Project: Investigate a Dataset - Analyze TMDb Movie Data

Table of Contents

Introduction

In [1]:
```
# Import the neccessary packages
import pandas as pd
import matplotlib as plt
% matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Data Wrangling

General Properties

DataSet chosen for analysis: TMDb Movie Data

The database contains information about movies collected from The Movies Database, including revenue, budget, ratings, and homepage. I displayed 10 rows to get a little more detailed result about the columns, values and structure. I decided to ask questions related financials, popularity and genres.

Questions posed:

1. How has the popularity of Western movies changed over the years?
2. Does budget correlate with popularity? What about the movies with the biggest budget?
3. Which 10 production company profited the most over the years?

In [2]:
```
# Load the data to check the structure and columns
df = pd.read_csv('tmdb-movies.csv')
df.head(5)
```

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...	
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://www
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	htt
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	

5 rows × 21 columns

The Data Structure

Before working with the data I checked the databese and looked for missing values, inconsistency or inadequate datatype. After getting more information and find out the questions I wanted to pose, I cleaned the database. There were unecessary columns with missing data, inadequate datatypes and rows with 0 value. The columns 'genres' and 'production_companies' contained multiple value that does no meet the requirements of first normal form.

The Cleaning Process

- I removed the columns cast, homepage, tagline, keywords, overview and imdb id to improve database performance.
- The column 'genres' and 'productions_companies' were not in the first normal form which requires that in the table should not have multiple value in the same row of data. I was unable to create a second joined column, so I decided to remove the values after the first '|' sign to get better groupping and cleaner visualization in the further analysis.
- I casted release_date from string to date datatype.
- I converted the columns revenue, budget, revenue_adj and budget_adj from float to int.
- The 0 values would distort the result of forther calculations so I replaced the 0 in revenue, budget, revenue_adj and budget_adj with means.
- I also replaced the Na values with 'Unknown' to improve interpretation.

In [3]:
```
# Get info about the database to check missing values and data types.

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
```

I converted the values to the proper data types.

In [4]:
```
# Convert release_date (object datatype) to date.

df['release_date'] = pd.to_datetime(df['release_date'])

# Convert budget_adj and revenue_adj from float to int.

df['budget_adj'] = df['budget_adj'].astype(int)
df['revenue_adj'] = df['revenue_adj'].astype(int)

# convert budget and revenue from float to int.

df['budget'] = df['budget'].astype(int)
df['revenue'] = df['revenue'].astype(int)
```

I removed some columns to improve readability and performance.

In [5]:
```
# Drop the unnecessary columns

df = df.drop(['cast', 'homepage', 'tagline', 'keywords', 'overview', 'imdb_id'
], axis=1)
```

Doublecheck my results.

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 15 columns):
id                      10866 non-null int64
popularity              10866 non-null float64
budget                  10866 non-null int64
revenue                 10866 non-null int64
original_title          10866 non-null object
director                10822 non-null object
runtime                 10866 non-null int64
genres                  10843 non-null object
production_companies    9836 non-null object
release_date            10866 non-null datetime64[ns]
vote_count              10866 non-null int64
vote_average            10866 non-null float64
release_year            10866 non-null int64
budget_adj              10866 non-null int64
revenue_adj             10866 non-null int64
dtypes: datetime64[ns](1), float64(2), int64(8), object(4)
memory usage: 1.2+ MB
```

In [7]:
```
# Replace the null values in the columns with 'unknown'

df['director'] = df['director'].fillna('Unknown')
df['production_companies'] = df['production_companies'].fillna('Unknown')
df['genres'] = df['genres'].fillna('Unknown')
```

In [8]:
```
# Check the results

df.query('director == "Unknown"').head(20)
df.query('production_companies == "Unknown"').head(20)
df.query('genres == "Unknown"').head(20)
```

Out[8]:

	id	popularity	budget	revenue	original_title	director	runtime	genres	pr
424	363869	0.244648	0	0	Belli di papÃ	Guido Chiesa	100	Unknown	
620	361043	0.129696	0	0	All Hallows' Eve 2	Antonio Padovan|Bryan Norton|Marc Roussel|Ryan...	90	Unknown	
997	287663	0.330431	0	0	Star Wars Rebels: Spark of Rebellion	Steward Lee|Steven G. Lee	44	Unknown	
1712	21634	0.302095	0	0	Prayers for Bobby	Russell Mulcahy	88	Unknown	
1897	40534	0.020701	0	0	Jonas Brothers: The Concert Experience	Bruce Hendricks	76	Unknown	
2370	127717	0.081892	0	0	Freshman Father	Michael Scott	0	Unknown	
2376	315620	0.068411	0	0	Doctor Who: A Christmas Carol	Unknown	62	Unknown	
2853	57892	0.130018	0	0	Vizontele	YÄ±lmaz ErdoÄŸan	110	Unknown	
3279	54330	0.145331	0	0	ì•„ê¸°ì™€ ë‚˜	Kim Jin-Yeong	96	Unknown	
4547	123024	0.520520	0	0	London 2012 Olympic Opening Ceremony: Isles of...	Danny Boyle	220	Unknown	
4732	139463	0.235911	0	0	The Scapegoat	Charles Sturridge	100	Unknown	
4797	369145	0.167501	0	0	Doctor Who: The Snowmen	Unknown	60	Unknown	
4890	126909	0.083202	0	0	Cousin Ben Troop Screening	Wes Anderson	2	Unknown	
5830	282848	0.248944	0	0	Doctor Who: The Time of the Doctor	James Payne	60	Unknown	
5934	200204	0.067433	0	0	Prada: Candy	Wes Anderson|Roman Coppola	3	Unknown	
6043	190940	0.039080	0	0	Bombay Talkies	Anurag Kashyap|Dibakar Banerjee|Zoya Akhtar|Ka...	127	Unknown	
6530	168891	0.092724	0	0	Saw Rebirth	Jeff Shuter|Daniel Viney	6	Unknown	
8234	56804	0.028874	0	0	Viaggi di nozze	Carlo Verdone	103	Unknown	

	id	popularity	budget	revenue	original_title	director	runtime	genres	pr
8614	65595	0.273934	0	0	T2 3-D: Battle Across Time	James Cameron	12	Unknown	
8878	92208	0.038045	0	0	Mom's Got a Date With a Vampire	Steve Boyum	85	Unknown	

I deleted some data from the rows with multiple values.

In [9]:
```
# df['genres'] = df['genres'].apply(lambda x: x.split('|')[0])
df['production_companies'] = df['production_companies'].apply(lambda x: x.spli
t('|')[0])
```

In [11]:
```
# Check changes
df['production_companies'].head(15)
```

Out[11]:
```
0                      Universal Studios
1               Village Roadshow Pictures
2                    Summit Entertainment
3                               Lucasfilm
4                     Universal Pictures
5                    Regency Enterprises
6                     Paramount Pictures
7      Twentieth Century Fox Film Corporation
8                     Universal Pictures
9                    Walt Disney Pictures
10                     Columbia Pictures
11              Village Roadshow Pictures
12                              DNA Films
13                     Columbia Pictures
14                        Marvel Studios
Name: production_companies, dtype: object
```

I replace 0 values with means in columns budget_adj and revenue_ad, budget and revenue.

In [12]:
```
df['budget']=df['budget'].replace(0,df['budget'].mean())

df['revenue']=df['revenue'].replace(0,df['revenue'].mean())

df['budget_adj']=df['budget_adj'].replace(0,df['budget_adj'].mean())

df['revenue_adj']=df['revenue_adj'].replace(0,df['revenue_adj'].mean())
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 15 columns):
id                      10866 non-null int64
popularity              10866 non-null float64
budget                  10866 non-null float64
revenue                 10866 non-null float64
original_title          10866 non-null object
director                10866 non-null object
runtime                 10866 non-null int64
genres                  10866 non-null object
production_companies    10866 non-null object
release_date            10866 non-null datetime64[ns]
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: datetime64[ns](1), float64(6), int64(4), object(4)
memory usage: 1.2+ MB
```

Exploratory Data Analysis

I recently read an article about the evolution of Western movies. It mentioned that the most prolific era was in the 1930s to the 1960s and the genre almost vanished in the 1980s. Tthere were little sign of resurgence after the 1990s but Western has not got back its popularity yet. As I am a big fan of the genre, I decided to analyze its populatiry over the decades and test the assumptions on the data. However, I use the article only as an interesting starting point - I do not intend to draw far-reaching conclusions.

1. Have changed the popularity of Western movies over the decades?

At first, I created a smaller dataframe which contained movies where in the column 'genre' appeared the word 'Western'. I decided to get every Western influenced or Western styled movie, I did not want to define the conditions too strict to get a bigger dataframe. I also planned to analyze the popularity by decades, not by release_year.

In [14]:
```
# Create dataframe to every movie with the genre 'Western'

df_western = df[df['genres'].str.contains("Western")]
```

To check the results I simply counted the records in the dataframe, the records with genre 'Western', and compared the numbers. I got True, so the two values are the same.

In [16]:
```
# Check the results by comparing the calculations.

df_western.genres.str.contains(r'Western').sum() == df_western['id'].count()
```

Out[16]:
```
True
```

Mapping the release years to decades.

In [17]:
```
# Create bin edges to decades
decades = [1960, 1970, 1980, 1990, 2000, 2010, 2020]

# Create labels
decade_names = ['1960', '1970', '1980', '1990', '2000', '2010']

# Create new column and cut into bins
df_western['release_decade'] = pd.cut(df['release_year'], decades, labels=deca
de_names)

df_western.head()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

Out[17]:

	id	popularity	budget	revenue	original_title	director	runtime	
5	281957	9.110700	1.350000e+08	5.329505e+08	The Revenant	Alejandro GonzÃ¡lez IÃ±Ã¡rritu	156	Western|[
15	273248	5.898400	4.400000e+07	1.557601e+08	The Hateful Eight	Quentin Tarantino	167	Crime|
125	223485	1.329702	1.462570e+07	2.290940e+05	Slow West	John Maclean	84	R
145	294963	1.073349	1.800000e+06	3.982332e+07	Bone Tomahawk	S. Craig Zahler	132	Horror|W
165	347969	0.913085	6.000000e+07	3.982332e+07	The Ridiculous 6	Frank Coraci	119	

In [18]:
```
# Check if western movies are available in every decade and count their number

df_western.groupby(['release_decade'], as_index = False)['id'].count()
```

Out[18]:

	release_decade	id
0	1960	38
1	1970	33
2	1980	12
3	1990	24
4	2000	30
5	2010	22

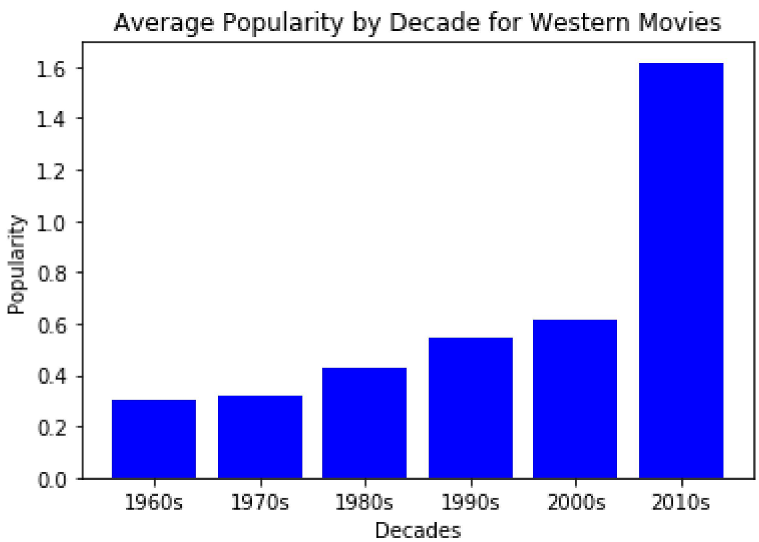
In [19]:
```
# Get the average popularity by decades

df_western.groupby(['release_decade'], as_index = False)['popularity'].mean()
```

Out[19]:

	release_decade	popularity
0	1960	0.301672
1	1970	0.315369
2	1980	0.429861
3	1990	0.544739
4	2000	0.612382
5	2010	1.616852

In [20]:
```
# Get the mean of vote_average in every decade

df_1960 = df_western.query('release_decade == "1960"')
df_1960_mean = df_1960['popularity'].mean()

df_1970 = df_western.query('release_decade == "1970"')
df_1970_mean = df_1970['popularity'].mean()

df_1980 = df_western.query('release_decade == "1980"')
df_1980_mean = df_1980['popularity'].mean()

df_1990 = df_western.query('release_decade == "1990"')
df_1990_mean = df_1990['popularity'].mean()

df_2000 = df_western.query('release_decade == "2000"')
df_2000_mean = df_2000['popularity'].mean()

df_2010 = df_western.query('release_decade == "2010"')
df_2010_mean = df_2010['popularity'].mean()
```

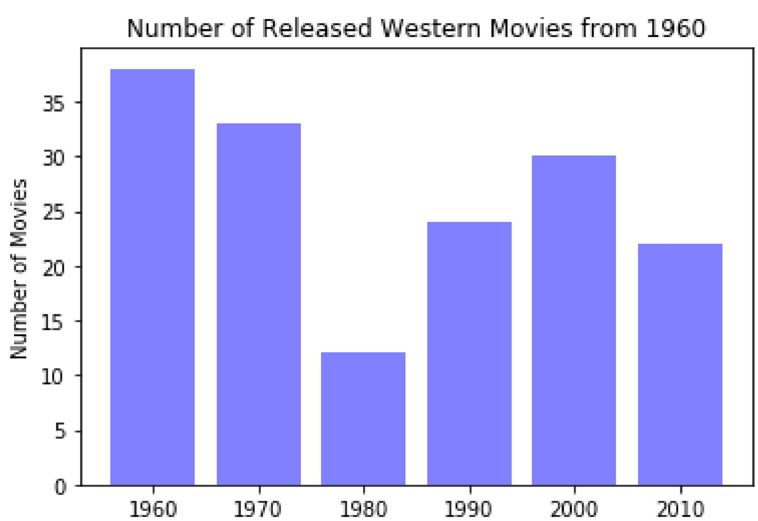
In [21]:
```
# labels_decades = ("60's", "70's", "80's", "90's", "2000's", "2010's")

# df_western_avg_votes.plot(tick_label = labels_decades, kind = 'bar', title
 ='Average Votes by Decade')

# createa bar chart with labels

locations = [1, 2, 3, 4, 5, 6]
heights = [df_1960_mean, df_1970_mean, df_1980_mean, df_1990_mean, df_2000_mea
n, df_2010_mean]
labels = ['1960s', '1970s', '1980s', '1990s', '2000s', '2010s']
plt.bar(locations, heights, tick_label=labels, color = 'b', )
plt.title('Average Popularity by Decade for Western Movies')
plt.xlabel('Decades')
plt.ylabel('Popularity');
```



As we can see, the popularity has steadly grew over the years which does not meet our expectations. But during the calculations I found something else that can confirm the assumptions, so I created another chart to visualize the popularity of the genre. In this case, I simply display the number of released Western movies.

In [22]:
```
df_west_sum = df_western.groupby(['release_decade'], as_index = False)['id'].c
ount()

years = ('1960', '1970', '1980', '1990', '2000', '2010')
pos = np.arange(0, 6, 1)
plt.xticks(pos, years)
bar_heights = df_west_sum['id']

print('Sum of Released Movies: ')
print(df_west_sum)

dc_bars = plt.bar(x = years, color='b', alpha=.5, height = bar_heights)

plt.ylabel('Number of Movies')
plt.title('Number of Released Western Movies from 1960')

plt.legend()
plt.show()
```

```
Sum of Released Movies: 
  release_decade  id
0           1960  38
1           1970  33
2           1980  12
3           1990  24
4           2000  30
5           2010  22
```



According to the chart, the number of Western movies slide started to slide steeply after the '70s and reached the bottom in the '80s. If we measured the evolution and popularity of Western by the number of released movies, the results would meet our assumptions but I would not like to imply causation based on my results.

2. Ratings for the Cheapest and Most Expensive Movies

I was curious about if there are any measurable difference between votes of the most and the least expensive movies. I decided to get information about the most and least expensive films and viualize the distribution of their votes in one histogram.

Part 1: Get the most expensive movies

At first, I sorted the movies by budget to get the 200 most expensive movies from the database.

In [25]:
```
# Sort movies by budget in descending order

sorted_budget_biggest = df.sort_values(by=['budget_adj'], ascending = False).head(200)
```

I got the most expensive movies and their ratings.

In [29]:
```
# Get the most expensive movies with ratings

sorted_budget_biggest.groupby('original_title')['vote_average'].mean()
```

Out[29]:
```
original_title
2012                                   5.6
47 Ronin                               5.8
A Bug's Life                           6.6
A Christmas Carol                      6.6
Alexander                              5.6
Alice in Wonderland                    6.3
Angels & Demons                        6.3
Armageddon                             6.4
Atlantis: The Lost Empire              6.5
Avatar                                 7.1
Avengers: Age of Ultron                7.4
Bad Boys II                            6.3
Batman & Robin                         4.4
Batman Begins                          7.3
Batman Forever                         5.2
Battleship                             5.5
Bee Movie                              5.6
Big Hero 6                             7.8
Bolt                                   6.3
Brave                                  6.6
Captain America: The Winter Soldier    7.6
Cars 2                                 5.8
Casino Royale                          7.1
Charlie and the Chocolate Factory      6.5
Charlie's Angels: Full Throttle        5.3
Chicken Little                         5.6
Cleopatra                              6.3
Cowboys & Aliens                       5.4
Cutthroat Island                       6.1
Dante's Peak                           5.5
                                      ...
Thor                                   6.5
Thor: The Dark World                   6.8
Titanic                                7.3
Tomorrow Never Dies                    5.9
Tomorrowland                           6.2
Tora! Tora! Tora!                      6.6
Toy Story 3                            7.5
Transformers                           6.6
Transformers: Age of Extinction        5.9
Transformers: Dark of the Moon         6.1
Transformers: Revenge of the Fallen    6.0
Treasure Planet                        7.0
Troy                                   6.8
True Lies                              6.6
Up                                     7.6
Van Helsing                            5.9
WALLÂ·E                                7.6
War of the Worlds                      5.9
Waterloo                               6.2
Waterworld                             5.8
White House Down                       6.4
Wild Wild West                         5.2
Windtalkers                            5.9
World War Z                            6.7
Wrath of the Titans                    5.5
```

```
Wreck-It Ralph                         7.0
X-Men Origins: Wolverine               6.2
X-Men: Days of Future Past             7.6
X-Men: First Class                     7.0
X-Men: The Last Stand                  6.2
Name: vote_average, Length: 199, dtype: float64
```

In [44]:

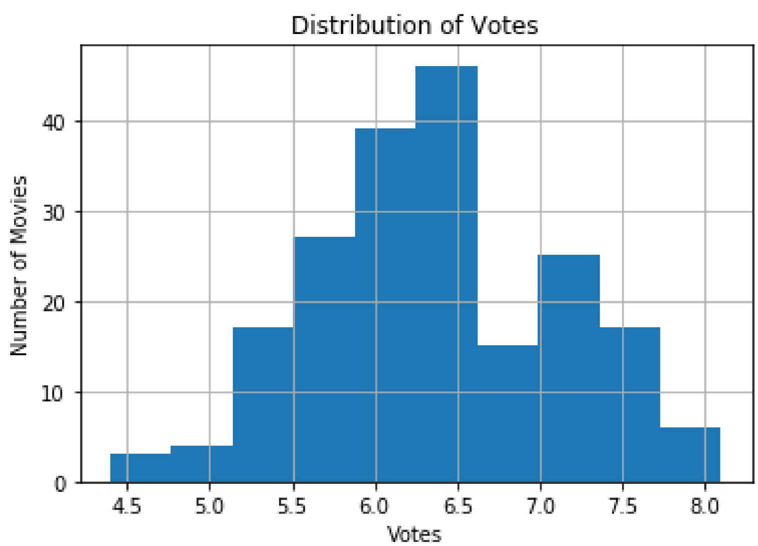
```
# Save the results to a variable.

exp_budget_vote = sorted_budget_biggest.groupby(['original_title'])['vote_aver
age'].mean()
```

I created a hitogram to get a picture about the distribution.

In [33]:

```
# Create a plot to visualize the results

plt.xlabel('Votes')
plt.ylabel('Number of Movies')
plt.title('Distribution of Votes')

exp_budget_vote.hist(histtype = 'stepfilled', label = 'Rates of the Most Expen
sive Movies')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd4aeb655f8>



Part 2: get the cheapest movies

I queried the 200 cheapest movies from the database.

In [38]:

```
# Get the cheapest movies with ratings

sorted_budget_cheapest = df.sort_values(by=['budget_adj'], ascending = True).h
ead(200)
```

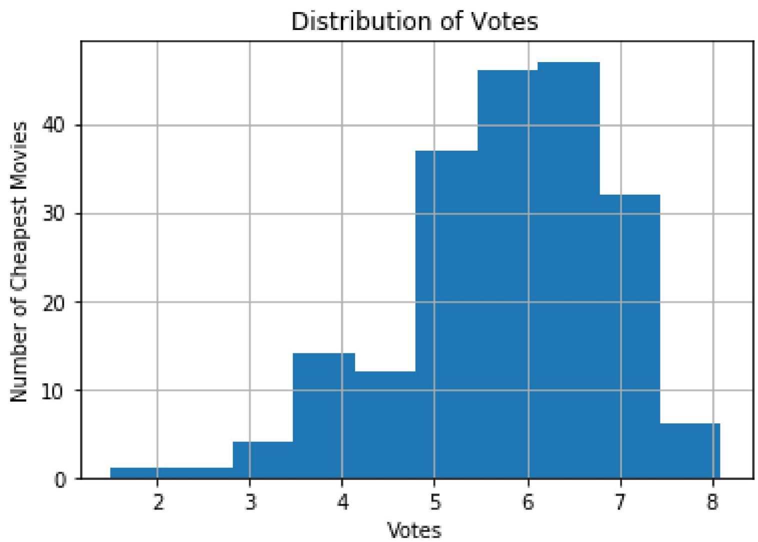
I queried the cheapest movies and their rating.

In [46]:
```
cheap_budget_vote = sorted_budget_cheapest.groupby('original_title')['vote_ave
rage'].mean()
```

I created a histogram to visualize the distribuion.

In [48]:
```
plt.xlabel('Votes')
plt.ylabel('Number of Cheapest Movies')
plt.title('Distribution of Votes')

cheap_budget_vote.hist(label = 'Rates of the Cheapest Movies')
```

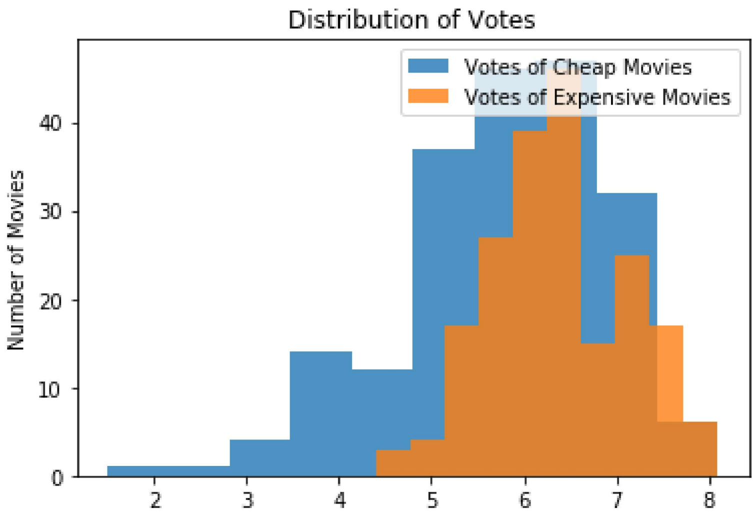
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd4ae74a0b8>



Part 3: Compare the results in one diagram

I created a diagram to display the differences between the ratings.

In [50]:
```
plt.hist(cheap_budget_vote, alpha=0.8, label='Votes of Cheap Movies')
plt.hist(exp_budget_vote, alpha=0.8, label='Votes of Expensive Movies')
plt.ylabel('Number of Movies')
plt.title('Distribution of Votes')
plt.legend(loc='upper right')
plt.show()
```



Part 4: Conclusion

As a conclusion, I can say that the most expensive movies generally got better rating that the cheaper ones. We can see on the diagram that the worst rating is 4.5 while the cheapest movies worst rating were lower than 2.

Conclusions

In the first section I examined the popularity of Western movies over the decades. I made my analyzation based on the values of 'released_year' and 'popularity'. I could not find any correlations between the numbers and the assumptions but I found it by taking into account the numbers of released movies.

After that I analyzed the ratings of the most and least expensive movies and I found out that the more expensive movies got higher votes than the cheaper ones.

Limitations

In the first section - although the literature details the phenomenon - I could not find any correlation between 'popularty' and 'release year'. It would be good to know more about what is behind the value 'popularity' and what popularity means here. Just to name a few... How was it calculated? Which criterias and values were measured exactly to get these numbers? It could be caculated based on ticket sales? Or based on audience appraisal? However, I found correlation between my assumptions and the number of released western movies but I would not name it causation without a much more detailed further analysis.

In the second section, I made my calculations based on the values of budget adjustment to take the fluctuations into account, I found this really useful. But there were more missing values in the 'budget_adj' column. During the cleaning process I replaced the missing values with the average, but it still can distort the result (for instance, there would be other movies among the most expensive 200 movies).

In [2]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Investigate_a_Dataset-Copy1.ipynb'])
```

Out[2]: 0

In [ ]:
```

```