1477	99	Spread
7610	99	The Last Mimzy
7626	99	The Visitor
7629	99	King of California
7644	99	The Invincible Iron Man

This didn't work well since there are so many movies that have a profitability of 99%. Profitability is better compared along with other variables such as revenue and popularity.

Conclusions

Tip: Once you are satisfied with your work, you should save a copy of the report in HTML or PDF form via the **File** > **Download as** submenu. Before exporting your report, check over it to make sure that the flow of the report is complete. You should probably remove all of the "Tip" quotes like this one so that the presentation is as tidy as possible. Congratulations!

Summary of Data

- Drama, Comedy, Thriller, and Action are the most popular genres in general, and make up about 50% of all movies made from 1960-2015. TV Movies, Westerns, and Foreigns are the least popular.
- Since 1960, Drama has been the most popular genre per decade except for the 80's when Comedy was more poular.
- Runtime lengths have decreased by 16% from 1960 to 2015.
- Revenues from 1960-2015 have increased 391%, from 11 million to 54 million. However, taking into account the inflation adjustment it's held fairly steady.
- Although Drama, Comedy, and Thriller are the most popular genre overall, the Top 100 revenue producing movies are predominantly Adventure, Action, and Fantasy.
- Movies with higher revenues are consistently more profitable.
- Revenue and budget have a weakly positive correlation.
- Movies with higher revenues make more profits.
- Revenue and profitability have a weak positive correlation.
- Movies with higher revenues are more popular.

Notes & Limitations

This data was collected through The Movie Databae (TMDB). The value of 'popularity' and 'votes' is subjective and dependent on those users voting and navigating through the website. Pooling imdb and rotten tomatoes and other sources might yield more accurate results.

For top grossing movies, I picked the top 100 box office revenue. I could have broken it down differently, such as over a billion and under, or billions / millions / under, but after assessing all I found this to be a good breakdown.

Splitting up the values for the cast and directors columns really slowed down the processing so I went back and created 2 copies of the dataframe to perform those operations separately. I first split the genre column in the original df, then realized it affected assessment (such as most popular by title since the title was repeated over multiple rows) so I went back and created a new df with the genre split.

About profitability ratios: https://www.s-cool.co.uk/a-level/business-studies/ratio-analysis/revise-it/profitability-ratios (https://www.s-cool.co.uk/a-level/business-studies/ratio-analysis/revise-it/profitability-ratios)