Movie Data Analysis

John Basil

This study directly references code from the course materials authored by Dr. Marko Samara for the Data Science curriculum at Arizona State University.

Introduction

A goal of business is to deliver products and services that are desirable to (or, demanded by) potential customers. This can be true of business in any industry, whether semiconductors or movies. As a result, in order to determine which products and services to deliver, a business may conduct research into the reasons that some products and services are more desirable than others to potential customers. One way to conduct such research is to analyze historical data of product sales, looking for trends that could help predict future sales.

This study seeks to demonstrate such an analysis by determining which features (or inputs) of past product/service offerings have generated the most sales (or output), using sales as an indicator of customer demand and predictor of future sales. Because demand and sales do not necessarily translate to margin, which to some extent is necessary in order for the business to continue operations, the analysis herein considers the efficiency (sales minus costs) of delivering certain features.

The dataset to be chosen for this study needs to be capable of satisfying the aforesaid requirements: namely, to provide both sales and cost data for each product/service offering, and to characterize each offering by numerous qualities that could be controlled by the business during production and prior to delivery to the customer (i.e. features). With over 1,000,000 offerings in the form of movies, and parameters encompassing sales, cost and quality, the TMDB movie dataset is initially selected.

Preparation

The dataset begins with 24 parameters, some of which are superfluous: id, title, vote_average, vote_count, status, release_date, sales, runtime, adult, backdrop_path, budget, homepage, imdb_id, original_language, original_title, overview, popularity, poster_path, tagline,

genres, production_companies, production_countries, spoken_languages, keywords. This survey of parameters now needs to be evaluated in order to identify which parameters are useful, and whether these residual parameters suffice for achieving the goals of this study.

The id is a unique numerical key identifier helpful for distinguishing two otherwise similar entries. In contrast, if title were used to distinguish movies, some mutually distinct movies share titles, causing some entries to be overwritten, and necessitating a unique identifier represented by id.

The title provides some basic information about the movie. When processing the data, an ad-hoc analysis using common knowledge of movie titles can help with determining whether results seem realistic, or otherwise if the data may have been processed incorrectly.

The vote_average is an average of ratings (on a scale of 1 to 10), each (presumably) assessed by an individual movie viewer. How representative these ratings are of the actual sentiments of the true population of movie viewers is uncertain; however, TMDB is generally considered a reliable resource and the ratings generally seem to match expected public sentiment (based on personal knowledge of public sentiment towward some movies). This rating may indicate how 'good' a movie is, which a production company could influence to some extent by applying additional quality control measures in order to ensure that the movie meets or exceeds the expectations of viewers (notwithstanding the cost of such additional measures).

The vote_count describes how many ratings were involved in the computation of the vote_average. This parameter could be used to remove from the dataset some movies which have too few ratings for the average_rating data to be useful.

The status describes whether a movie has been released or not. A movie that has not yet been released will be missing most data which is not useful for this analysis, so unreleased movies would need be removed from the dataset, after which this parameter could also be removed.

The release_date is useful for determining whether, and if so, how, trends change over time. Each date is formatted as a string, so in order to facilitate comparison between dates, conversion to a more intuitive format would seem sensirble.

The sales and budget parameters map to the sales and cost data which are focal points of this analysis. Movies with limited or out of range sales and/or cost data could be filtered out from the dataset.

The runtime is a quality that could be controlled in order to better suit the preferences of viewers if a trend in runtime and satisfaction (i.e. vote_average) can be identified (and especially where vote_average may translate to higher demand).

The popularity is an amalgam of other parameters such as vote count and average, and release date, as well as excluded data such as favorite and view counts, recompiled on a daily basis. Data collection for this parameter only began on 2017-04-28 (source: https://developer.themoviedb.org/docs/popularity-and-trending), limiting the number of observations upon which to base inferences, which lowers confidence in any observed trends. However, inclusion of this parameter might be useful for further substantiating trends observed from other parameters over the pertinent time period.

The genres parameter lists genre categories that are suitable for describing the style of the movie. Each list is formatted as a string, so in order to associate a single genre with a movie, each string list would need to be split into individual strings describing each category, where each of these categories could be appended to the dataset as an individual column confirming whether or not the category applies to each movie. The style of the movie can be controlled during the movie planning process, so inferring whether, and if so, to what extent, certain genres are more desirable (as well as cost-effective) than others could help inform the production company where to focus its resources. Categories with limited applicability (possibly being outdated or niche) can be identified and filtered out of the dataset.

Additional parameters, such as production_companies and original_languages, can be investigated in order to infer whether, and if so, how movies produced by production companies and/or for certain countries are more desirable. Due to limited time, the resulting need to limit scope, and the limited predicted usefulness of such parameters compared with other selected parameters, these parameters, in addition to those parameters that are irrelevant to this study, can be excluded from the dataset.

If the results of this study were more consequential, potentially impacting margins and losses, additional measures (i.e. research and validation into the collection methodology for each parameter of the dataset) would be necessary in order to ascertain the extent of data integrity. However, for the purpose of an academic demonstration, the selected parameters seem sufficient in order to analyze whether, and if so, to what extent certain features influence sales and margins.

This study incorporates source code that is written semantically and procedurally so as to serve as additional dialogue into the thought process behind the analysis. In addition to base Python, a selection of libraries are imported into the project file in order to facilitate data processing and visualization. The dataset file is then read into a new dataframe object. An instance of this dataframe object is assigned to a new variable in order to preserve the original dataframe where the instance assigned to the variable is subjected to modifications and type conversions.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import warnings
```

In [3]:
```
warnings.filterwarnings('ignore')
df = pd.read_csv('~/Desktop/datasets/TMDB_movie_dataset_v11.csv')
movies = df[:]
```

An initial count confirms that the dataset contains over 1,000,000 movies. This analysis is only concerned with a selection of parameters for released movies meeting certain thresholds, so the dataset is subset on the basis of columns (only a selection of parameters) and rows (only released movies meeting certain thresholds). The status column is not useful for the analysis after released movies are obtained, so this subset of released movies is assigned to the dataset variable prior to column selection. After a subset of selected columns is assigned to the dataset variable, a subset of rows selected by meeting certain thresholds (in the budget, sales and votes columns) is assigned to the dataset variable. The columns are also renamed for clarity and simplicity. The margin column, which represents the difference between sales and budget, is limited in usefulness by the lack of actual cost data; in order to further substantiate results, a subsequent analysis could assess the impact of variance between budget and actual cost by comparing the subset of results from this dataset that correspond to results from a different dataset containing actual cost data.

In [4]:
```
movies['id'].size
```

Out[4]:
```
1131296
```

In [5]:
```
movies = movies[movies.status == 'Released']
movies['id'].size
```

Out[5]:
```
1105740
```

In [6]:
```
movies = movies[['id','title','vote_average','vote_count','release_date','revenue','runtime','budget','popularity',
```

In [7]:
```
movies.head()
```

Out[7]:

	id	title	vote_average	vote_count	release_date	revenue	runtime	budget	popularity	genres
0	27205	Inception	8.364	34495	2010-07-15	825532764	148	160000000	83.952	Action, Science Fiction, Adventure
1	157336	Interstellar	8.417	32571	2014-11-05	701729206	169	165000000	140.241	Adventure, Drama, Science Fiction
2	155	The Dark Knight	8.512	30619	2008-07-16	1004558444	152	185000000	130.643	Drama, Action, Crime, Thriller
3	19995	Avatar	7.573	29815	2009-12-15	2923706026	162	237000000	79.932	Action, Adventure, Fantasy, Science Fiction
4	24428	The Avengers	7.710	29166	2012-04-25	1518815515	143	220000000	98.082	Science Fiction, Action, Adventure

In [8]:
```
movies = movies.rename(columns={'vote_average' : 'rating', 'vote_count' : 'votes',
                                'release_date' : 'date', 'revenue' : 'sales'})
margin = movies.sales - movies.budget
profitable = margin > 0
```

In [9]:
```
movies = pd.DataFrame(movies, columns=['id','title','rating','votes','date','sales',
                                       'runtime','budget','popularity','genres','margin','profitable'])
movies.margin = margin
profitable = margin > 0
movies.profitable = profitable
movies.columns
```

Out[9]:
```
Index(['id', 'title', 'rating', 'votes', 'date', 'sales', 'runtime', 'budget',
       'popularity', 'genres', 'margin', 'profitable'],
      dtype='object')
```

In [10]:
```
movies = movies[abs(movies.sales) > 9999][movies.budget > 9999][movies.votes > 99]
movies['id'].size
```

Out[10]:
```
7022
```

In [11]:
```
# credit: https://www.reddit.com/r/learnpython/comments/k6opo8/pandas_convert_strings_to_boolean_columns/?rdt=49966
```

```
genres = (genre for genres in movies.genres for genre in genres.split(', '))
for genre in set(genres):
    movies[genre] = movies.genres.str.contains(genre)
```

In [12]:
```
movies.columns
```

Out[12]:
```
Index(['id', 'title', 'rating', 'votes', 'date', 'sales', 'runtime', 'budget',
       'popularity', 'genres', 'margin', 'profitable', 'Drama', 'History',
       'Crime', 'Horror', 'Adventure', 'Western', 'Mystery', 'Fantasy',
       'Documentary', 'TV Movie', 'Comedy', 'Thriller', 'Music', 'War',
       'Romance', 'Science Fiction', 'Family', 'Action', 'Animation'],
      dtype='object')
```

In [13]:
```
movies.head()
```

Out[13]:

	id	title	rating	votes	date	sales	runtime	budget	popularity	genres	...	TV Movie	Comedy	Thriller	Mus
0	27205	Inception	8.364	34495	2010-07-15	825532764	148	160000000	83.952	Action, Science Fiction, Adventure	...	False	False	False	Fal
1	157336	Interstellar	8.417	32571	2014-11-05	701729206	169	165000000	140.241	Adventure, Drama, Science Fiction	...	False	False	False	Fal
2	155	The Dark Knight	8.512	30619	2008-07-16	1004558444	152	185000000	130.643	Drama, Action, Crime, Thriller	...	False	False	True	Fal
3	19995	Avatar	7.573	29815	2009-12-15	2923706026	162	237000000	79.932	Action, Adventure, Fantasy, Science Fiction	...	False	False	False	Fal
4	24428	The Avengers	7.710	29166	2012-04-25	1518815515	143	220000000	98.082	Science Fiction, Action, Adventure	...	False	False	False	Fal

5 rows × 31 columns

In [14]:
```
movies['profitable']
```

Out[14]:
```
0         True
1         True
2         True
3         True
4         True
         ...
18119     True
18127    False
18129     True
18132    False
18156     True
Name: profitable, Length: 7022, dtype: bool
```

In [15]:
```
movies = movies.rename(columns={'Music' : 'Musical', 'Science Fiction' : 'SciFi', 'TV Movie': 'TV'})
genre_counts = [movies[movies.Action == True][['id']].size, movies[movies.Adventure == True][['id']].size,
                movies[movies.Animation == True][['id']].size, movies[movies.Comedy == True][['id']].size,
                movies[movies.Crime == True][['id']].size, movies[movies.Drama == True][['id']].size,
                movies[movies.Family == True][['id']].size, movies[movies.Fantasy == True][['id']].size,
                movies[movies.History == True][['id']].size, movies[movies.Horror == True][['id']].size,
                movies[movies.Musical == True][['id']].size, movies[movies.Mystery == True][['id']].size,
                movies[movies.Romance == True][['id']].size, movies[movies.SciFi == True][['id']].size,
                movies[movies.Thriller == True][['id']].size, movies[movies.TV == True][['id']].size,
                movies[movies.War == True][['id']].size, movies[movies.Western == True][['id']].size]
genre_counts
```

Out[15]:
```
[1811,
 1313,
 425,
 2507,
 1143,
 3156,
 754,
 760,
 380,
 895,
 232,
 655,
 1226,
 835,
 1881,
 3,
 274,
 116]
```

In [16]:
```
movies = movies.drop(['genres', 'Western', 'TV'], axis='columns')
movies.columns
```

Out[16]:
```
Index(['id', 'title', 'rating', 'votes', 'date', 'sales', 'runtime', 'budget',
       'popularity', 'margin', 'profitable', 'Drama', 'History', 'Crime',
       'Horror', 'Adventure', 'Mystery', 'Fantasy', 'Documentary', 'Comedy',
       'Thriller', 'Musical', 'War', 'Romance', 'SciFi', 'Family', 'Action',
       'Animation'],
      dtype='object')
```

In [17]:
```
genre_counts.pop()
genre_counts.pop(15)
genre_counts
```

Out[17]:
```
[1811,
 1313,
 425,
 2507,
 1143,
 3156,
 754,
 760,
 380,
 895,
 232,
 655,
 1226,
 835,
 1881,
 274]
```

In [18]:
```
genres = ['Action', 'Adventure', 'Animation', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy',
          'History', 'Horror', 'Musical', 'Mystery', 'Romance', 'SciFi', 'Thriller', 'War']
movies = movies.reindex(columns=(['id','title','date','runtime','votes','rating','popularity','budget','sales','mar
movies.columns
```

Out[18]:
```
Index(['id', 'title', 'date', 'runtime', 'votes', 'rating', 'popularity',
       'budget', 'sales', 'margin', 'profitable', 'Action', 'Adventure',
       'Animation', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'History',
       'Horror', 'Musical', 'Mystery', 'Romance', 'SciFi', 'Thriller', 'War'],
      dtype='object')
```

In [19]:
```
date = movies.date
budget = movies.budget
votes = movies.votes
rating = movies.rating
runtime = movies.runtime
popularity = movies.popularity
sales = movies.sales
margin = movies.margin
profitable = movies.profitable
```

Exploration

To start, the data is cleaned and minified by dropping irrelevant columns and by filtering out rows dated earlier than a relevance baseline. Then, the data is plotted in order to develop familiarity with the dataset by identifying possible patterns, checking for reasonableness, and refining assumptions before beginning the analyses.

In [20]:
```
print(movies.shape)
```

```
(7022, 27)
```

In [21]:
```
movies.head()
```

Out[21]:

	id	title	date	runtime	votes	rating	popularity	budget	sales	margin	...	Family	Fantasy	History	Ho
0	27205	Inception	2010-07-15	148	34495	8.364	83.952	160000000	825532764	665532764	...	False	False	False	F
1	157336	Interstellar	2014-11-05	169	32571	8.417	140.241	165000000	701729206	536729206	...	False	False	False	F
2	155	The Dark Knight	2008-07-16	152	30619	8.512	130.643	185000000	1004558444	819558444	...	False	False	False	F
3	19995	Avatar	2009-12-15	162	29815	7.573	79.932	237000000	2923706026	2686706026	...	False	True	False	F
4	24428	The Avengers	2012-04-25	143	29166	7.710	98.082	220000000	1518815515	1298815515	...	False	False	False	F

5 rows × 27 columns

In [22]:
```
sorted_ratings = movies.sort_values(by='rating')[['rating']]
sorted_ratings.head()
```

Out[22]:

	rating
16333	1.908
11021	2.719
2259	2.890
15744	2.909
8057	3.152

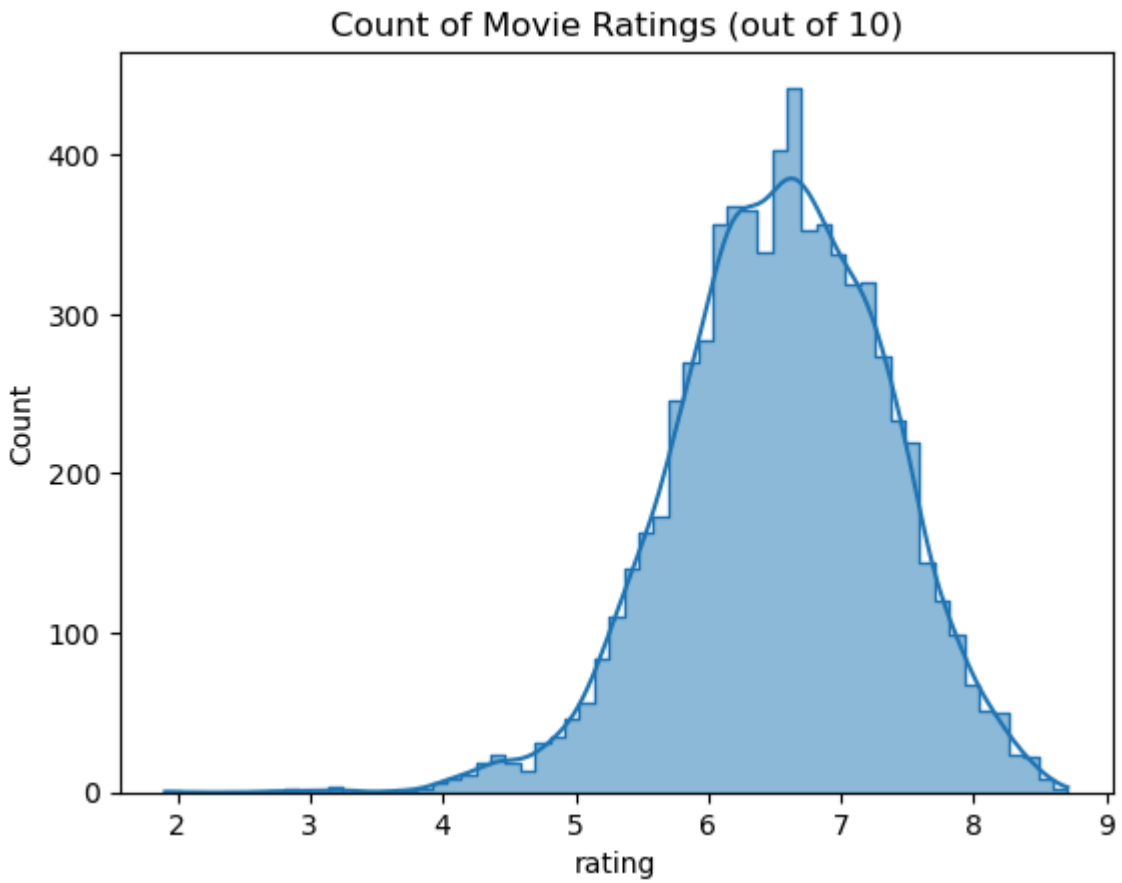
In [23]:
```
sorted_ratings.tail()
```

Out[23]:

	rating
991	8.552
119	8.573
215	8.591
14	8.702
53	8.707

In [24]:
```
sns.histplot(x=rating, kde=True, element='step')
plt.title("Count of Movie Ratings (out of 10)")
```

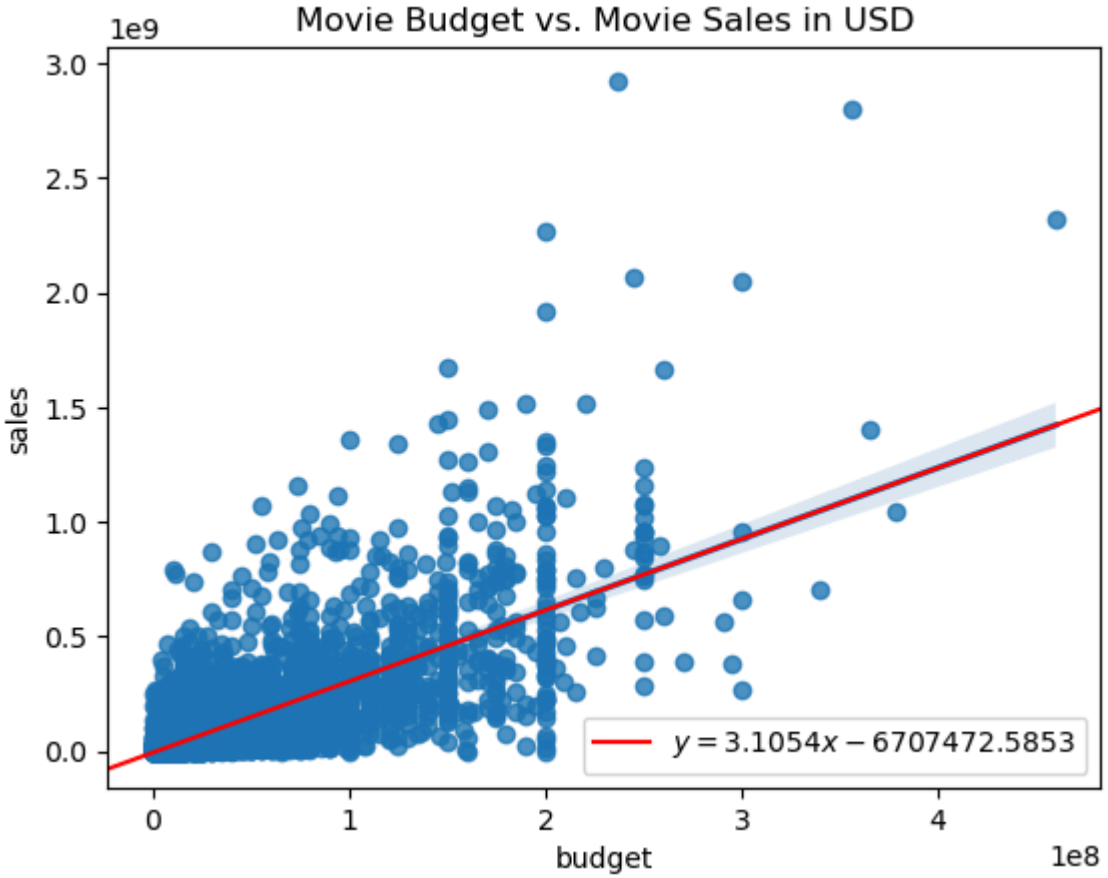
Out[24]:
```
Text(0.5, 1.0, 'Count of Movie Ratings (out of 10)')
```



Rather than centering between the lowest and highest ratings (1.908 and 8.707, respectively, centered at 5.308), this histogram of ratings centers at around 6.600. This result represents a possible 24.4% increase in ratings beyond expected values under the assumption of the rating results being balanced. However, the probability distribution is slightly uneven, with a sharper slope on the high end of the distibution.

In [25]:
```
sns.regplot(x=budget, y=sales)
m, b = np.polyfit(x=budget, y=sales, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Budget vs. Movie Sales in USD")
plt.legend(loc='lower right', bbox_to_anchor=(0,0,1,1))
```

Out[25]:
```
<matplotlib.legend.Legend at 0x70b985be5340>
```



This plot treating sales as a response to budget shows a strong positive correlation. The equation of the linear regression line suggests that sales triples for each unit budget increase. If the relationship between sales and budget increase was one-to-one or less, margin would decrease as budget increased, but these results suggest that margin also increases as budget increases. Nonetheless, budget increase without appreciation for what particular aspects of budget increase drives sales (and margin) is meaningless in a business context.

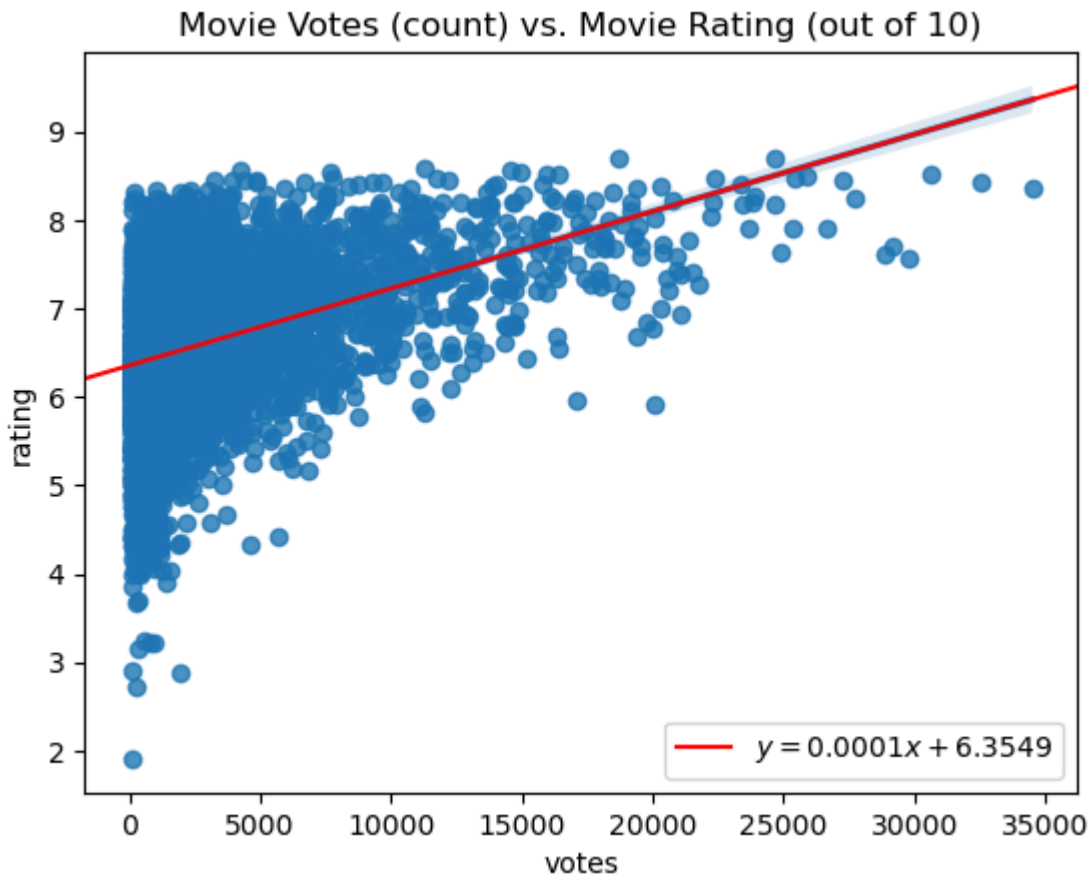
Among parameters chosen from the dataset for this study, budget is relatively straightforward to control. In addition, budget can drive other parameters, parameters which themselves can drive sales. Increases to these other parameters may be more proximate to and as such more descriptive of sales increase. As such, budget could be viewed as a driver of other drivers, with other drivers serving as intermediaries to the sales response.

In addition to some of the parameters associated with this study, typically, an increase in movie budget is associated with better casting, more production time, and increased advertising, so while these factors are not described by this dataset, one could infer based on typical movie production practices how budget is allocated so as to extrapolate other parameters besides those described by this dataset.

Even if the aforesaid extraneous parameters are more consequential to sales than the parameters described by this dataset, one could still use the results of this study to rule-out parameters described by this dataset that do not positively correlate with sales. In order to be more conclusive about which parameters to drive in order to increase sales, a fuller view of areas associated with budget increase, including parameters described in different datasets, should also be considered as part of a cost-benefit analysis.

In [26]:
```
sns.regplot(x=votes, y=rating)
m, b = np.polyfit(x=votes, y=rating, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Votes (count) vs. Movie Rating (out of 10)")
plt.legend(loc='lower right')
```

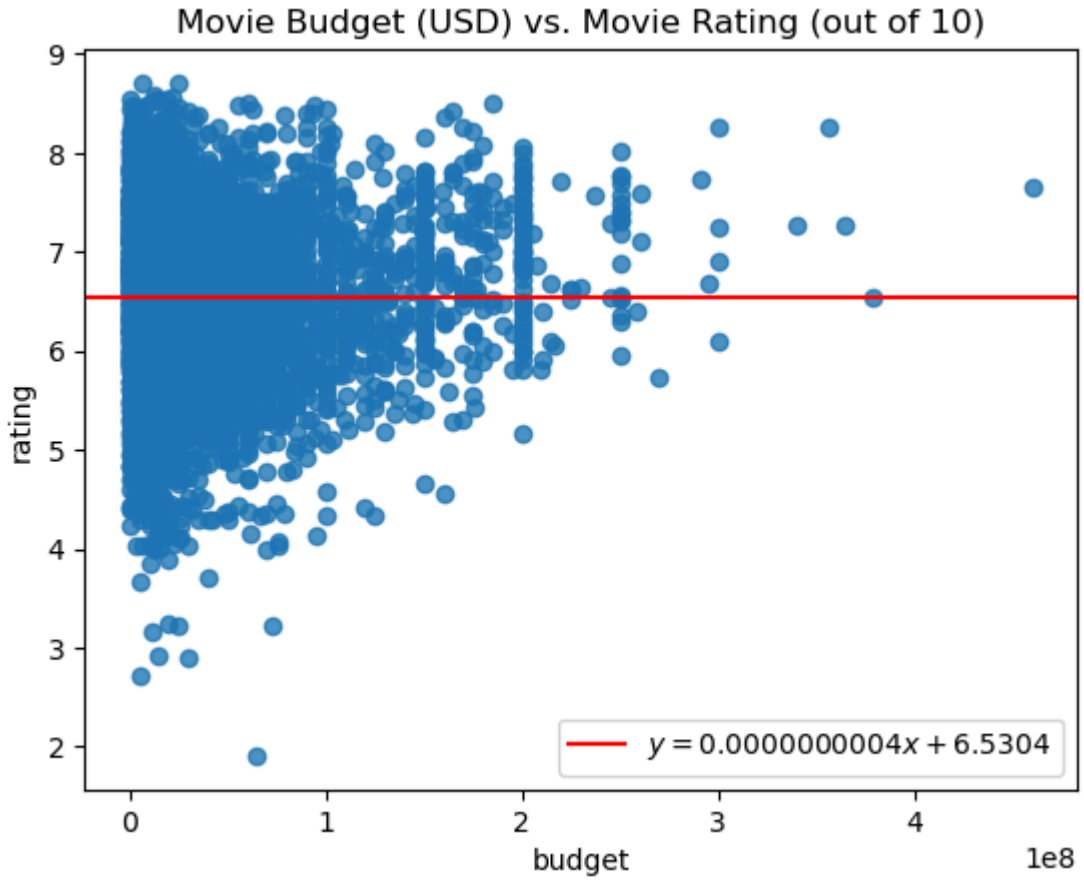
Out[26]:
```
<matplotlib.legend.Legend at 0x70b985aba000>
```



This plot treating rating as a response to votes shows a positive correlation. According to the linear regression line, a one-point increase in rating corresponds to a 10,000 count increase in votes (from a baseline 6.35 rating for 0 votes). This result suggests that voters tend to vote more for movies that they rate higher. This relationship reflects the observation from the earlier histogram of ratings that showed more observations (i.e. votes) at the higher end of the histogram (i.e. the end corresponding to higher ratings).

In [27]:
```
sns.regplot(x=budget, y=rating, fit_reg=False)
m, b = np.polyfit(x=budget, y=rating, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.10f}x {b:+.4f}$', color='red')
plt.title("Movie Budget (USD) vs. Movie Rating (out of 10)")
plt.legend(loc='lower right')
```

Out[27]: <matplotlib.legend.Legend at 0x70b985be4a10>

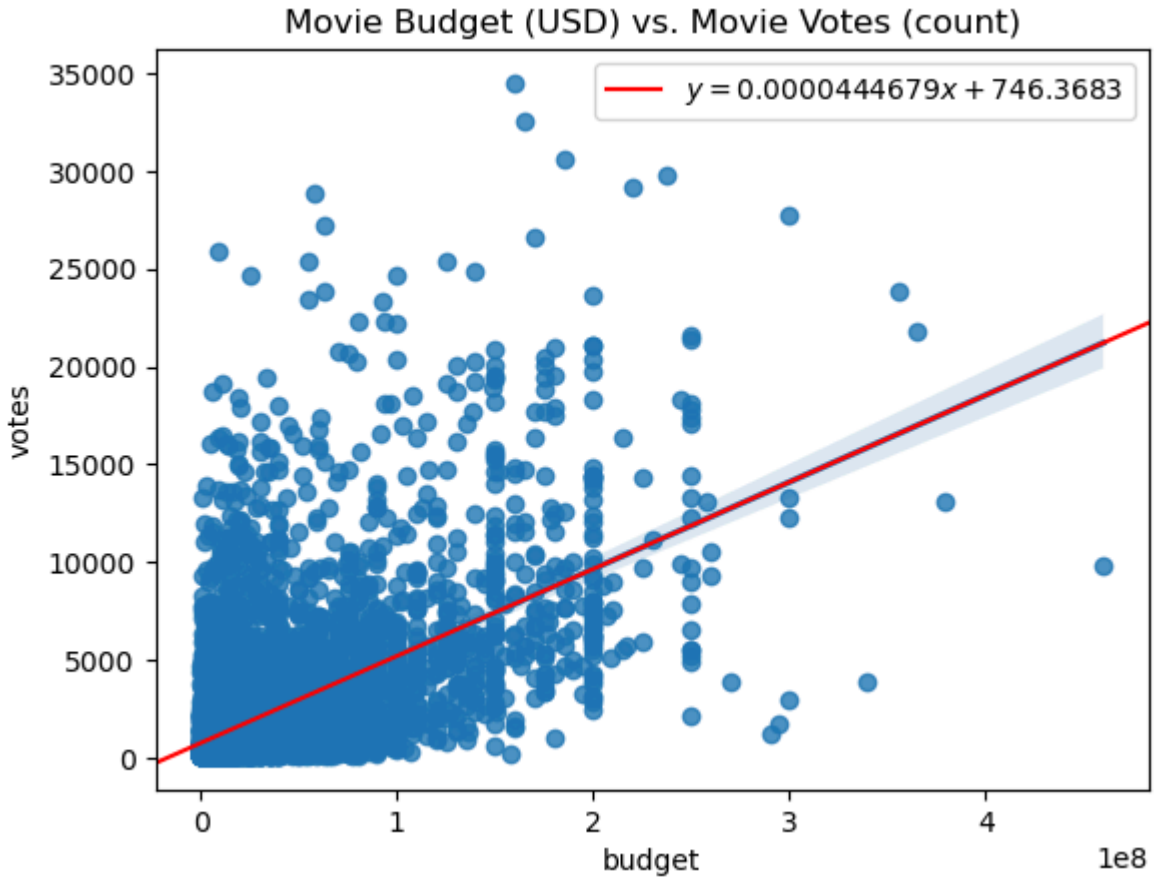


This plot treating rating as a response to budget shows practically no correlation. One might then infer that increasing the movie budget does not necessarily drive movie quality (as suggested by ratings), so even if rating positively affects income, budget may not be used to drive quality necessarily as effectively as other inputs, such as identifying, recruiting and retaining effective production staff, choosing movie genres and/or topics that resonate with viewers, and discerning scripts that are well-written from those that are poorly-written. None of these factors are necessarily driven by budget increases.

In [28]:
```
sns.regplot(x=budget, y=votes)
m, b = np.polyfit(x=budget, y=votes, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.10f}x {b:+.4f}$', color='red')
```

```
plt.title("Movie Budget (USD) vs. Movie Votes (count)")
plt.legend(loc='upper right')
```

Out[28]: <matplotlib.legend.Legend at 0x70b985a5dcd0>



This plot treating votes as a response to budget shows a positive correlation. According to the linear regression line, a roughly 25,000 USD increase in budget corresponds to an increase of one vote (from a baseline of 746 votes for a budget of 0). This differential estimates that best-selling movie (Avatar, 2009) with a budget of 237,000,000 USD would garner 9,480 votes, which seems reasonable considering that most viewers do not vote on TMDB (the dataset source).

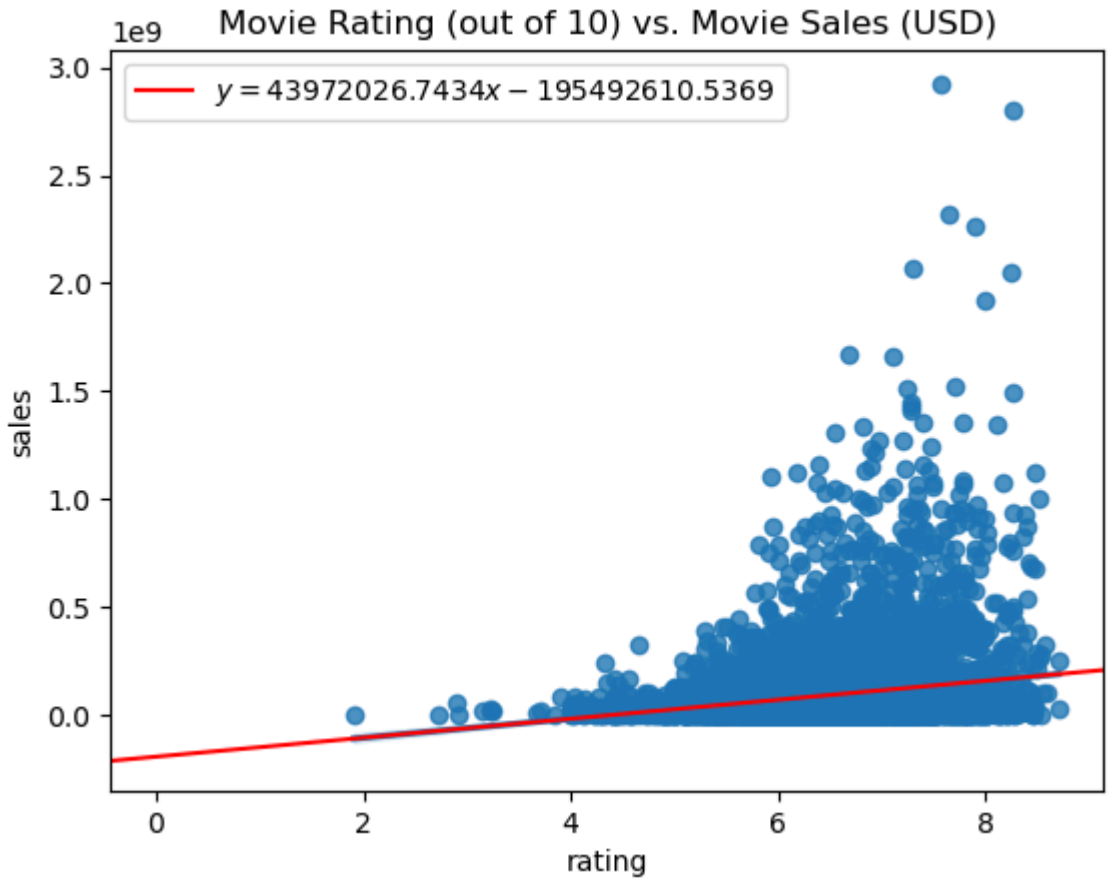
While budget seems disconnected to ratings response, this result suggests that voters tend to vote more frequently as more budget is allocated to a movie. From practical knowledge, one might infer that movie budget increase typically corresponds with more marketing,

contributing to greater exposure to the average viewer, inducing more movie sales, which increases viewership, and which expands the base of possible voters.

If (responsibly-allocated) budget drives votes and votes drive ratings, do ratings drive sales?

In [29]:
```
sns.regplot(x=rating, y=sales)
m, b = np.polyfit(x=rating, y=sales, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Rating (out of 10) vs. Movie Sales (USD)")
plt.legend(loc='upper left')
```

Out[29]:
```
<matplotlib.legend.Legend at 0x70b986450590>
```



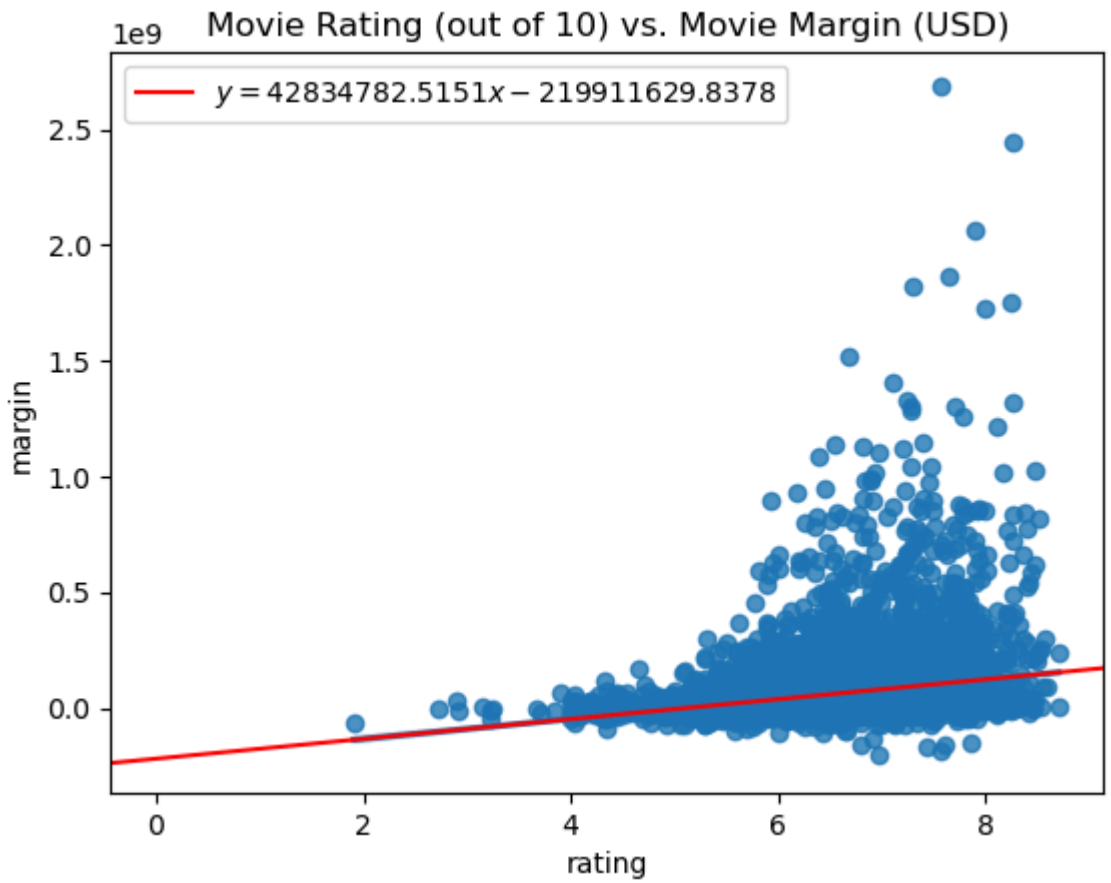
This plot treating rating as a response to sales shows a reasonably positive correlation. According to the linear regression line, a one-point increase in rating corresponds with a 43,972,027 USD increase in sales (from a baseline of -195,492,610 USD at a rating of 0), so while rating may not necessarily be well-controlled by increasing the budget, finding ways to increase viewer satisfaction (e.g. movie quality) should positively affect sales, which is reasonable considering that word of mouth can circulate quickly enough to dissuade potential viewers who seldom ever are interested in paying for a bad movie.

Comparing a parameter with margin after comparing the same parameter with sales is prudent due to a variety of conditions, including the following:

- When the linear regression slope with the sales response is less than the inverse of the linear regression slope with the budget driver (inverting sets budget as the response so as to compare with the sales response). If a budget response increases at a faster rate than sales response, then the margin response could turn negative (depending on the relative positioning of y-intercepts).
- When the inverse of the linear regression slope with the budget driver is greater than 1, even with the linear regression slope of the sales response being greater, in order to assess the degree to which budget offsets sales and erodes margin growth relative to sales growth.
- When the linear regression y-intercept with a budget response is greater than the linear regression y-intercept with the sales response, in order to assess the degree at which margin starts in the negative.

In [30]:
```
sns.regplot(x=rating, y=margin)
m, b = np.polyfit(x=rating, y=margin, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Rating (out of 10) vs. Movie Margin (USD)")
plt.legend(loc='upper left')
```

Out[30]:
```
<matplotlib.legend.Legend at 0x70b9851237a0>
```



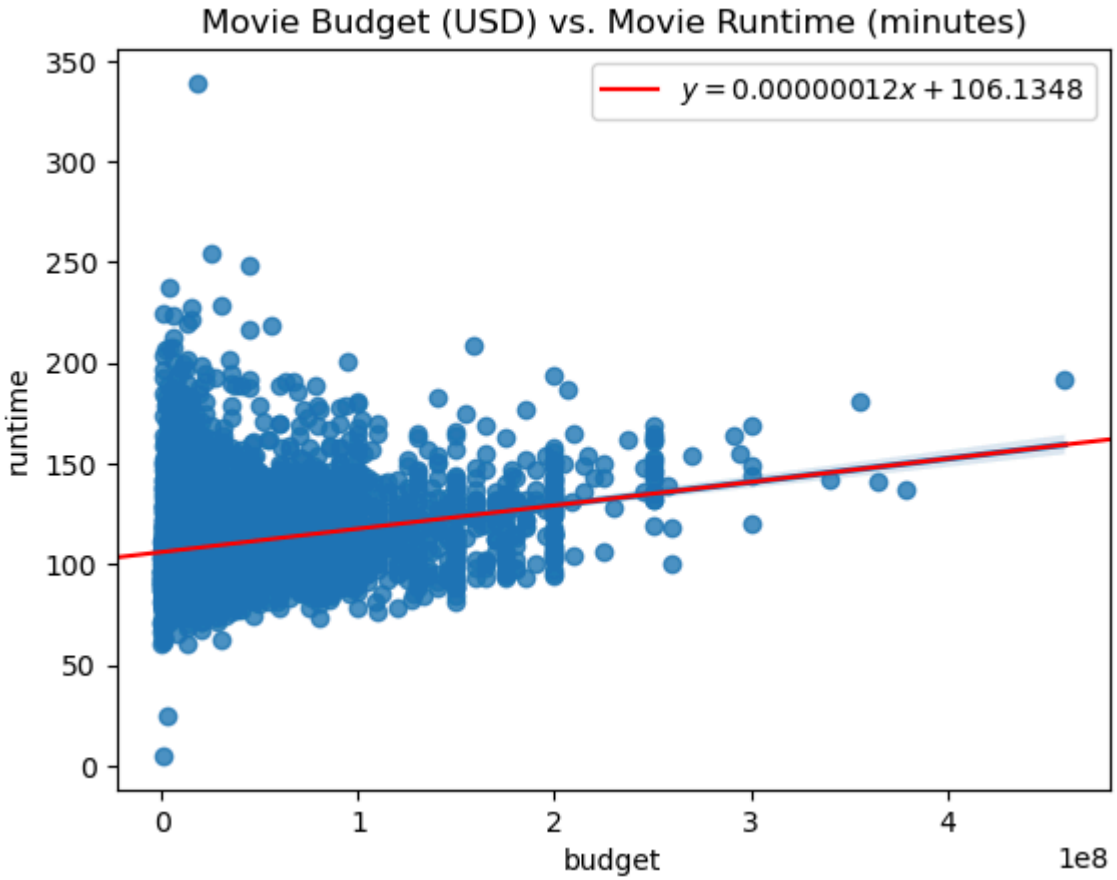
This plot treating rating as a response to margin shows a reasonably positive correlation that is nearly identical to the positive correlation shown for sales (which is reasonable considering very little correlation with budget), supporting the expectation that improving viewer satisfaction (reflected in ratings) leads to better margin.

The preceding results have shown how budget can drive votes, which can drive rating, which can drive sales as well as margin. Herein, the parameter of runtime is assessed as a possible driver of sales, either directly, or indirectly (in the latter case, as by driving another parameter like rating).

In [31]:
```
sns.regplot(x=budget, y=runtime)
m, b = np.polyfit(x=budget, y=runtime, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.8f}x {b:+.4f}$', color='red')
```

```
plt.title("Movie Budget (USD) vs. Movie Runtime (minutes)")
plt.legend()
```

Out[31]: <matplotlib.legend.Legend at 0x70b98513dac0>

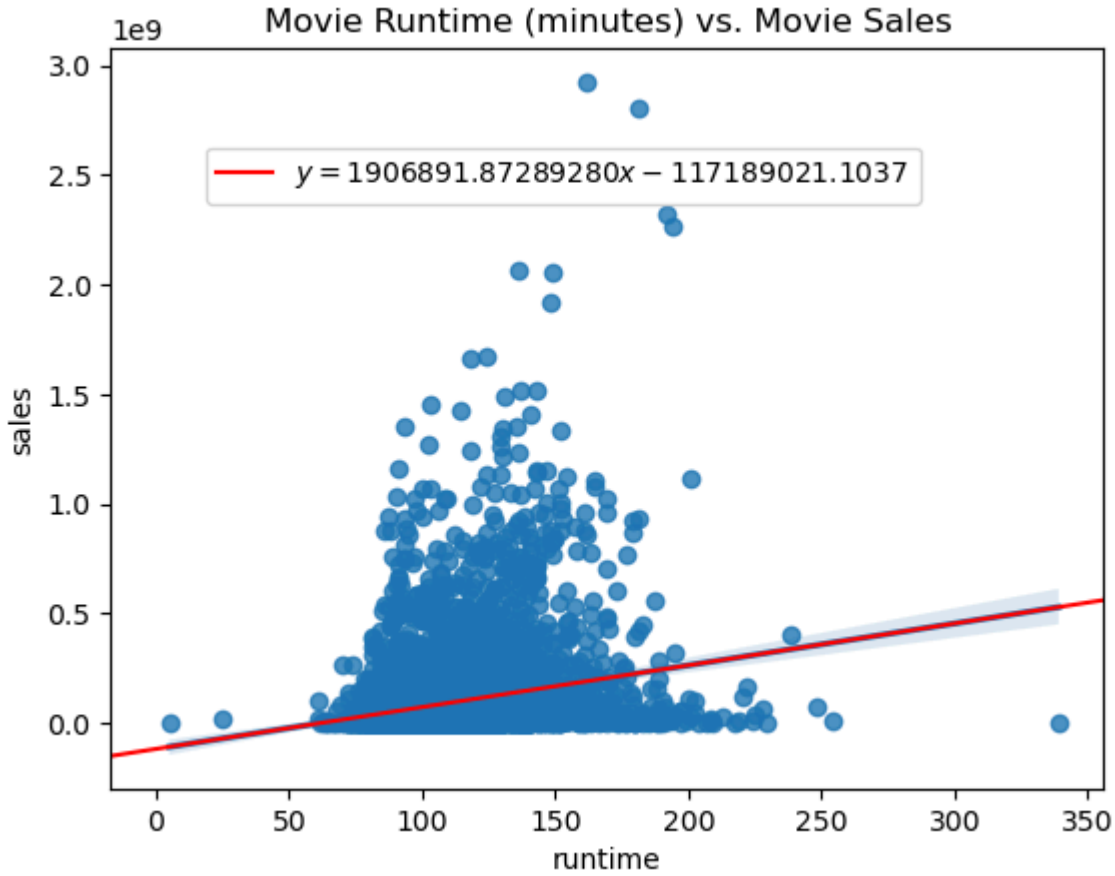


This plot treating runtime as a response to budget shows a reasonably positive correlation, which is reasonable considering the additional resources required in order to extend a movie runtime without compromise to other factors such as movie quality. If a longer runtime correlates with viewer satisfaction, and if viewer satisfaction correlates with greater margins, then the additional cost of extending a movie runtime may be useful for increasing margin.

In [32]:
```
sns.regplot(x=runtime, y=sales)
m, b = np.polyfit(x=runtime, y=sales, deg=1)
```

```
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.8f}x {b:+.4f}$', color='red')
plt.title("Movie Runtime (minutes) vs. Movie Sales")
plt.legend(loc='upper right', bbox_to_anchor=(0.33,0.385,0.5,0.5))
```

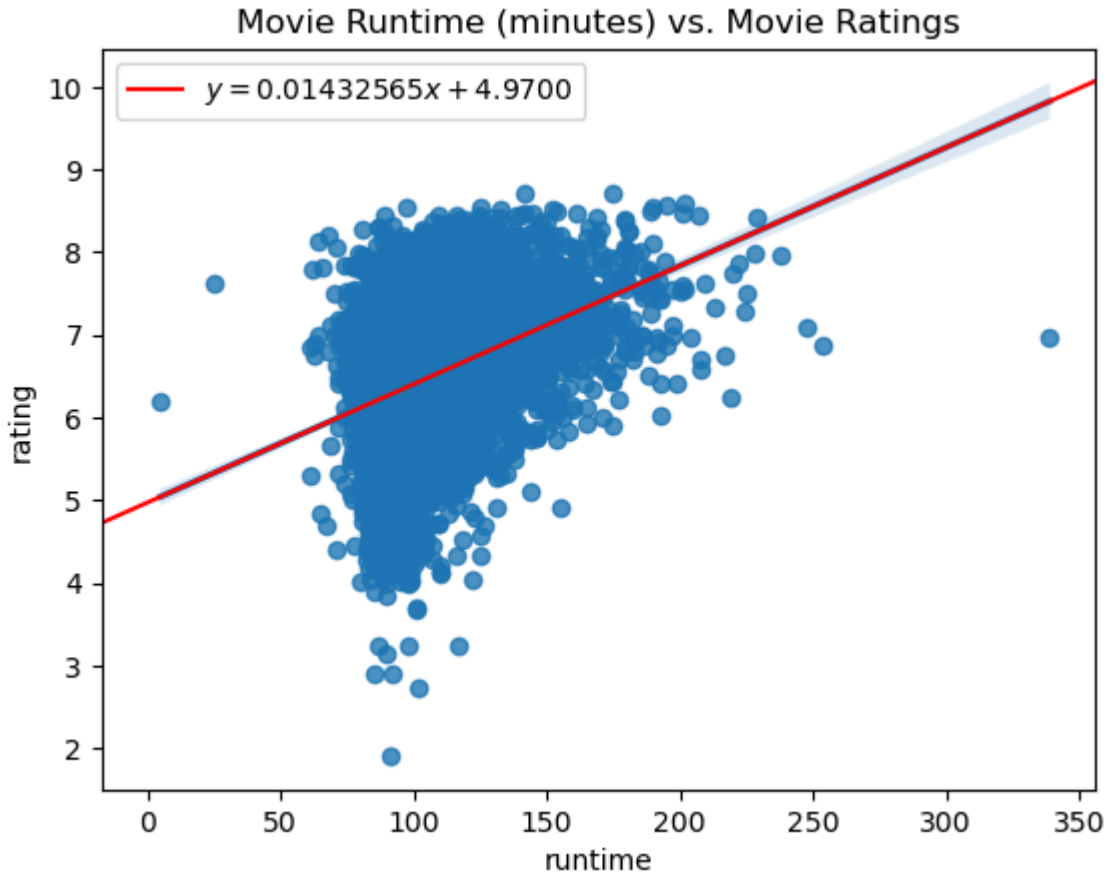
Out[32]: <matplotlib.legend.Legend at 0x70b985b6a1b0>



This plot treating runtime as a resonably positive correlation that is weighed down by the few results with runtime over 225 minutes which did not sell well. With most movies timing at between 90 and 150 minutes, this result seems to suggest that extending a movie from 90 to 150 minutes might result in greater sales, which might also be correlated with higher viewer satisfaction, possibly from viewers associating a longer movie with better value. If true, viewers perceiving better value could drive higher ratings. This possibility could be further substantiated if higher ratings correlated with longer runtimes.

In [33]:
```
sns.regplot(x=runtime, y=rating)
m, b = np.polyfit(x=runtime, y=rating, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.8f}x {b:+.4f}$', color='red')
plt.title("Movie Runtime (minutes) vs. Movie Ratings")
plt.legend(loc='upper left')
```

Out[33]: <matplotlib.legend.Legend at 0x70b98518e7b0>



This plot treating runtime as a response to rating shows a positive correlation. A viewer who values their own scarce time might be more inclined to judge a longer movie more harshly, yet in these results, no movie running over 200 minutes was rated on average lower than a 5. However, not many movies running over 200 minutes were included in this dataset as is evident from this chart and further evidenced by a

count (yielding 22 movies over 200 minutes), possibly suggesting that longer movies with poor reviews did not receive the minimum number of votes (99) that this dataset was filtered on during the preparation stage of this study.

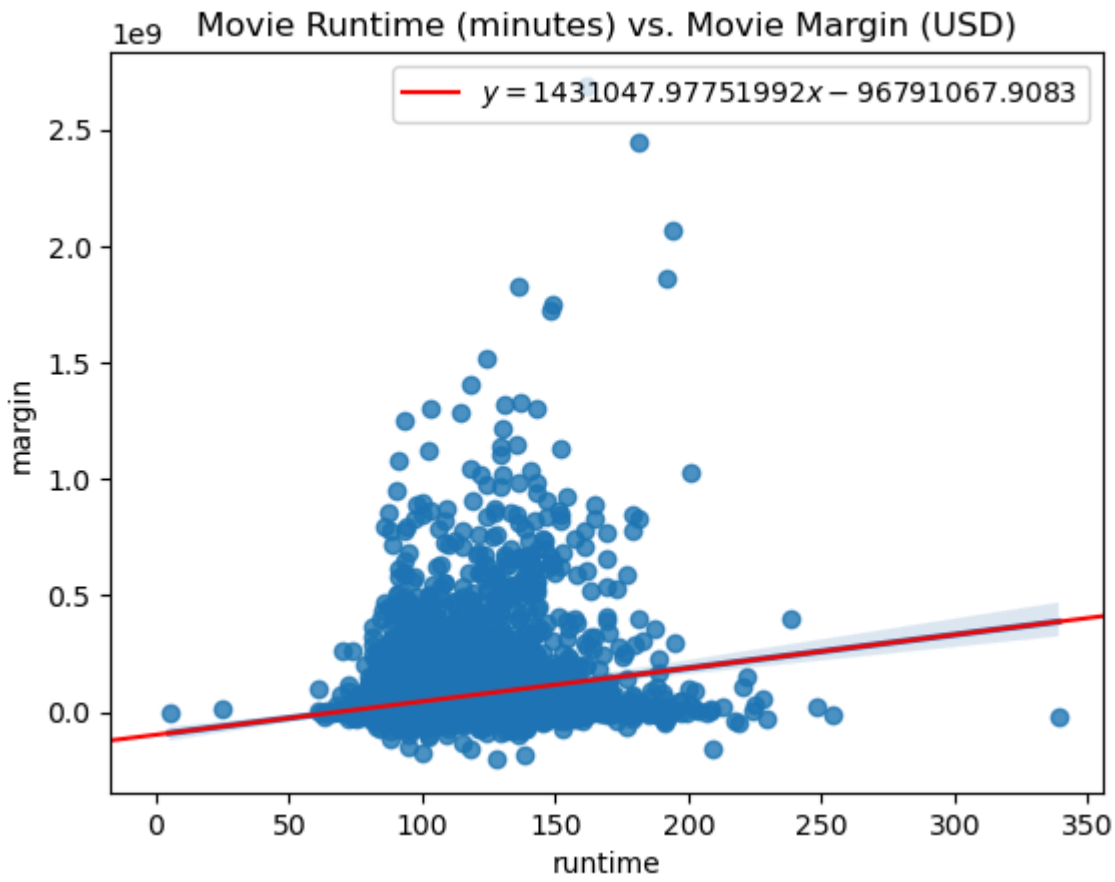
In [34]:
```
runtime[runtime > 200].count()
```

Out[34]:
```
np.int64(22)
```

Between 90 to 150 minutes, where most of the data resides, a positive correlation between rating and runtime is evident, further evidencing as suggested from the earlier plot that a movie that is too short is more likely to receive negative reviews, whereas 150 minutes seems to be considered a satisfactory length.

In [35]:
```
sns.regplot(x=runtime, y=margin)
m, b = np.polyfit(x=runtime, y=margin, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.8f}x {b:+.4f}$', color='red')
plt.title("Movie Runtime (minutes) vs. Movie Margin (USD)")
plt.legend()
```

Out[35]:
```
<matplotlib.legend.Legend at 0x70b984ef42f0>
```



This plot treating runtime as a response to margin shows a reasonably positive correlation. One reason for this possible relationship is that in some of these cases viewers are associating a longer movie with better value, providing better ratings (and word of mouth) which reflects on sales as the effect of a higher quality movie. However, no movies running over 201 minutes netted more than 500 million, with most movies bearly breaking even, in spite of the higher reviews, and likely due to the fact that fewer people have the patience to sit in a theater for nearly three and a half hours.

In [36]:
```
movies[runtime > 201][margin > 500000000][['id']].size
```

Out[36]:
```
0
```

In order to further validate earlier inferences, some of the same plots as above are produced below restricting the dataset to movies produced from the beginning of 2017 through the end of 2022. In addition, results for the popularity parameter, tracked only for recent movies, are included in order to further substantiate the parameters which popularity itself considers in its own computation.

In [37]:
```
dates = pd.to_datetime(movies.date)
dates = dates.dt.strftime('%Y')
movies.date = dates
movies = movies[movies.date > '2016'][movies.date < '2023']
movies = movies
movies[['id']].size
```

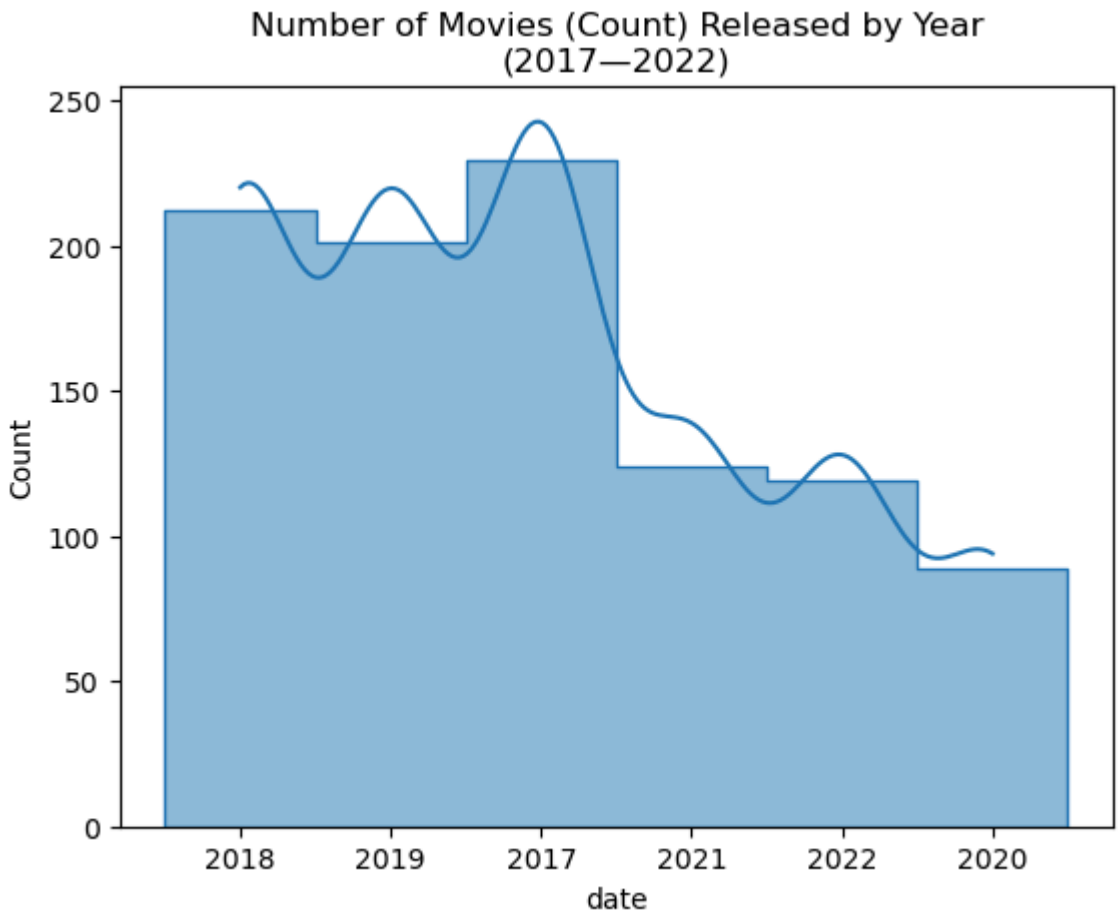
Out[37]:
```
974
```

In [38]:
```
date = movies.date
budget = movies.budget
votes = movies.votes
rating = movies.rating
runtime = movies.runtime
popularity = movies.popularity
sales = movies.sales
margin = movies.margin
```

Before performing the regression analysis as above with the time-restricted dataset, movies are counted for each year, revealing a sharp decline since 2020, which marked the beginning of the global pandemic. This result explains the flatter probability densities for those same years in the subsequent histogram. However, the same histogram shows that the density curves center at around the same rating, which is further reflected in the subsequent boxplot showing nearly the same median rating values for each year.

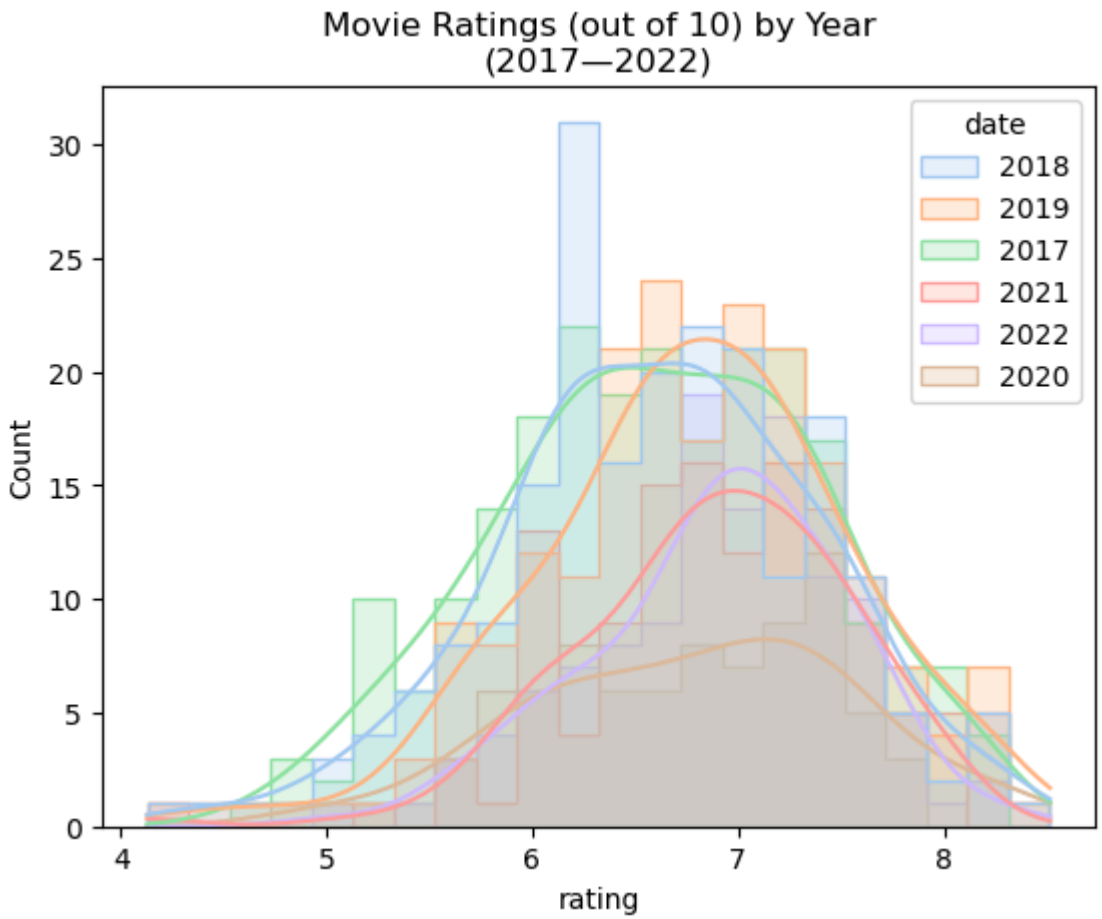
In [39]:
```
sns.histplot(x=date, kde=True, element = 'step')
plt.title("Number of Movies (Count) Released by Year\n(2017–2022)")
```

Out[39]:
```
Text(0.5, 1.0, 'Number of Movies (Count) Released by Year\n(2017–2022)')
```



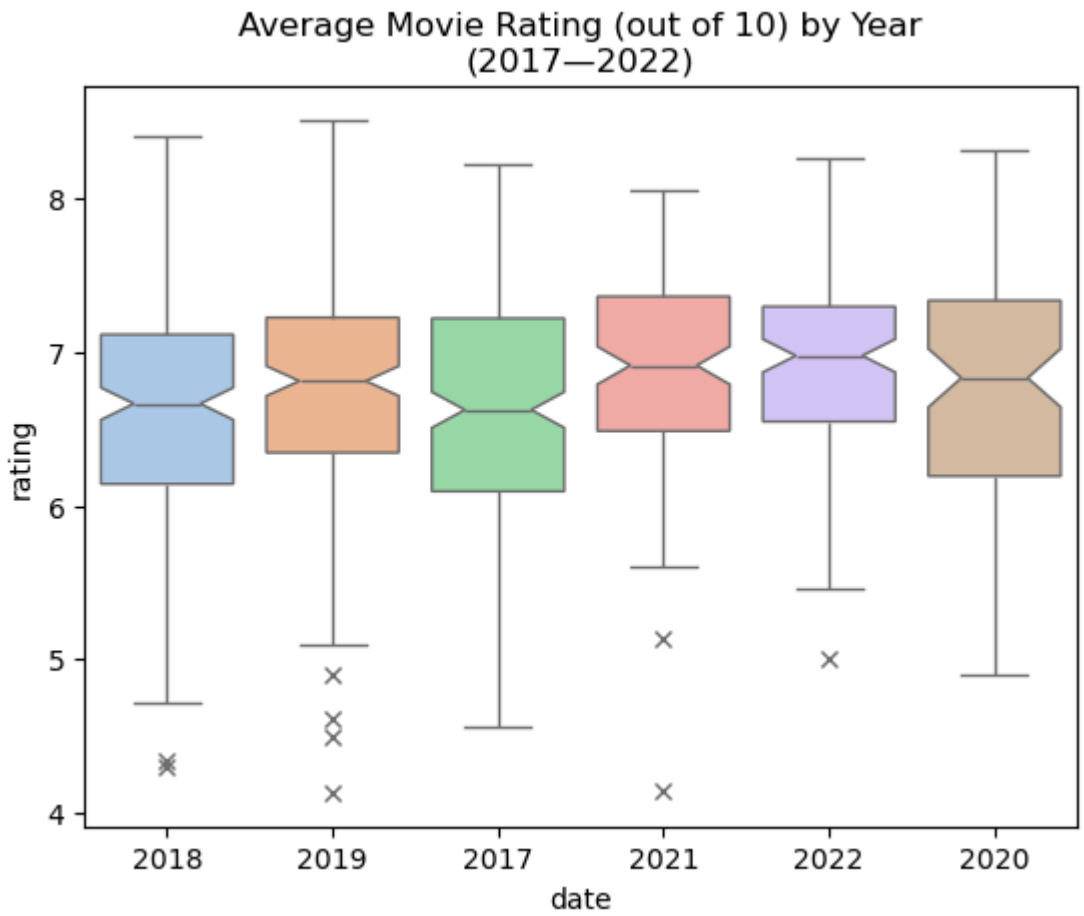
In [40]:
```
sns.histplot(x=rating, kde=True, hue=date, element = 'step', palette=sns.color_palette('pastel', 6))
plt.title("Movie Ratings (out of 10) by Year\n(2017–2022)")
```

Out[40]:
```
Text(0.5, 1.0, 'Movie Ratings (out of 10) by Year\n(2017–2022)')
```



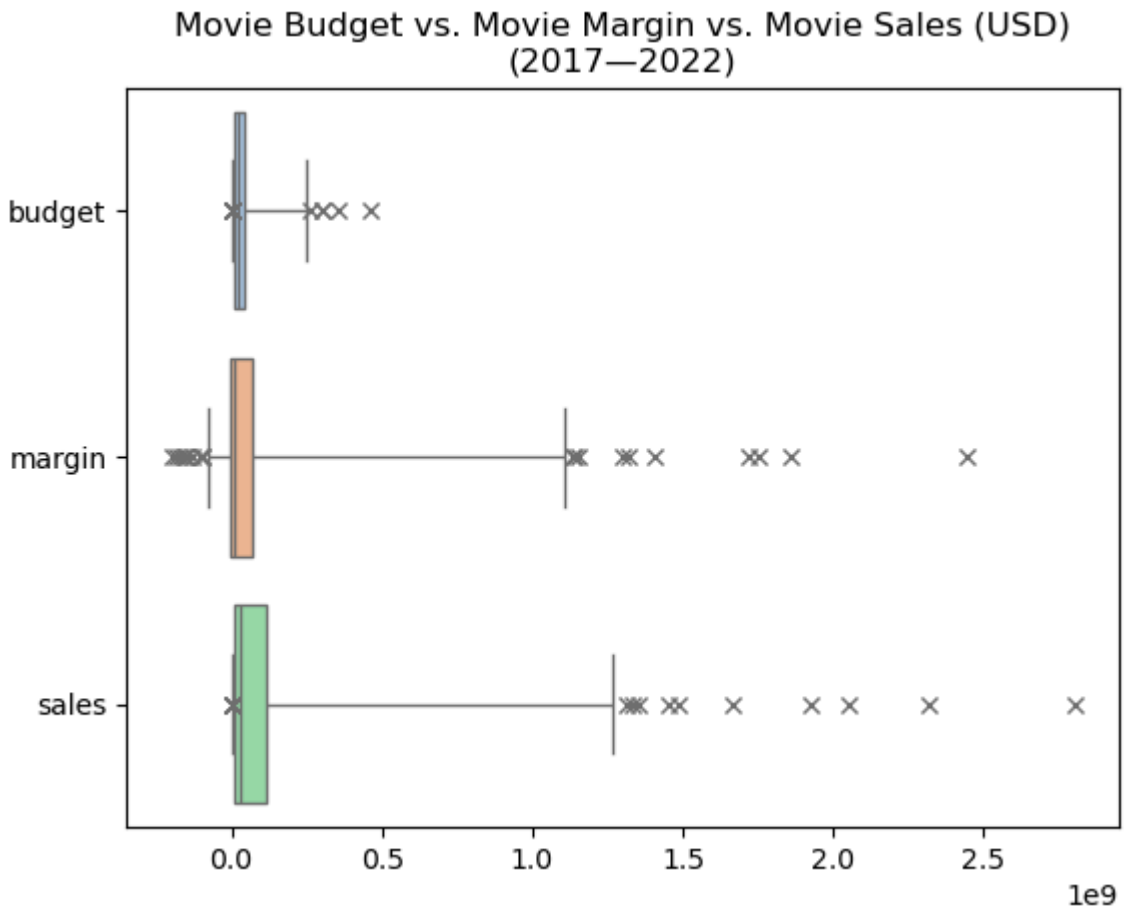
In [41]:
```
sns.boxplot(x=date, y=rating, data=movies, flierprops={'marker':'x'}, notch=True, palette=sns.color_palette('pastel
plt.title("Average Movie Rating (out of 10) by Year\n(2017–2022)")
```

Out[41]:
```
Text(0.5, 1.0, 'Average Movie Rating (out of 10) by Year\n(2017–2022)')
```



In [42]:
```
sns.boxplot(data=movies[['budget','margin','sales']], orient='h', whis=(1,99), flierprops={'marker':'x'}, palette=s
plt.title("Movie Budget vs. Movie Margin vs. Movie Sales (USD)\n(2017–2022)")
```

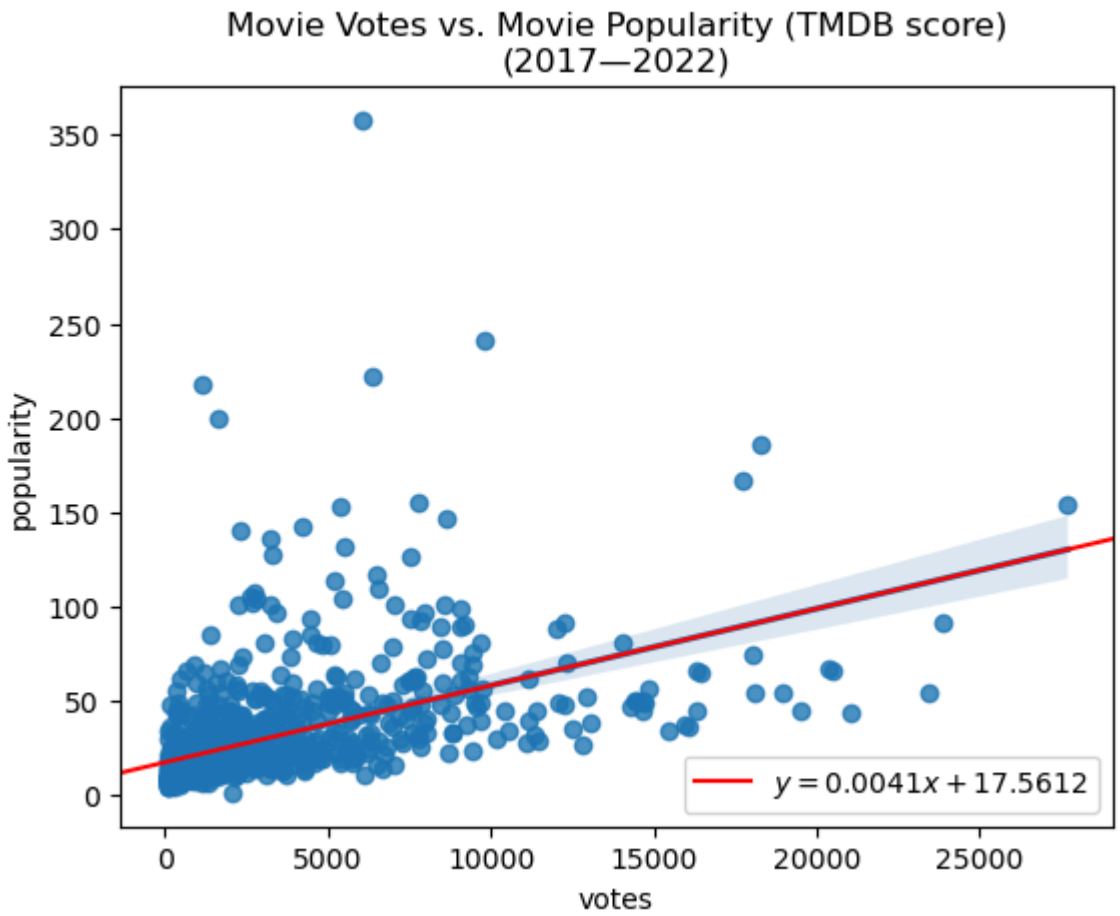
Out[42]:
```
Text(0.5, 1.0, 'Movie Budget vs. Movie Margin vs. Movie Sales (USD)\n(2017–2022)')
```



The time-restricted plots below also compare favorably with the compatible time-unrestricted plots above, with the only difference being, for the plots below, a wider confidence interval resulting from greater error due to a smaller population in each case.

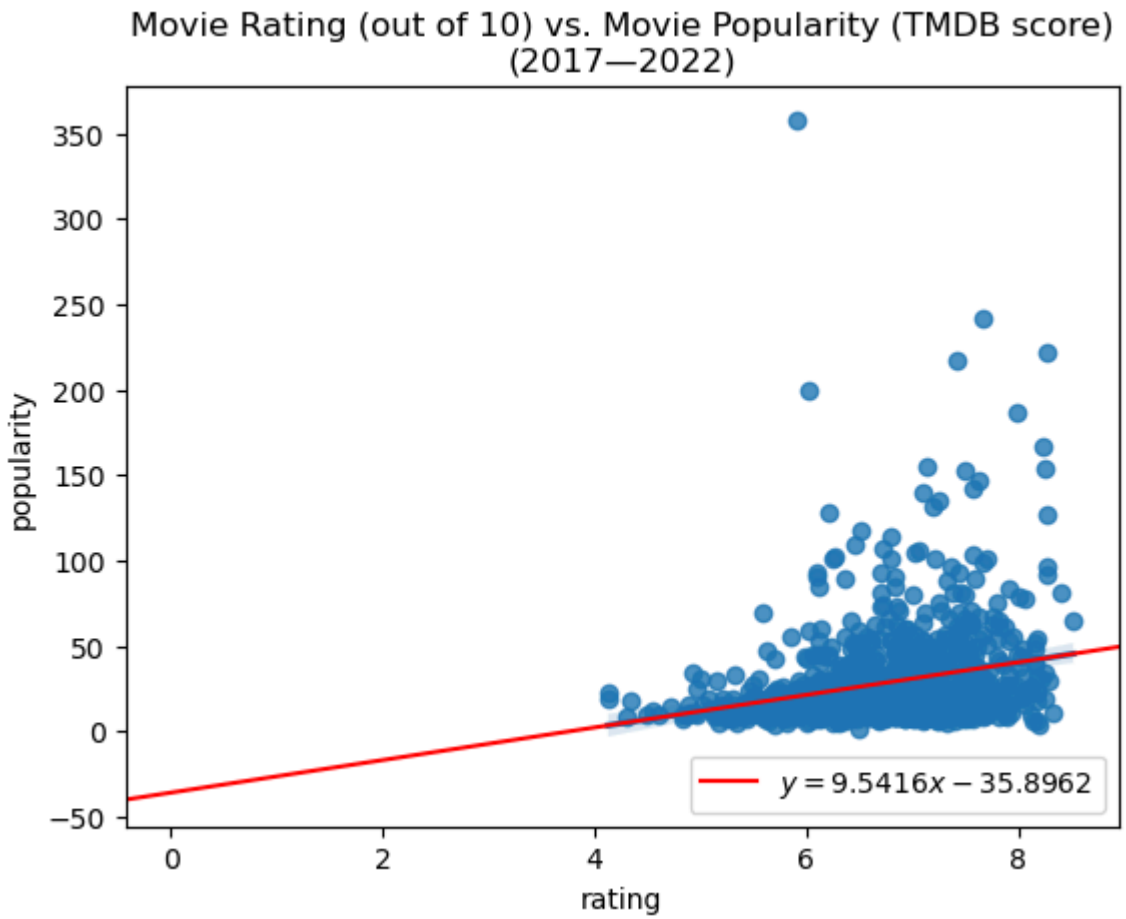
In [43]:
```
sns.regplot(x=votes, y=popularity)
m, b = np.polyfit(x=votes, y=popularity, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Votes vs. Movie Popularity (TMDB score)\n(2017–2022)")
plt.legend(loc='lower right')
```

Out[43]:
```
<matplotlib.legend.Legend at 0x70b9850a23f0>
```



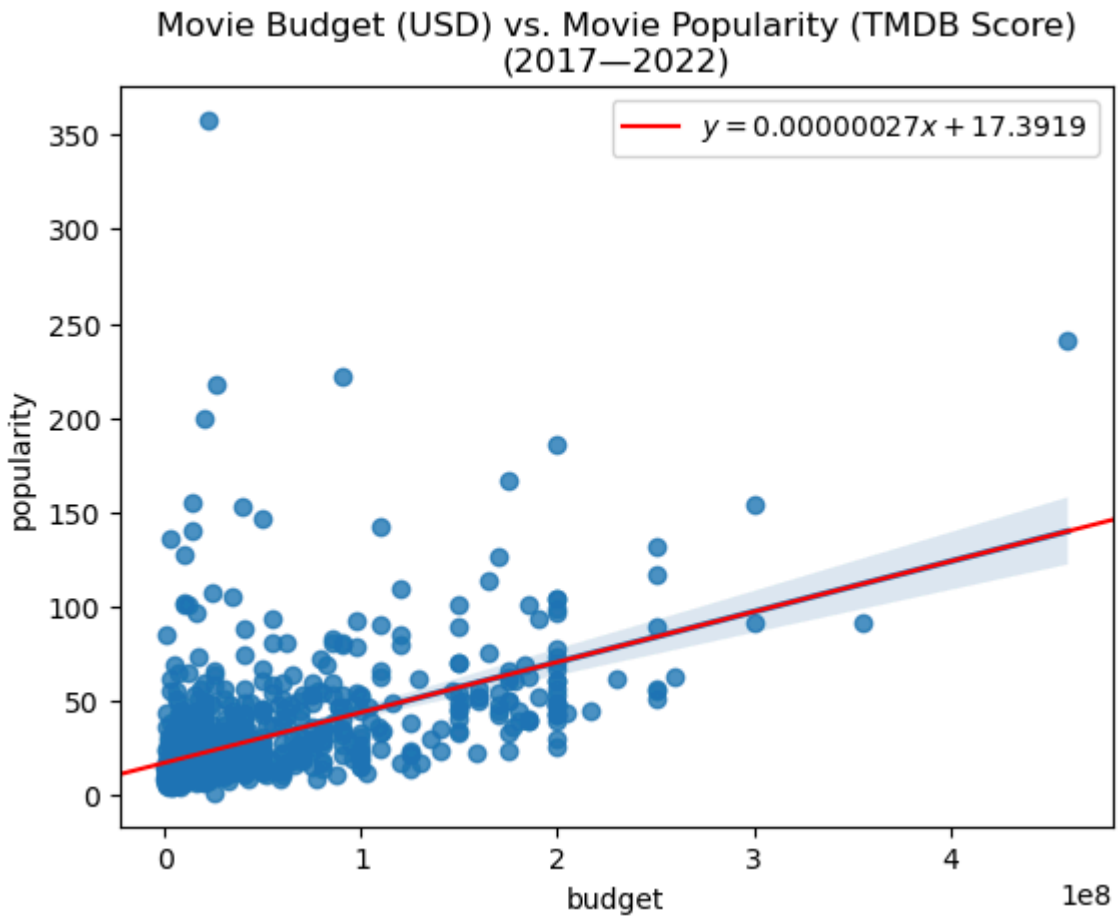
In [44]:
```
sns.regplot(x=rating, y=popularity)
m, b = np.polyfit(x=rating, y=popularity, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Rating (out of 10) vs. Movie Popularity (TMDB score)\n(2017–2022)")
plt.legend(loc='lower right')
```

Out[44]:
```
<matplotlib.legend.Legend at 0x70b980d75250>
```



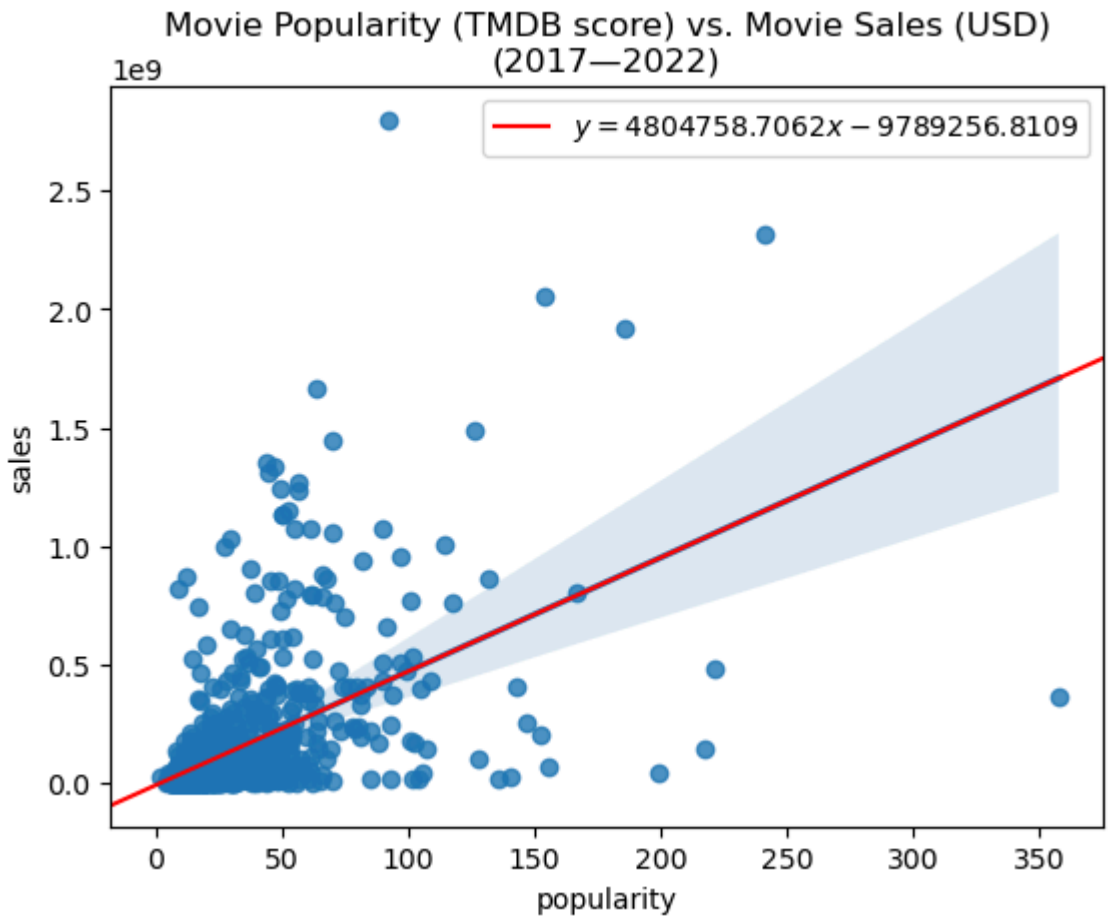
In [45]:
```
sns.regplot(x=budget, y=popularity)
m, b = np.polyfit(budget, popularity, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.8f}x {b:+.4f}$', color='red')
plt.title("Movie Budget (USD) vs. Movie Popularity (TMDB Score)\n(2017–2022)")
plt.legend(loc='upper right')
```

Out[45]:
```
<matplotlib.legend.Legend at 0x70b984dd4e90>
```



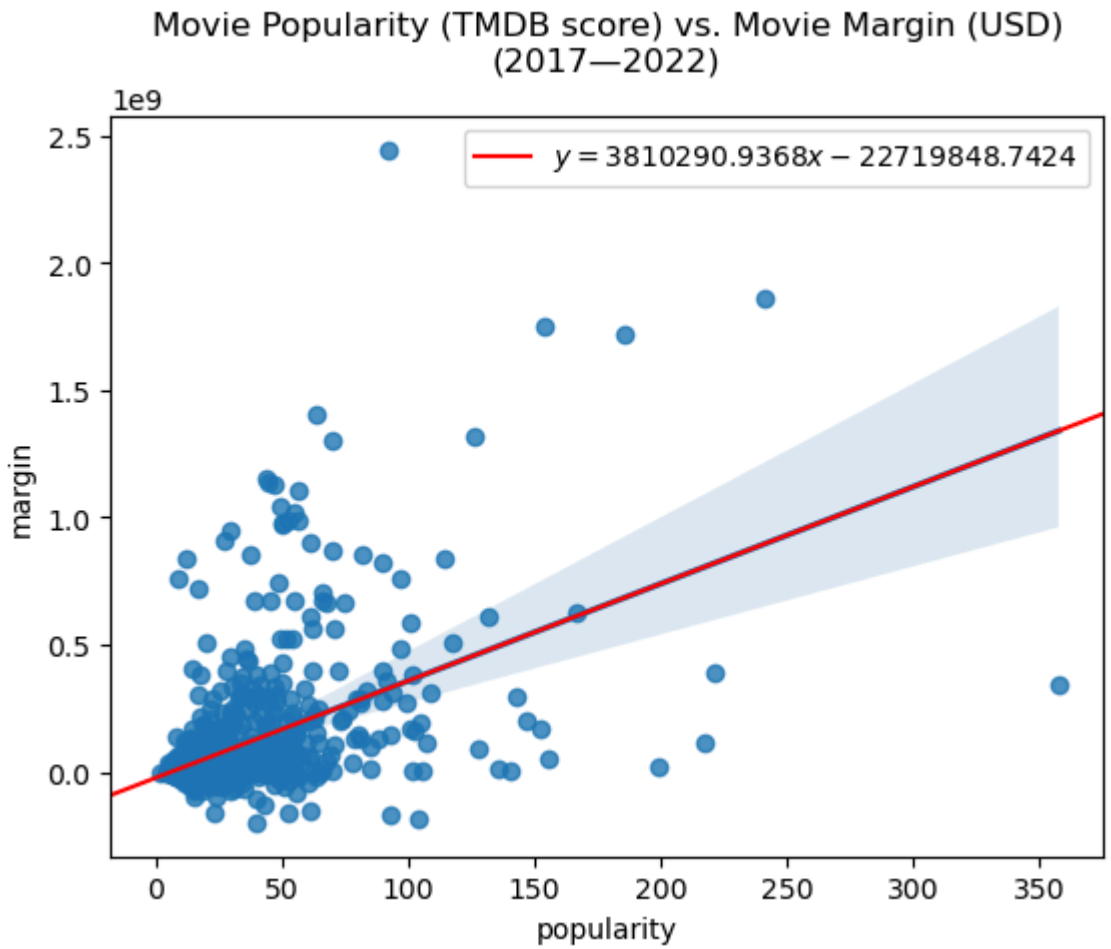
In [46]:
```
sns.regplot(x=popularity, y=sales)
m, b = np.polyfit(x=popularity, y=sales, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Popularity (TMDB score) vs. Movie Sales (USD)\n(2017–2022)")
plt.legend(loc='upper right')
```

Out[46]:
```
<matplotlib.legend.Legend at 0x70b984b49400>
```



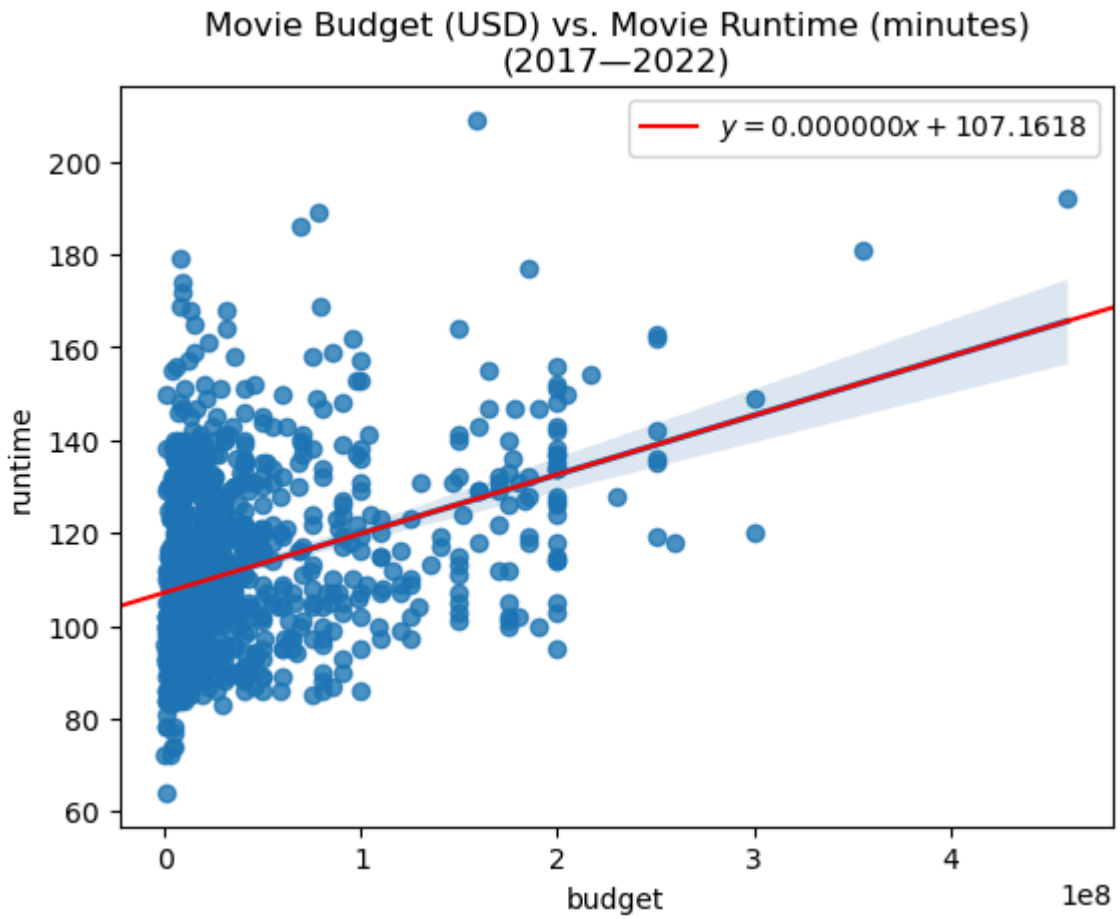
In [47]:
```
sns.regplot(x=popularity, y=margin)
m, b = np.polyfit(x=popularity, y=margin, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Popularity (TMDB score) vs. Movie Margin (USD)\n(2017–2022)")
plt.legend(loc='upper right')
```

Out[47]:
```
<matplotlib.legend.Legend at 0x70b980d46cc0>
```



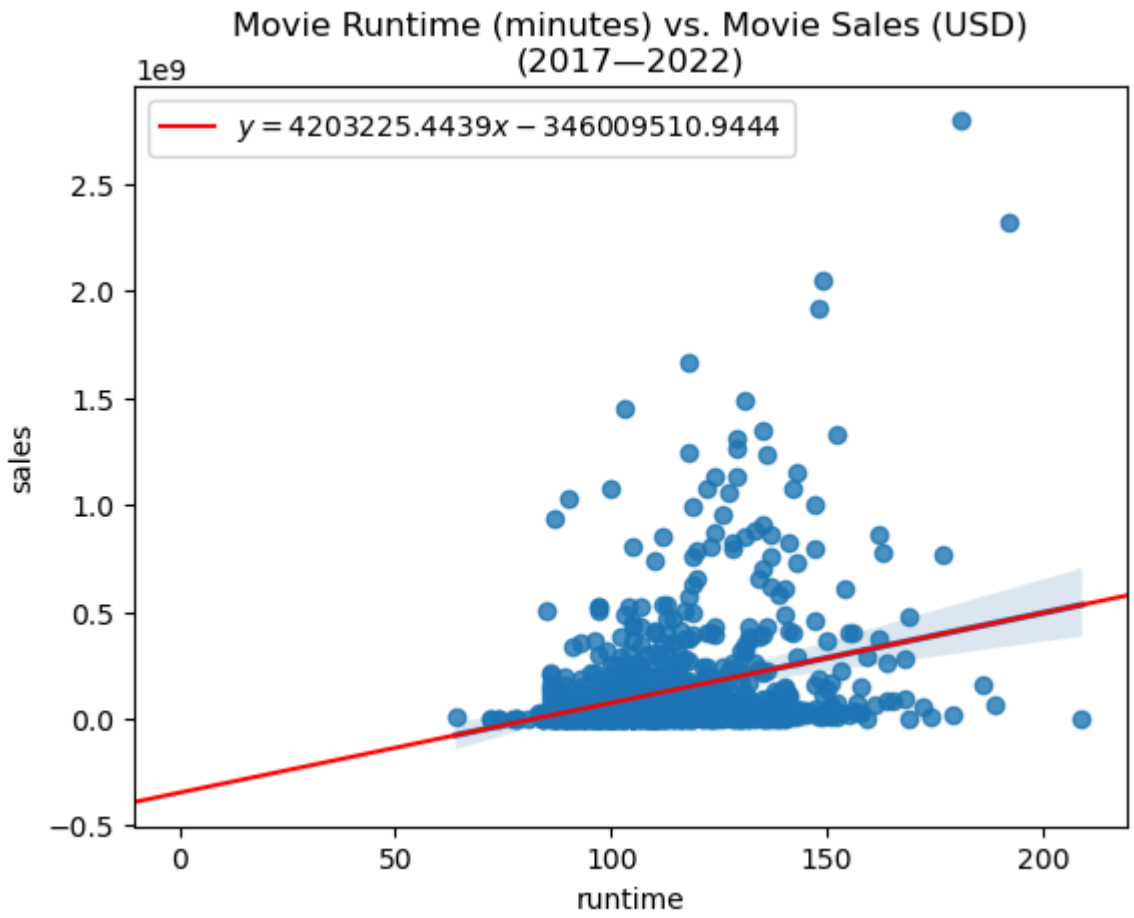
In [48]:
```
sns.regplot(x=budget, y=runtime)
m, b = np.polyfit(x=budget, y=runtime, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.6f}x {b:+.4f}$', color='red')
plt.title("Movie Budget (USD) vs. Movie Runtime (minutes)\n(2017–2022)")
plt.legend(loc='upper right')
```

Out[48]:
```
<matplotlib.legend.Legend at 0x70b980d2a570>
```



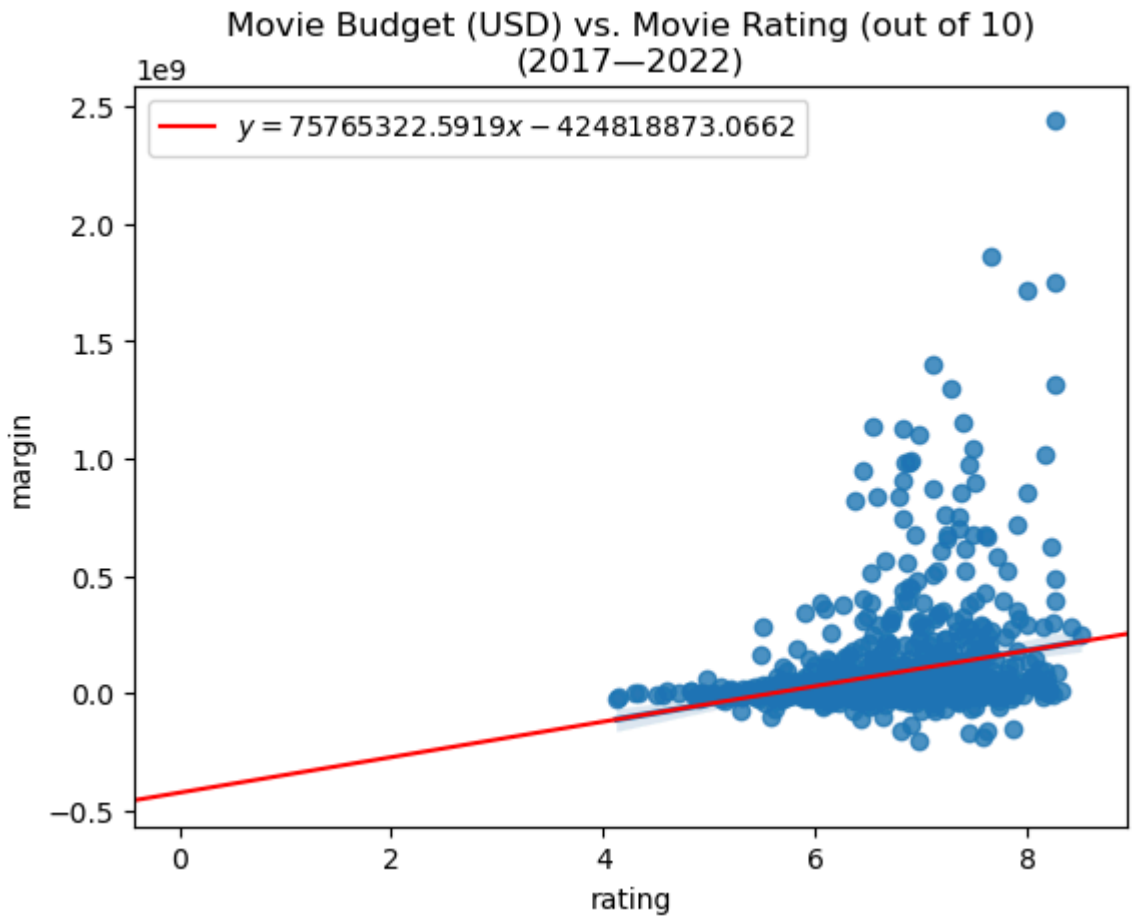
In [49]:
```
sns.regplot(x=runtime, y=sales)
m, b = np.polyfit(runtime, sales, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Runtime (minutes) vs. Movie Sales (USD)\n(2017–2022)")
plt.legend(loc='upper left')
```

Out[49]:
```
<matplotlib.legend.Legend at 0x70b984c5a4e0>
```



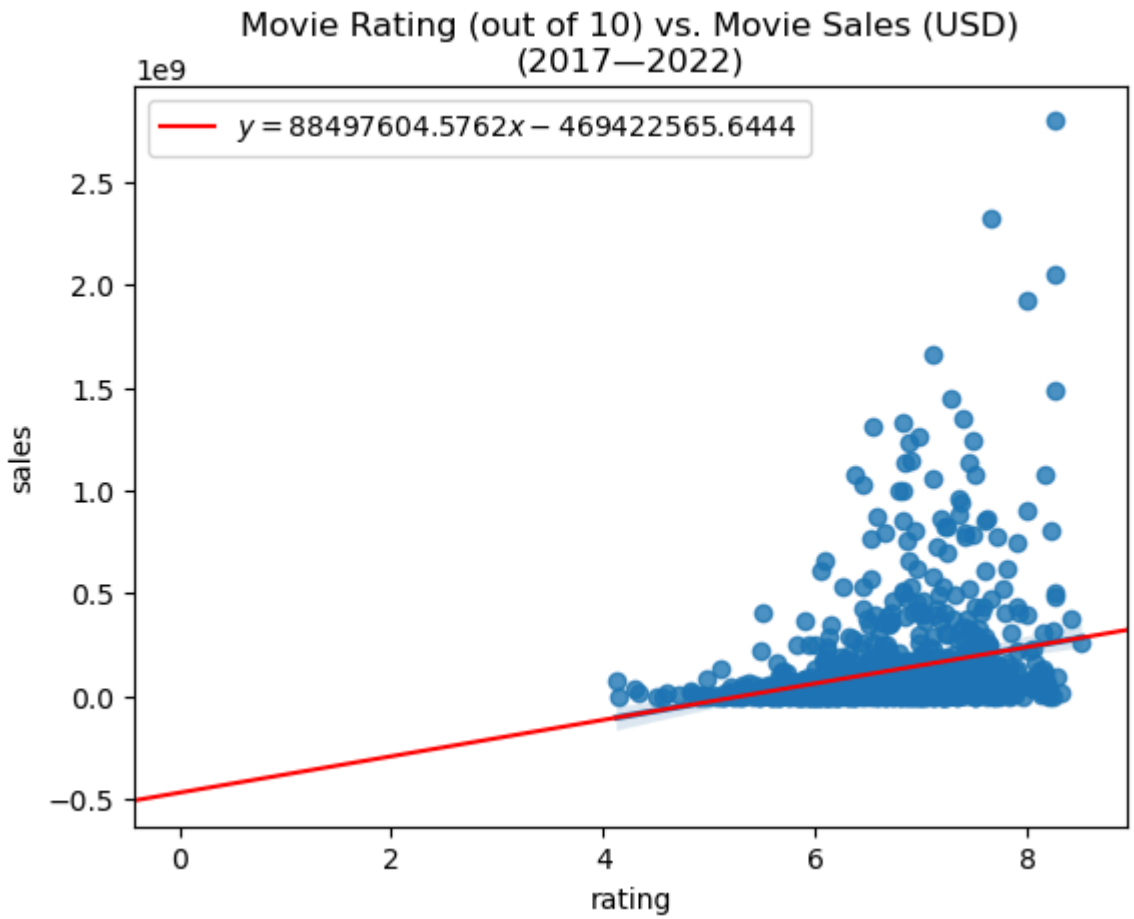
In [50]:
```
sns.regplot(x=rating, y=margin)
m, b = np.polyfit(x=rating, y=margin, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Budget (USD) vs. Movie Rating (out of 10)\n(2017–2022)")
plt.legend(loc='upper left')
```

Out[50]:
```
<matplotlib.legend.Legend at 0x70b95ce71fa0>
```



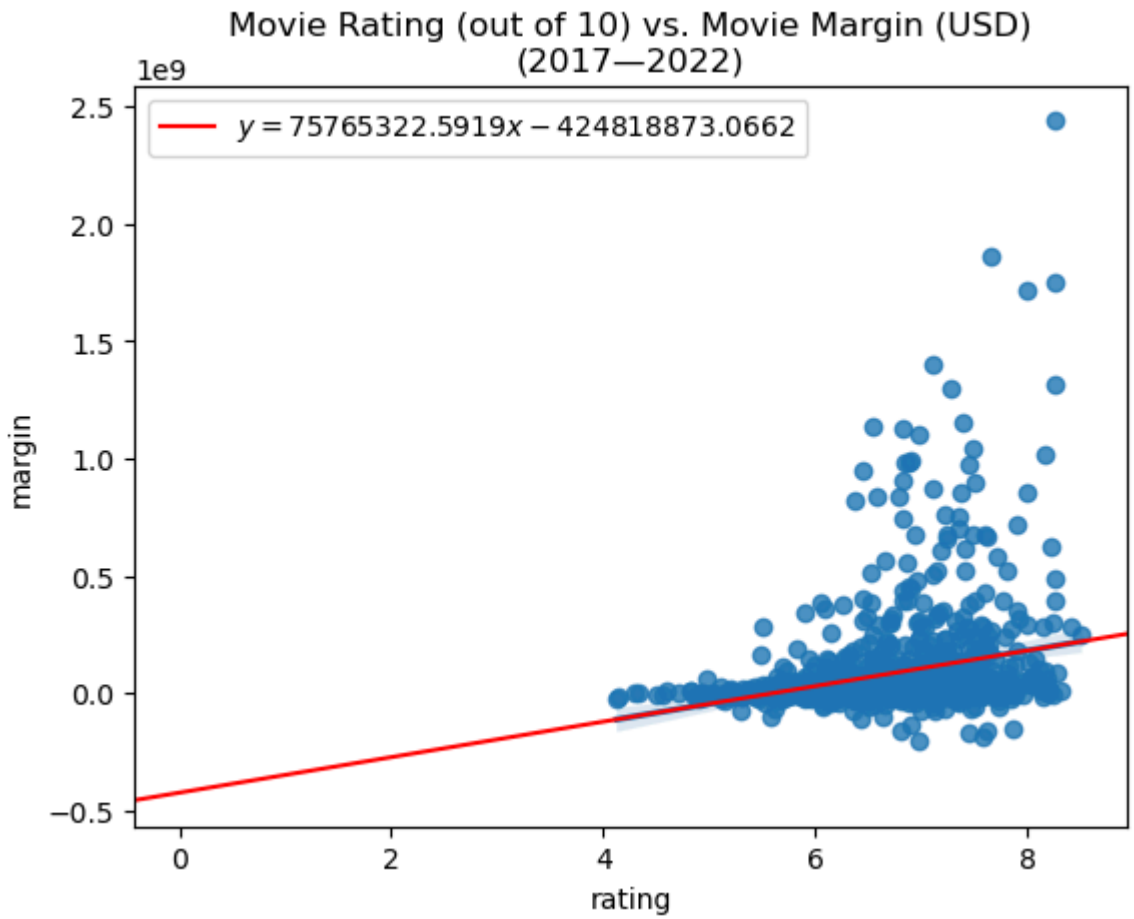
In [51]:
```
sns.regplot(x=rating, y=sales)
m, b = np.polyfit(rating, sales, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Rating (out of 10) vs. Movie Sales (USD)\n(2017–2022)")
plt.legend(loc='upper left')
```

Out[51]:
```
<matplotlib.legend.Legend at 0x70b95cfb30b0>
```



In [52]:
```
sns.regplot(x=rating, y=margin)
m, b = np.polyfit(x=rating, y=margin, deg=1)
plt.axline(xy1=(0,b), slope=m, label=f'$y = {m:.4f}x {b:+.4f}$', color='red')
plt.title("Movie Rating (out of 10) vs. Movie Margin (USD)\n(2017–2022)")
plt.legend(loc='upper left')
```

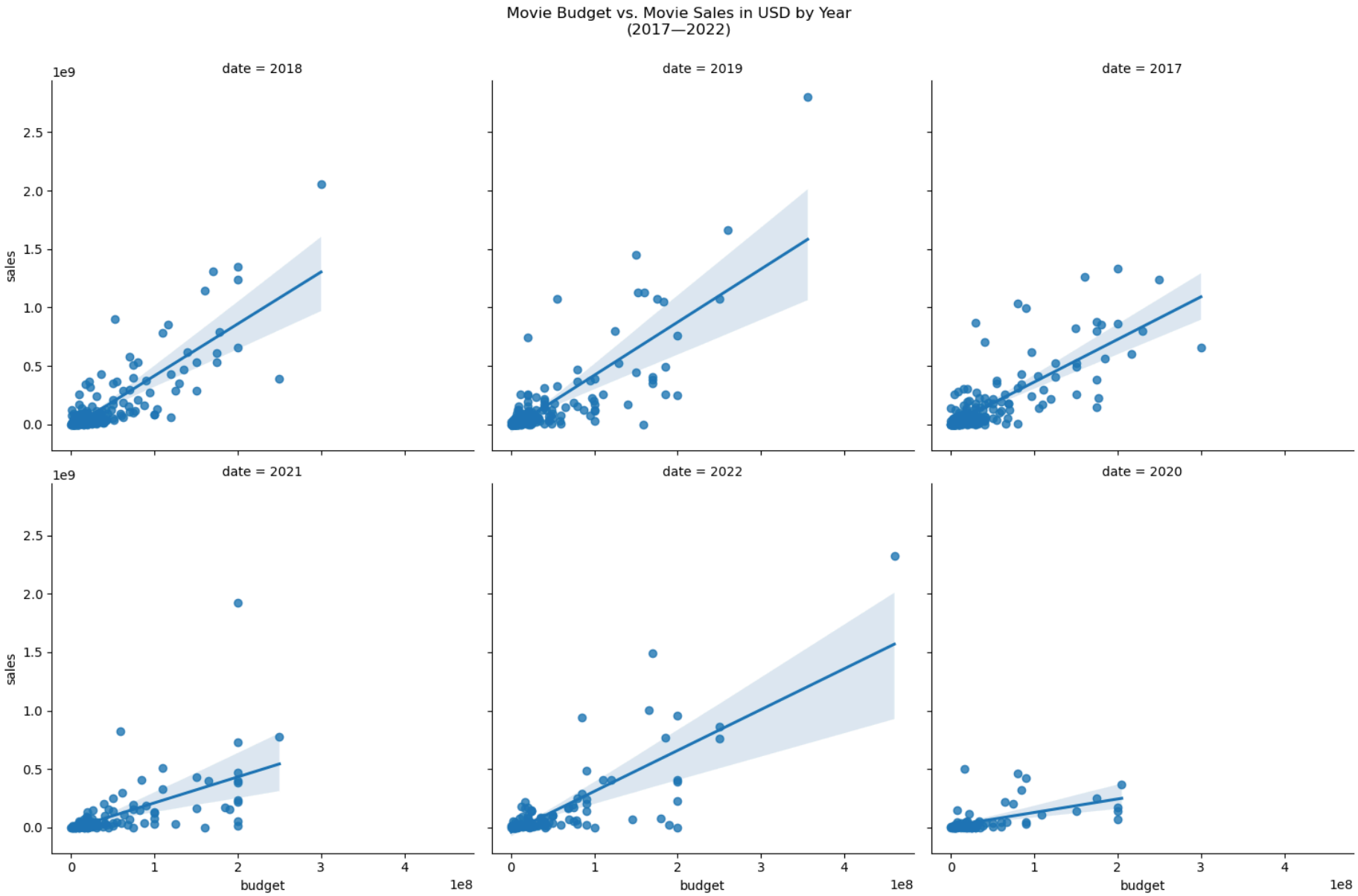
Out[52]:
```
<matplotlib.legend.Legend at 0x70b95cd56570>
```



In order to further substantiate recent trends, some of the plots above are extrapolated into faceted plots below showing the annual results between 2017 and 2022, and for the most part these results align with the results of the corresponding plots above, with the exception that sales are generally down and less responsive to inputs during the years of the height of the global pandemic from 2020 to 2021. In 2022, despite the lingering effects of the pandemic, the correspondence between ratings and sales better aligns with results seen pre-pandemic. One possibile explanation for this result is that after the pandemic, for the average viewer, word of mouth travels faster, which would make sense if the average viewer become more active in social circles on the Internet during the pandemic and maintained these habits afterward.

In [53]:
```
budget_sales_facets = sns.lmplot(x='budget', y='sales', col='date', col_wrap=3, data=movies)
budget_sales_facets.fig.subplots_adjust(top=0.9)
budget_sales_facets.fig.suptitle("Movie Budget vs. Movie Sales in USD by Year\n(2017–2022)")
```

Out[53]:
```
Text(0.5, 0.98, 'Movie Budget vs. Movie Sales in USD by Year\n(2017–2022)')
```

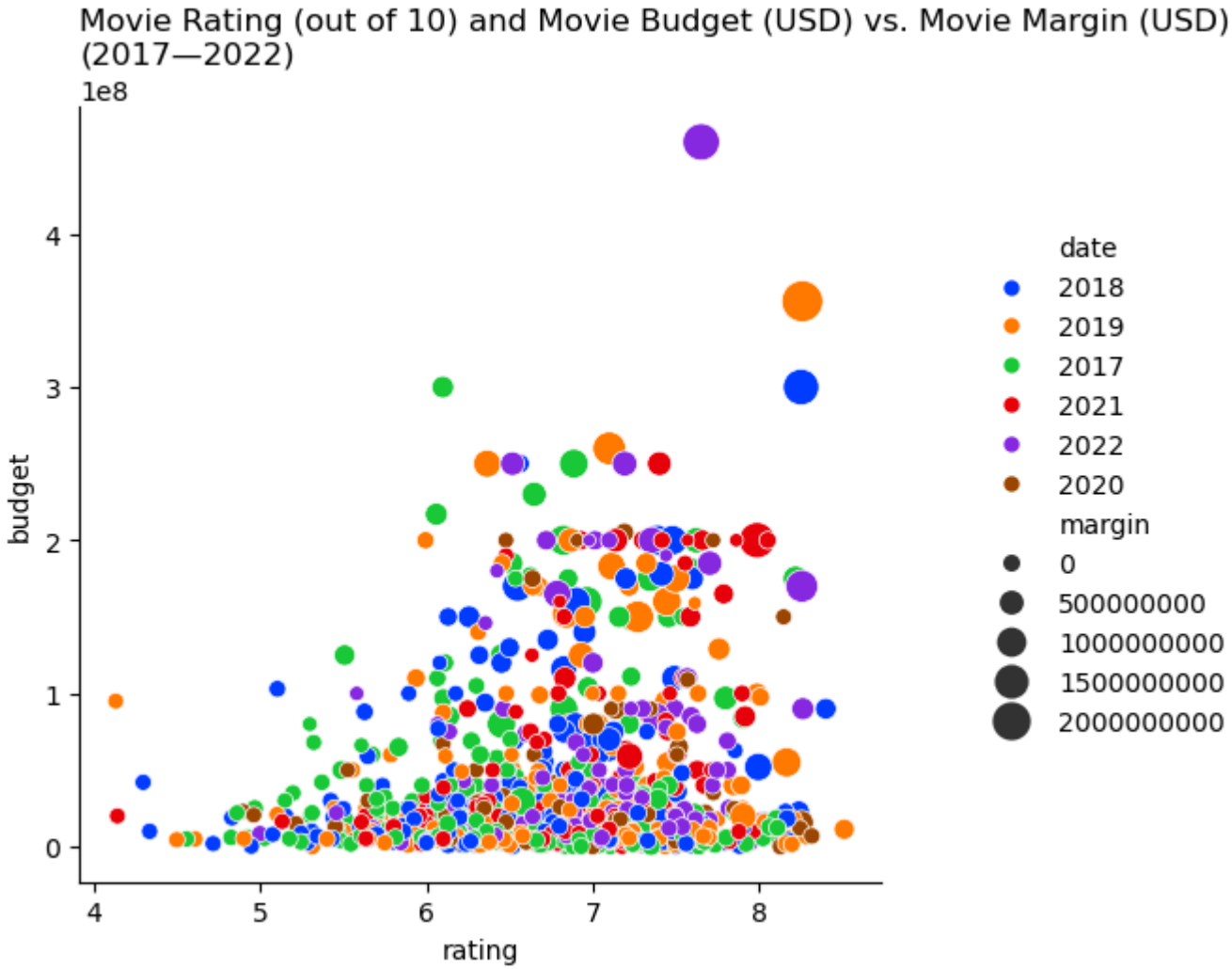


In [54]:
```
rating_sales_facets = sns.lmplot(x='rating', y='margin', col='date', col_wrap=3, data=movies)
rating_sales_facets.fig.subplots_adjust(top=0.9)
rating_sales_facets.fig.suptitle("Movie Rating (out of 10) vs. Movie Margin (USD) by Year\n(2017–2022)")
```

Out[54]: Text(0.5, 0.98, 'Movie Rating (out of 10) vs. Movie Margin (USD) by Year\n(2017–2022)')

In [55]:
```
sns.relplot(x=rating, y=budget, size=margin, sizes=(25,250), hue=date, palette=sns.color_palette('bright', 6))
plt.title("Movie Rating (out of 10) and Movie Budget (USD) vs. Movie Margin (USD)\n(2017–2022)", loc='left')
```

Out[55]: Text(0.0, 1.0, 'Movie Rating (out of 10) and Movie Budget (USD) vs. Movie Margin (USD)\n(2017–2022)')



In [56]:
```
rm_by_margin = movies.sort_values(by='margin', ascending=False)[['title','date','budget','rating','popularity','sal
rm_by_margin.head(20)
```

Out[56]:

	title	date	budget	rating	popularity	sales	margin
15	Avengers: Endgame	2019	356000000	8.263	91.756	2800000000	2444000000
282	Avatar: The Way of Water	2022	460000000	7.654	241.285	2320250281	1860250281
6	Avengers: Infinity War	2018	300000000	8.255	154.340	2052415039	1752415039
57	Spider-Man: No Way Home	2021	200000000	7.990	186.065	1921847111	1721847111
317	The Lion King	2019	260000000	7.100	63.351	1663075401	1403075401
444	Top Gun: Maverick	2022	170000000	8.260	126.291	1488732821	1318732821
336	Frozen II	2019	150000000	7.272	70.221	1450026933	1300026933
27	Black Panther	2018	200000000	7.390	43.665	1349926083	1149926083
254	Jurassic World: Fallen Kingdom	2018	170000000	6.548	44.365	1310466296	1140466296
129	Star Wars: The Last Jedi	2017	200000000	6.825	47.241	1332698830	1132698830
108	Beauty and the Beast	2017	160000000	6.967	56.226	1266115964	1106115964
187	Incredibles 2	2018	200000000	7.481	49.215	1242805359	1042805359
18	Joker	2019	55000000	8.168	54.522	1074458282	1019458282
156	Aquaman	2018	160000000	6.900	52.471	1148528393	988528393
288	The Fate of the Furious	2017	250000000	6.887	56.124	1236005118	986005118
117	Captain Marvel	2019	152000000	6.843	50.399	1131416446	979416446
121	Spider-Man: Far From Home	2019	160000000	7.447	49.913	1131927996	971927996
554	Despicable Me 3	2017	80000000	6.451	29.104	1031552585	951552585
163	Jumanji: Welcome to the Jungle	2017	90000000	6.827	26.669	995339117	905339117
333	Toy Story 4	2019	175000000	7.500	61.083	1073394593	898394593

In [57]:
```
{'Budget > 200' :
     {'Margin' : rm_by_margin[rm_by_margin.budget >= 200000000][['margin']].mean(axis='rows', skipna=False).iloc[0]
      'Rating' : rm_by_margin[rm_by_margin.budget >= 200000000][['rating']].mean(axis='rows', skipna=False).iloc[0]
 'Budget > 100' :
```

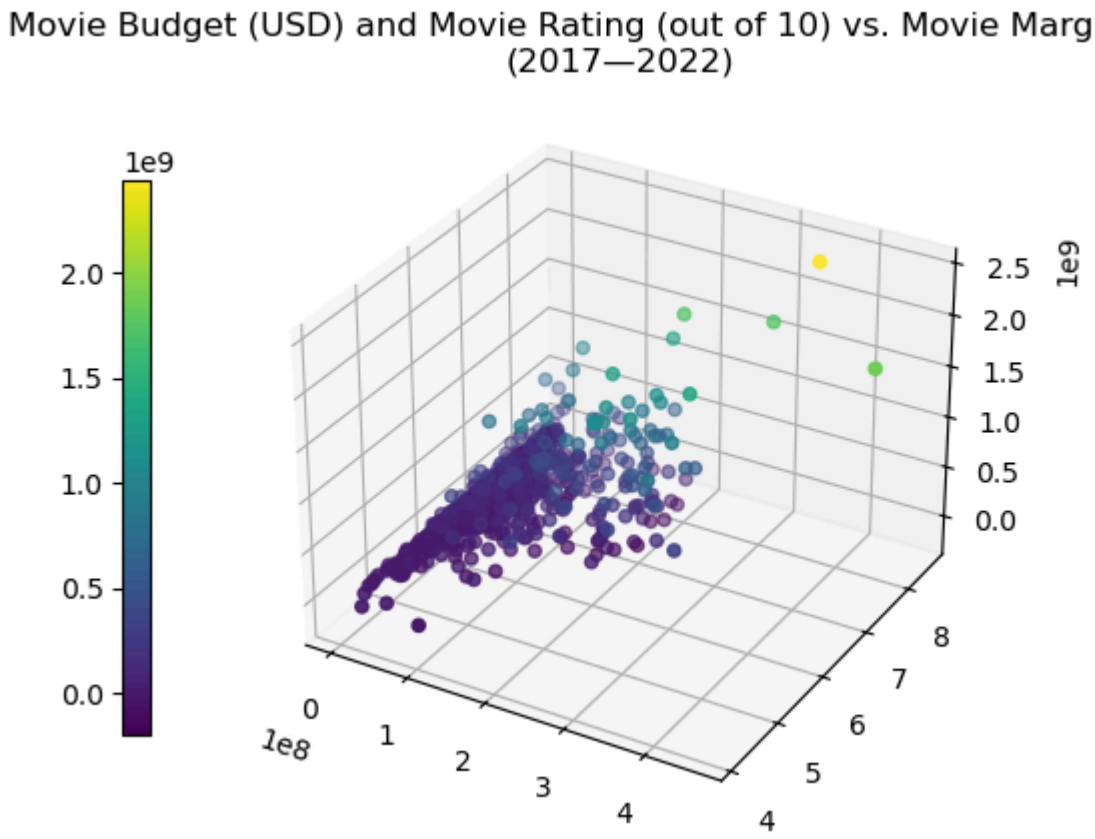
```
    {'Margin' : rm_by_margin[rm_by_margin.budget >= 100000000][['margin']].mean(axis='rows', skipna=False).iloc[0]
     'Rating' : rm_by_margin[rm_by_margin.budget >= 100000000][['rating']].mean(axis='rows', skipna=False).iloc[0]
'Any Budget' :
    {'Margin' : rm_by_margin[['margin']].mean(axis='rows', skipna=False).iloc[0],
     'Rating' : rm_by_margin[['rating']].mean(axis='rows', skipna=False).iloc[0]}}
```

Out[57]:
```
{'Budget > 200': {'Margin': np.float64(551735252.8947369),
  'Rating': np.float64(7.14371052631579)},
 'Budget > 100': {'Margin': np.float64(380096403.71900827),
  'Rating': np.float64(7.005925619834712)},
 'Any Budget': {'Margin': np.float64(85278317.76180698),
  'Rating': np.float64(6.732594455852157)}}
```

The 3D plot below helps visualize the results above between rating, budget, and margin showing that margins grow at an accelerated rate where both rating and budget increase simultaneously.

In [58]:
```
ax = plt.axes(projection='3d')
s = ax.scatter(budget, rating, margin, c=margin)
plt.title("Movie Budget (USD) and Movie Rating (out of 10) vs. Movie Margin (USD)\n(2017–2022)")
plt.colorbar(s, ax=ax, location='left', shrink=0.75)
```

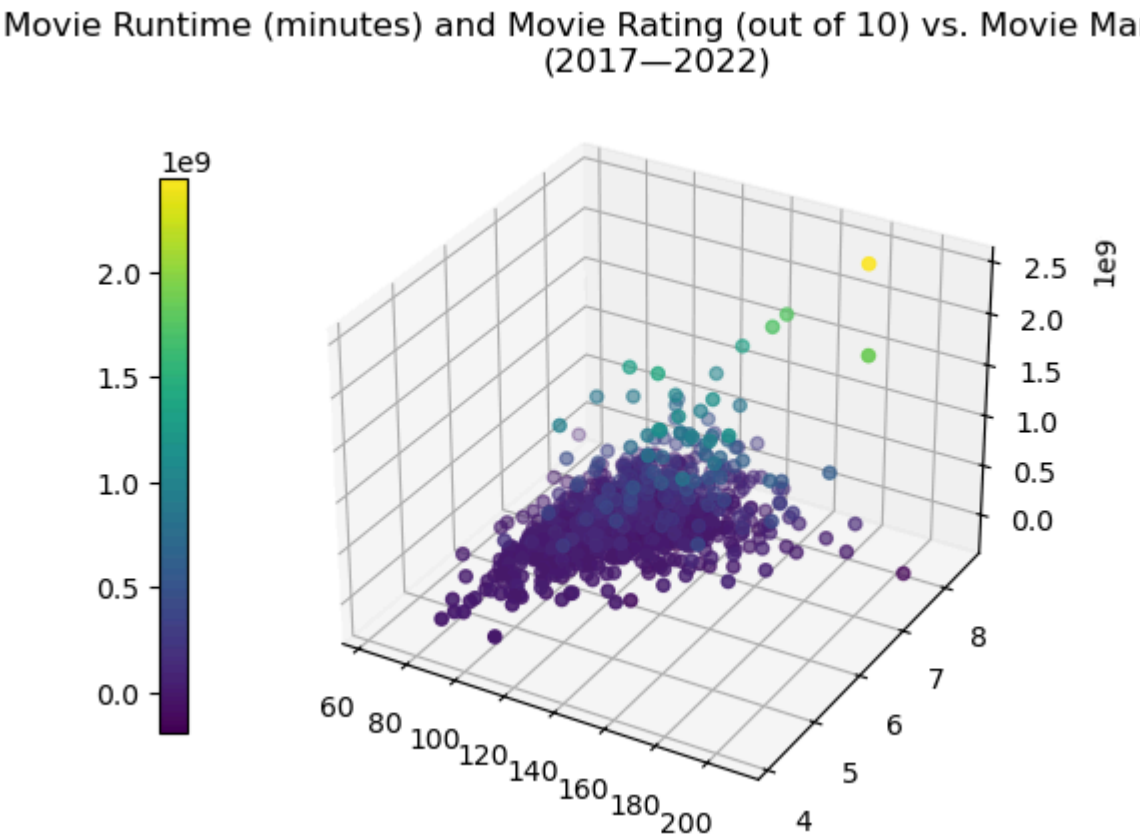
Out[58]:
```
<matplotlib.colorbar.Colorbar at 0x70b95cca4e30>
```



The 3D plot below helps visualize the results above between runtime, rating and margin, showing that this relationship is not as strongly positive as the relationship demonstrated in the prior plot.

In [59]:
```
ax = plt.axes(projection='3d')
s = ax.scatter(runtime, rating, margin, c=margin)
plt.title("Movie Runtime (minutes) and Movie Rating (out of 10) vs. Movie Margin (USD)\n(2017–2022)")
plt.colorbar(s, ax=ax, location='left', shrink=0.75)
```

Out[59]:
```
<matplotlib.colorbar.Colorbar at 0x70b984d11280>
```



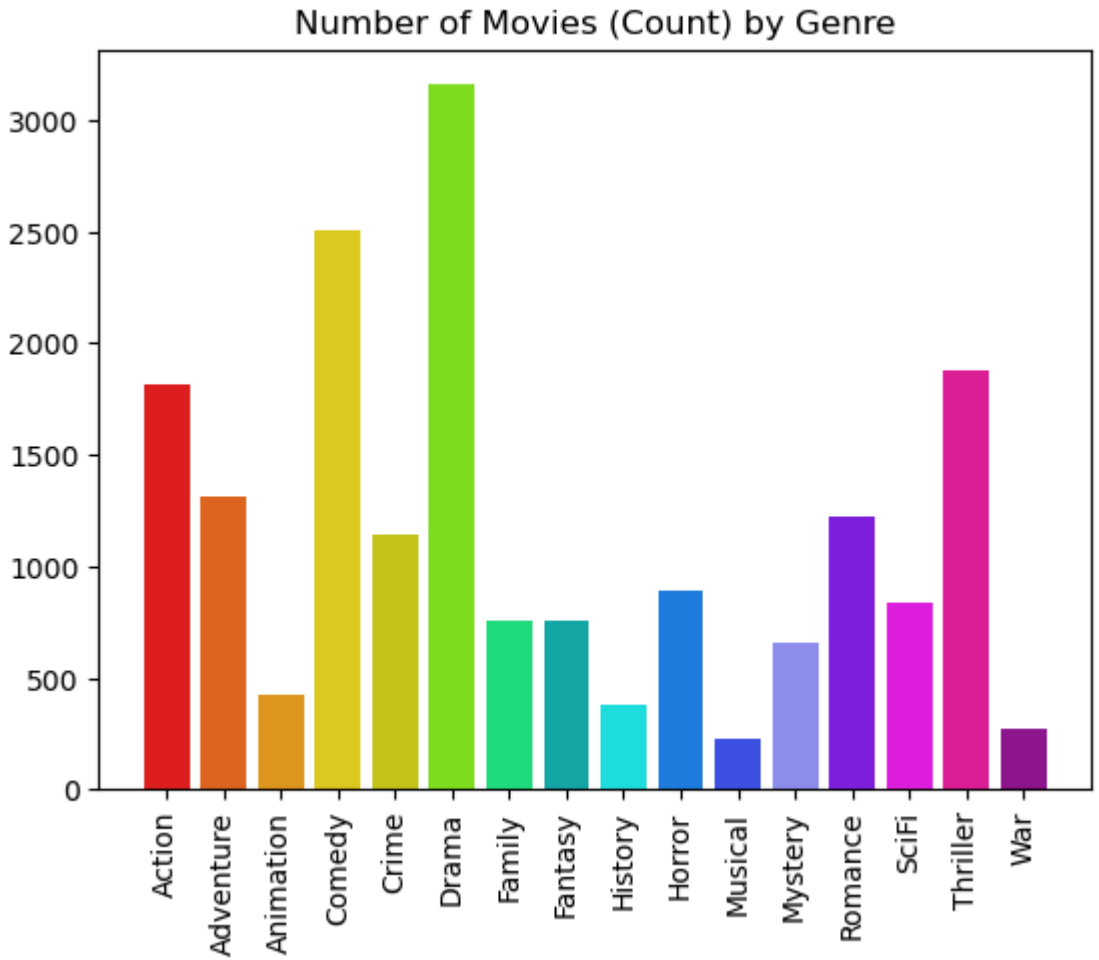
The bar and pie charts below help visualize the differences between genres, taking into consideration the results above and what seems to drive or dampen response from sales and margin.

- The movie count helps determine which of the subsequent results may be more or less substantiated by the number of observations, with confidence nonetheless depending also on the strength of the trend. The drama category takes nearly 20% of the pie.
- Runtime and ratings are consistent across genres, hence the corresponding pies are spared from inclusion.
- Cumulative results may inform on which genres viewers are willing to spend in, balancing perspectives from the results of averages.

In [60]:
```
genre_colors = ['#ff0000', '#ff6000', '#ffa000', '#ffe000', '#e0e000', '#80ff00', '#00ff80', '#00c0c0',
                '#00ffff', '#0080ff', '#2040ff', '#8080ff', '#8000ff', '#ff00ff', '#ff00a0', '#a000a0']
genre_palette = sns.color_palette(genre_colors, 16, .75)
sns.set_palette(genre_palette)
```

In [61]:
```
plt.bar(genres, genre_counts, color=genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Number of Movies (Count) by Genre")
```

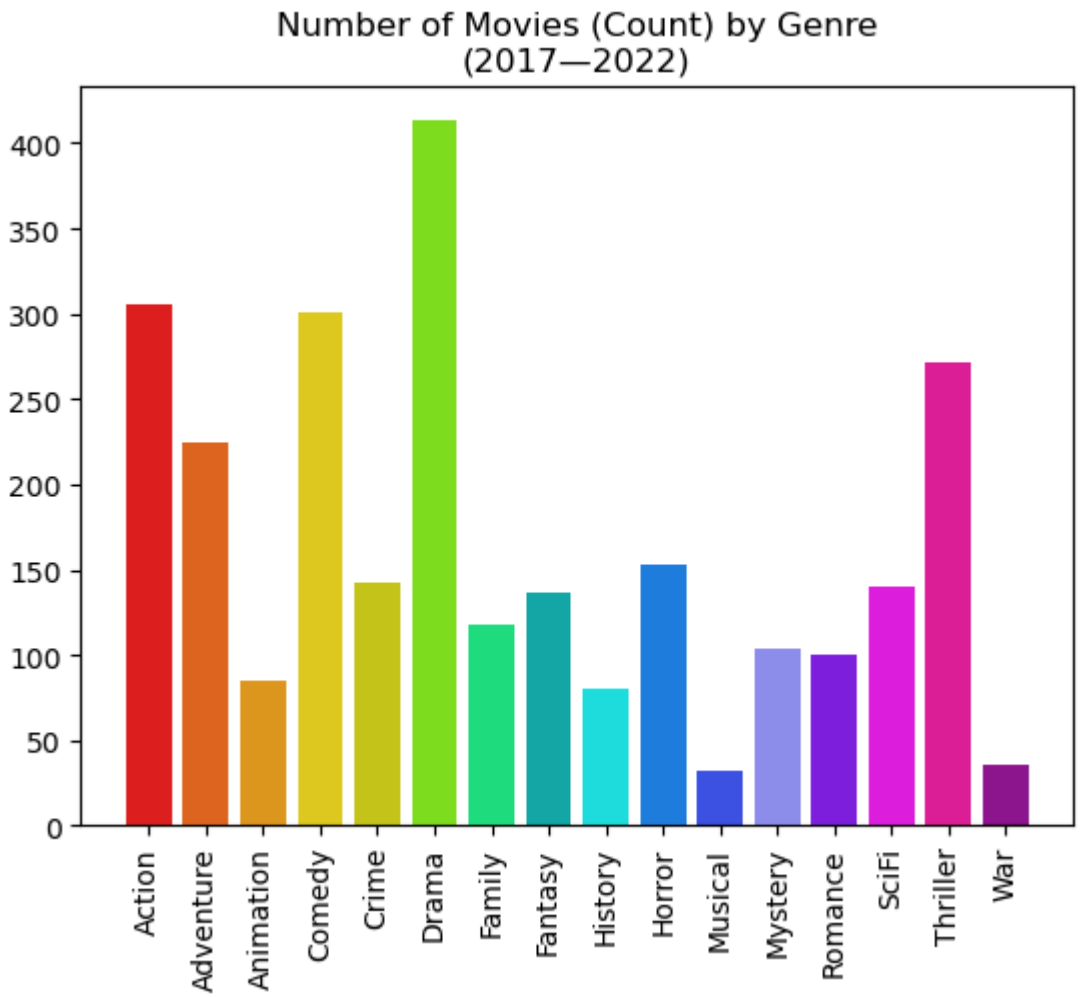
Out[61]: Text(0.5, 1.0, 'Number of Movies (Count) by Genre')



In [62]:
```
genre_counts = [movies[movies.Action == True][['id']].size, movies[movies.Adventure == True][['id']].size,
                movies[movies.Animation == True][['id']].size, movies[movies.Comedy == True][['id']].size,
                movies[movies.Crime == True][['id']].size, movies[movies.Drama == True][['id']].size,
                movies[movies.Family == True][['id']].size, movies[movies.Fantasy == True][['id']].size,
```

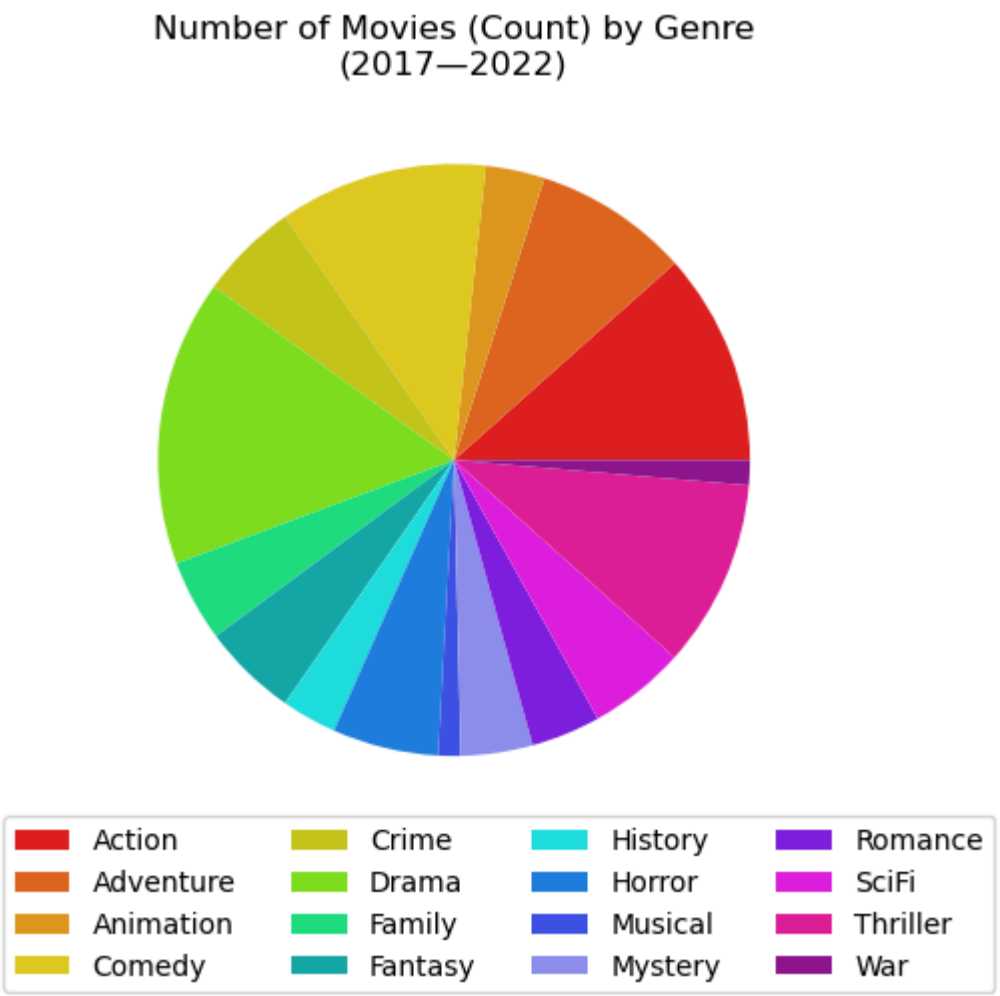
```
                  movies[movies.History == True][['id']].size, movies[movies.Horror == True][['id']].size,
                  movies[movies.Musical == True][['id']].size, movies[movies.Mystery == True][['id']].size,
                  movies[movies.Romance == True][['id']].size, movies[movies.SciFi == True][['id']].size,
                  movies[movies.Thriller == True][['id']].size, movies[movies.War == True][['id']].size]

plt.bar(genres, genre_counts, color=genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Number of Movies (Count) by Genre\n(2017–2022)")
```

Out[62]: Text(0.5, 1.0, 'Number of Movies (Count) by Genre\n(2017–2022)')



In [63]:
```
plt.pie(genre_counts, colors=genre_palette)
plt.title("Number of Movies (Count) by Genre\n(2017—2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

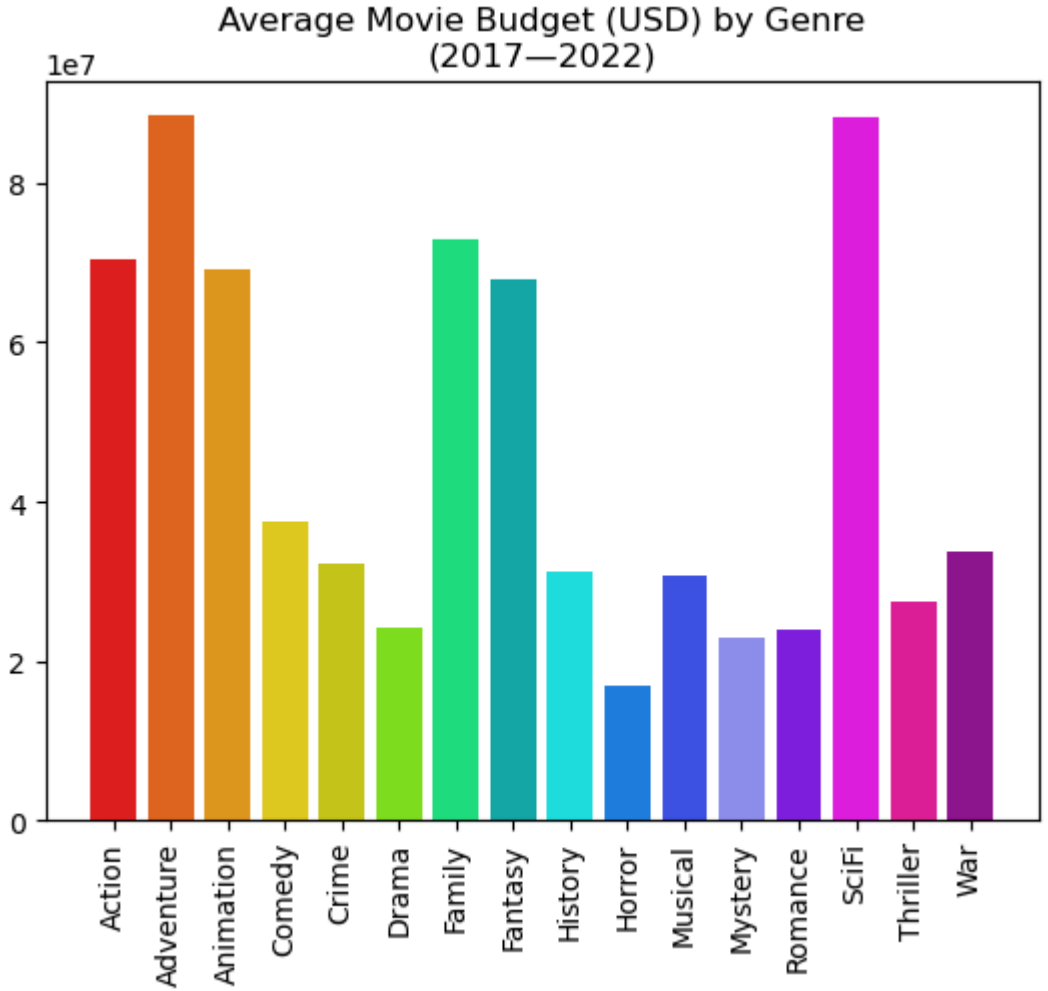
Out[63]:
```
<matplotlib.legend.Legend at 0x70b95c301f10>
```



In [64]:
```
genre_avg_budgets = [movies[movies.Action == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Adventure == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Animation == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Comedy == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Crime == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Drama == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Family == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Fantasy == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.History == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Horror == True][['budget']].mean(axis='rows', skipna=False).iloc[0],
```

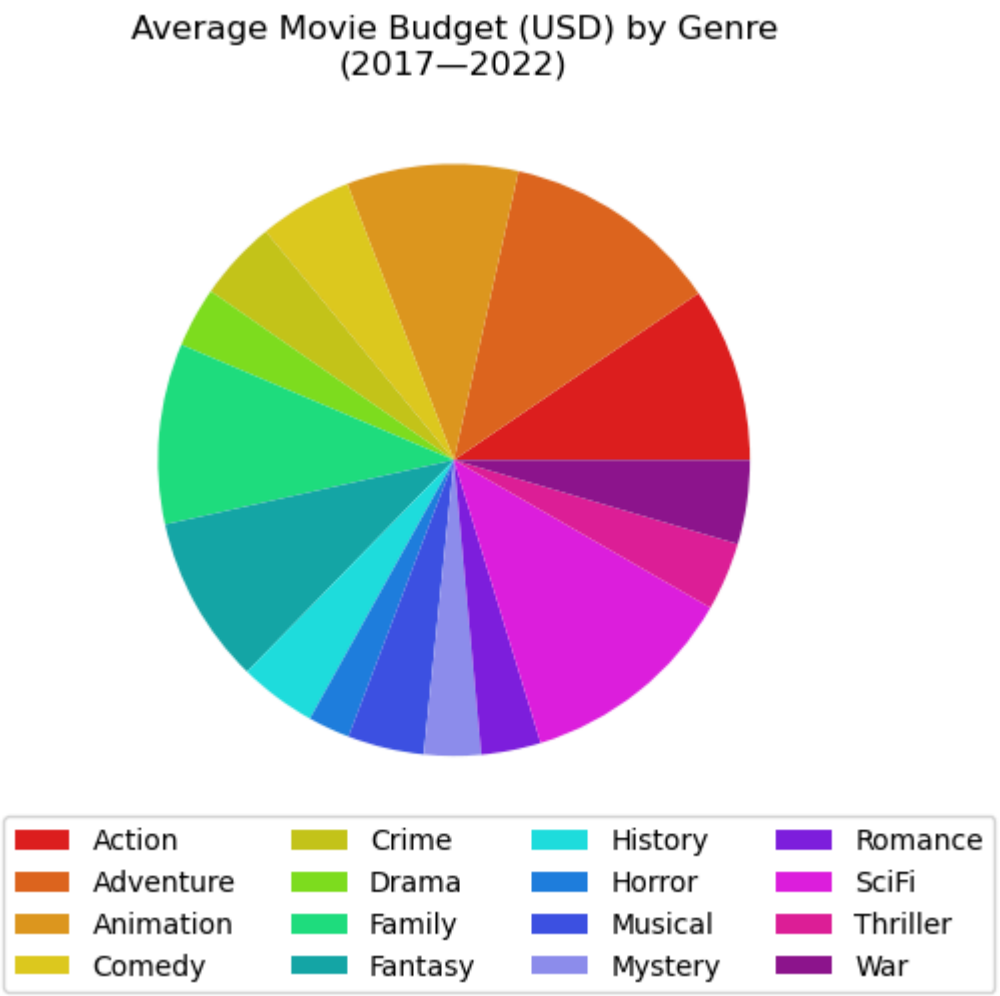
```
                      movies[movies.Musical == True]['budget']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Mystery == True]['budget']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Romance == True]['budget']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.SciFi == True]['budget']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Thriller == True]['budget']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.War == True]['budget']].mean(axis='rows', skipna=False).iloc[0]]

plt.bar(genres, genre_avg_budgets, color=genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Average Movie Budget (USD) by Genre\n(2017–2022)")
```

Out[64]: Text(0.5, 1.0, 'Average Movie Budget (USD) by Genre\n(2017–2022)')



In [65]:
```
piec = plt.pie(genre_avg_budgets, colors=genre_palette)
plt.title("Average Movie Budget (USD) by Genre\n(2017–2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

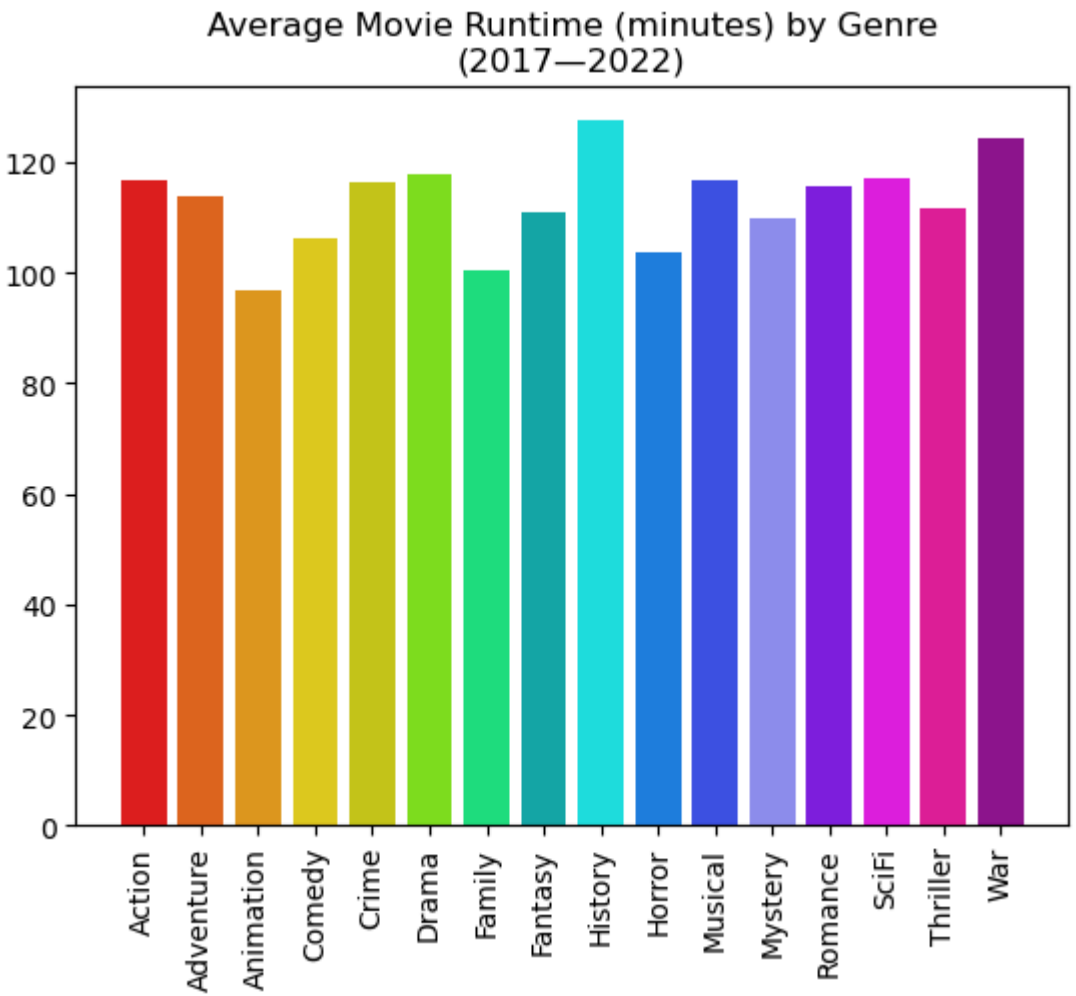
Out[65]:
```
<matplotlib.legend.Legend at 0x70b95c69bec0>
```



In [66]:
```
genre_avg_runtimes = [movies[movies.Action == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Adventure == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Animation == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Comedy == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Crime == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Drama == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Family == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Fantasy == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.History == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Horror == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
```

```
                      movies[movies.Musical == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Mystery == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Romance == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.SciFi == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Thriller == True][['runtime']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.War == True][['runtime']].mean(axis='rows', skipna=False).iloc[0]]

plt.bar(genres, genre_avg_runtimes, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Average Movie Runtime (minutes) by Genre\n(2017–2022)")
```

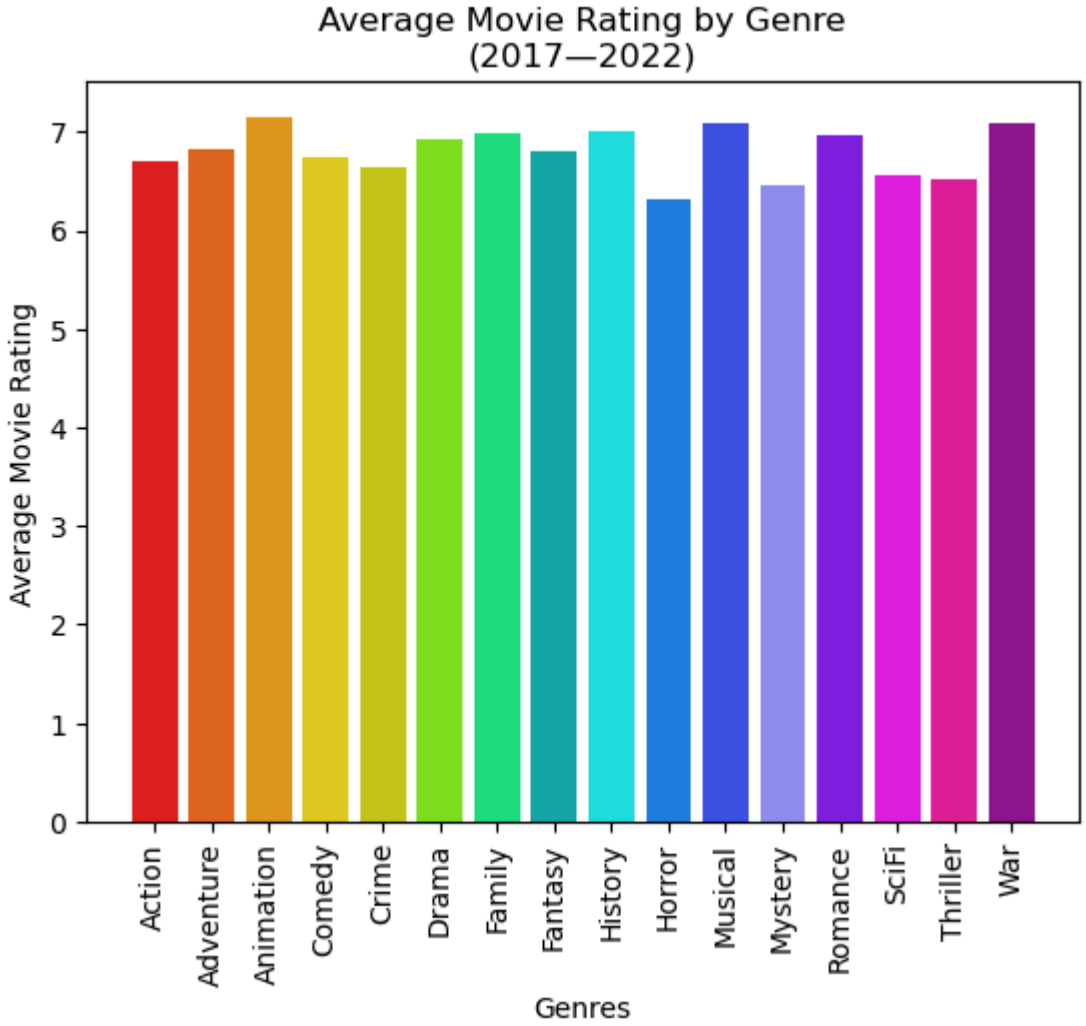
Out[66]: Text(0.5, 1.0, 'Average Movie Runtime (minutes) by Genre\n(2017–2022)')



In [67]:
```
genre_avg_ratings = [movies[movies.Action == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Adventure == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Animation == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Comedy == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Crime == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Drama == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Family == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Fantasy == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.History == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
                     movies[movies.Horror == True][['rating']].mean(axis='rows', skipna=False).iloc[0],
```

```
                      movies[movies.Musical == True]['rating']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Mystery == True]['rating']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Romance == True]['rating']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.SciFi == True]['rating']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.Thriller == True]['rating']].mean(axis='rows', skipna=False).iloc[0],
                      movies[movies.War == True]['rating']].mean(axis='rows', skipna=False).iloc[0]]

plt.bar(genres, genre_avg_ratings, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()

plt.xlabel("Genres")
plt.ylabel("Average Movie Rating")
plt.title("Average Movie Rating by Genre\n(2017–2022)")
```

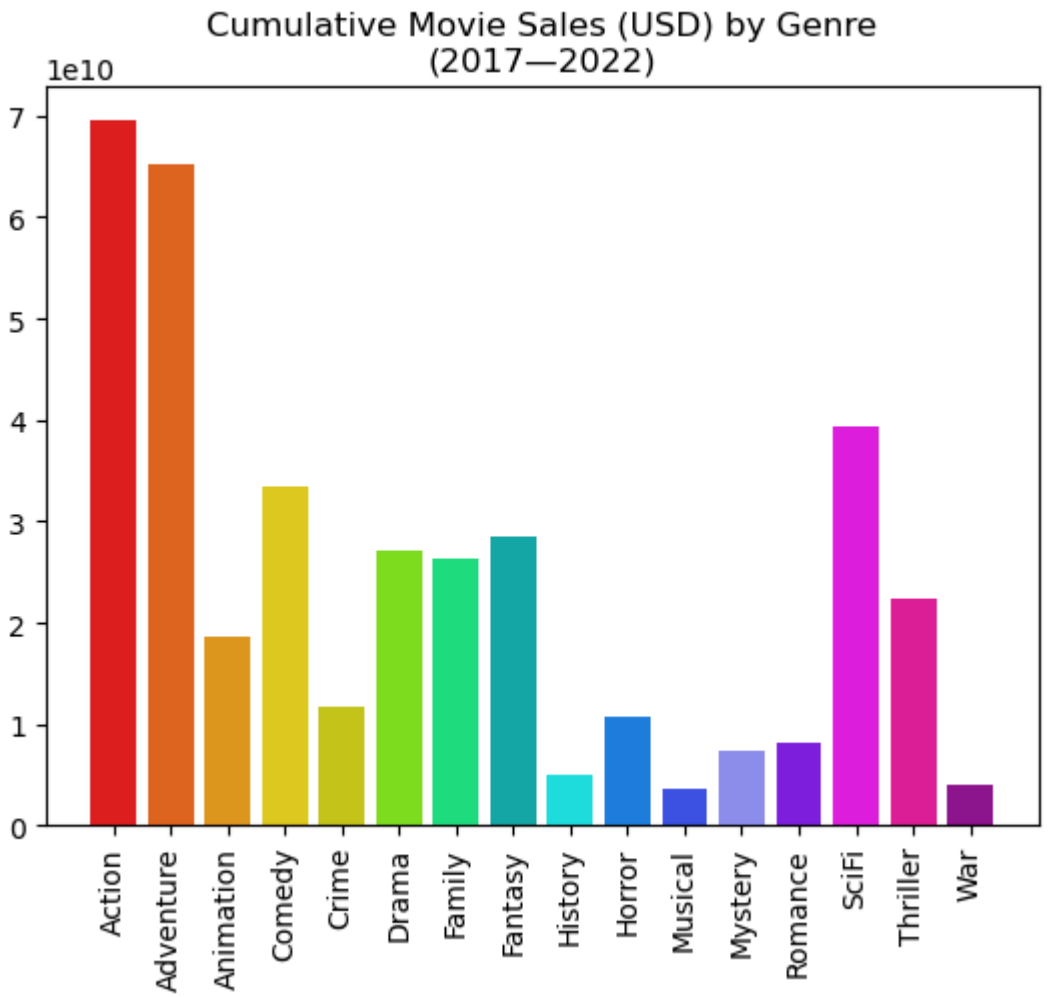
Out[67]: Text(0.5, 1.0, 'Average Movie Rating by Genre\n(2017–2022)')



In [68]:
```
genre_cum_sales =  [movies[movies.Action == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Adventure == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Animation == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Comedy == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Crime == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Drama == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Family == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Fantasy == True][['sales']].sum(axis='rows', skipna=False).iloc[0],
```

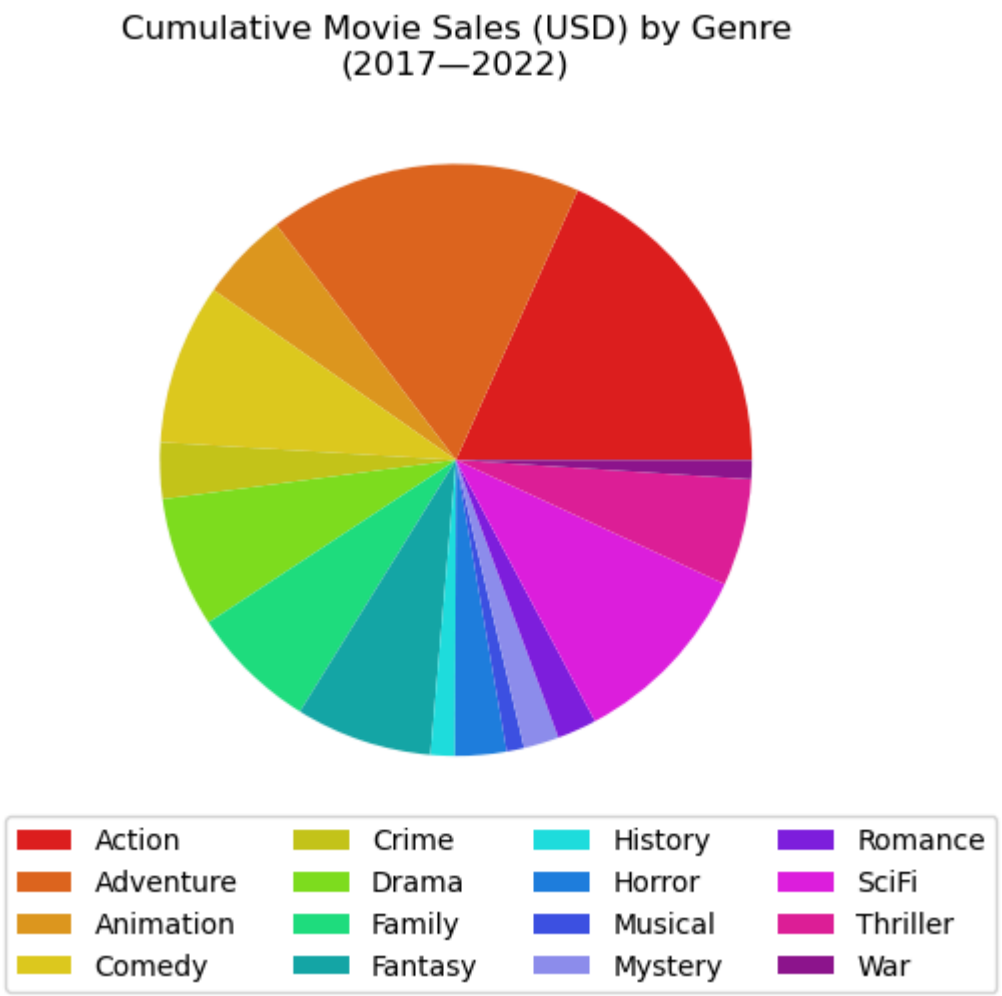
```
                    movies[movies.History == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Horror == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Musical == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Mystery == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Romance == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.SciFi == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.Thriller == True]['sales']].sum(axis='rows', skipna=False).iloc[0],
                    movies[movies.War == True]['sales']].sum(axis='rows', skipna=False).iloc[0]]

plt.bar(genres, genre_cum_sales, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Cumulative Movie Sales (USD) by Genre\n(2017–2022)")
```

Out[68]: Text(0.5, 1.0, 'Cumulative Movie Sales (USD) by Genre\n(2017–2022)')



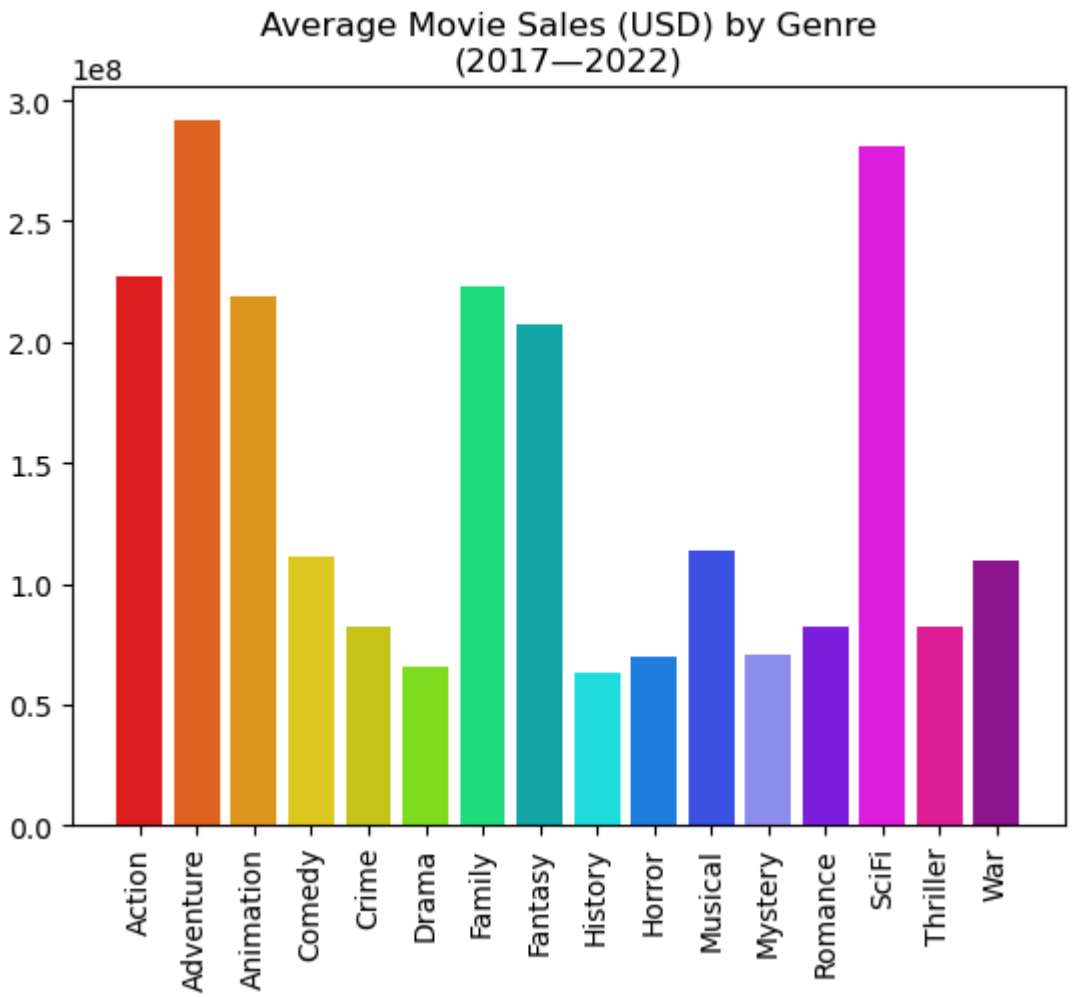
In [69]:
```
piec = plt.pie(genre_cum_sales, colors=genre_palette)
plt.title("Cumulative Movie Sales (USD) by Genre\n(2017–2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

Out[69]:
```
<matplotlib.legend.Legend at 0x70b95c4d5df0>
```



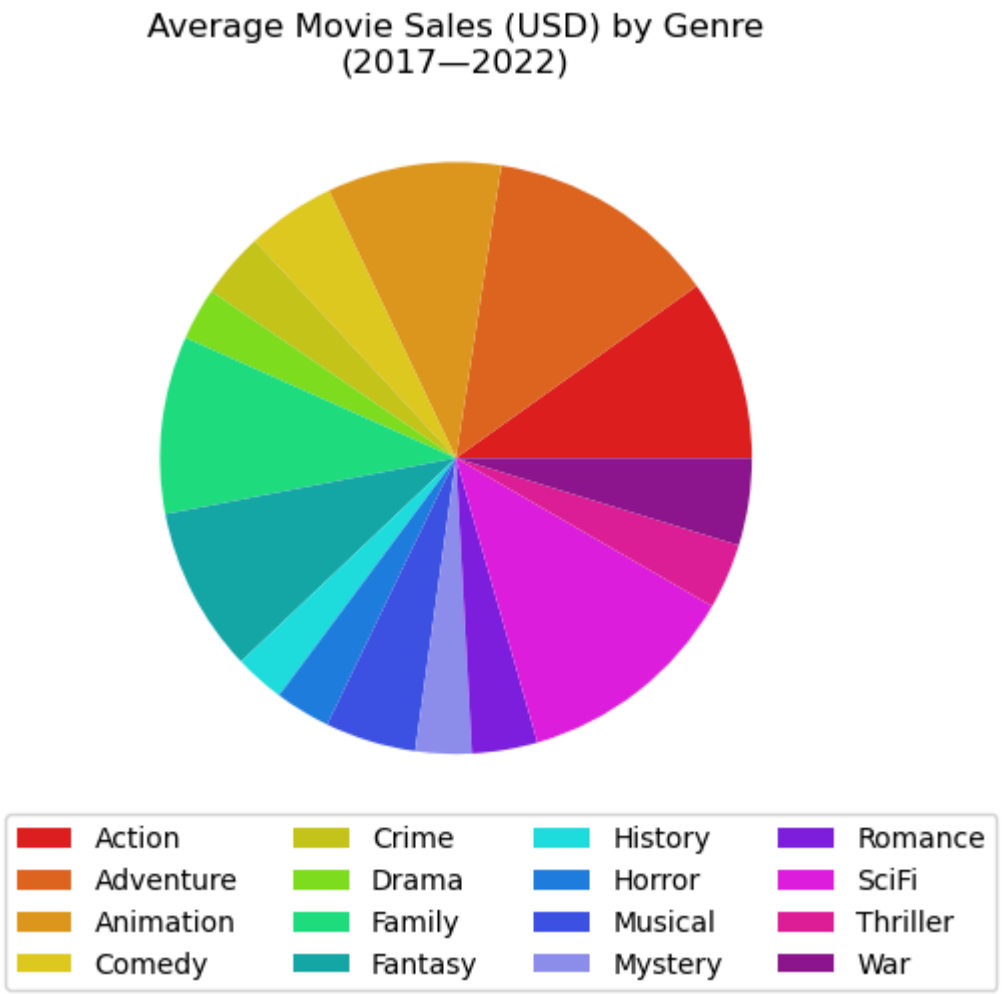
In [70]:
```
genre_avg_sales = np.divide(genre_cum_sales, genre_counts)

plt.bar(genres, genre_avg_sales, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Average Movie Sales (USD) by Genre\n(2017–2022)")
```

Out[70]:
```
Text(0.5, 1.0, 'Average Movie Sales (USD) by Genre\n(2017–2022)')
```



In [71]:
```
plt.pie(genre_avg_sales, colors=genre_palette)
plt.title("Average Movie Sales (USD) by Genre\n(2017–2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

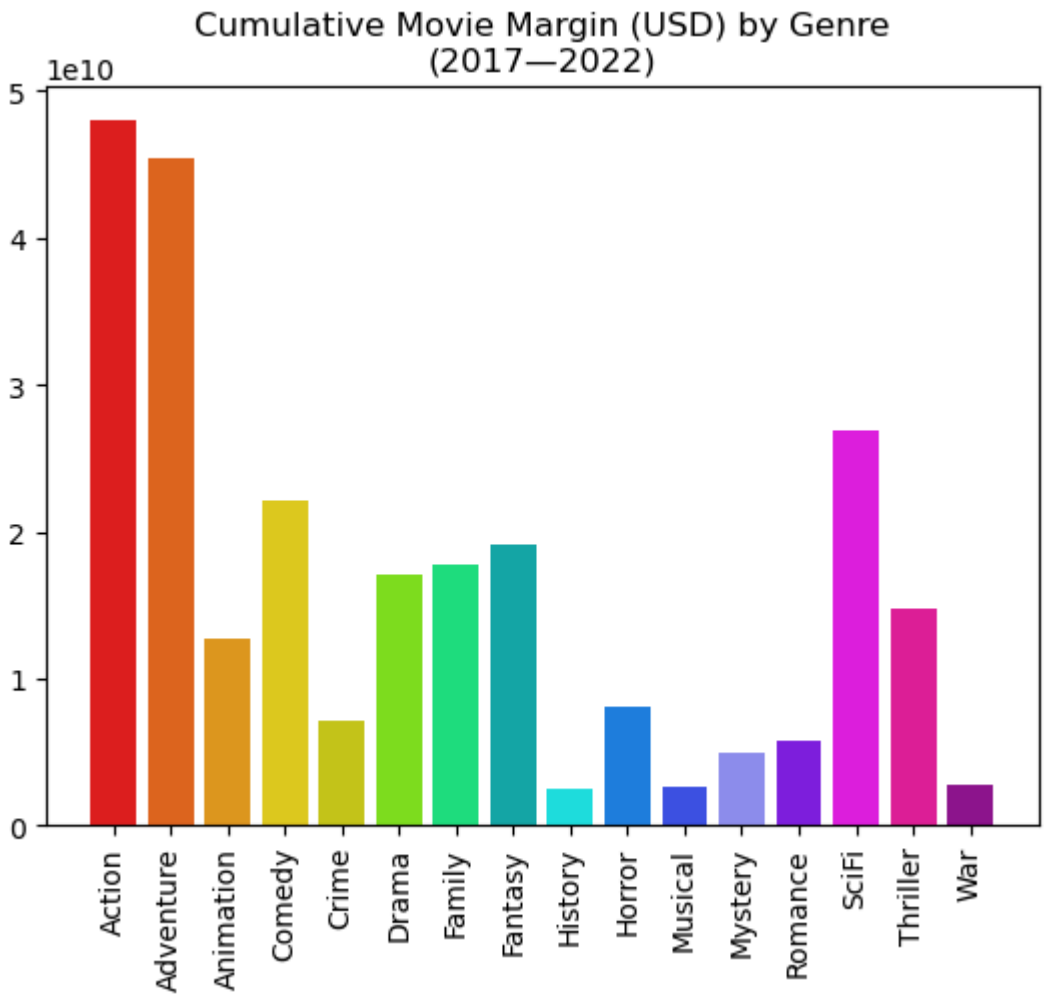
Out[71]:
```
<matplotlib.legend.Legend at 0x70b957ea9f10>
```



In [72]:
```
genre_cum_margins = [movies[movies.Action == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Adventure == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Animation == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Comedy == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Crime == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Drama == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Family == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Fantasy == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.History == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                     movies[movies.Horror == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
```

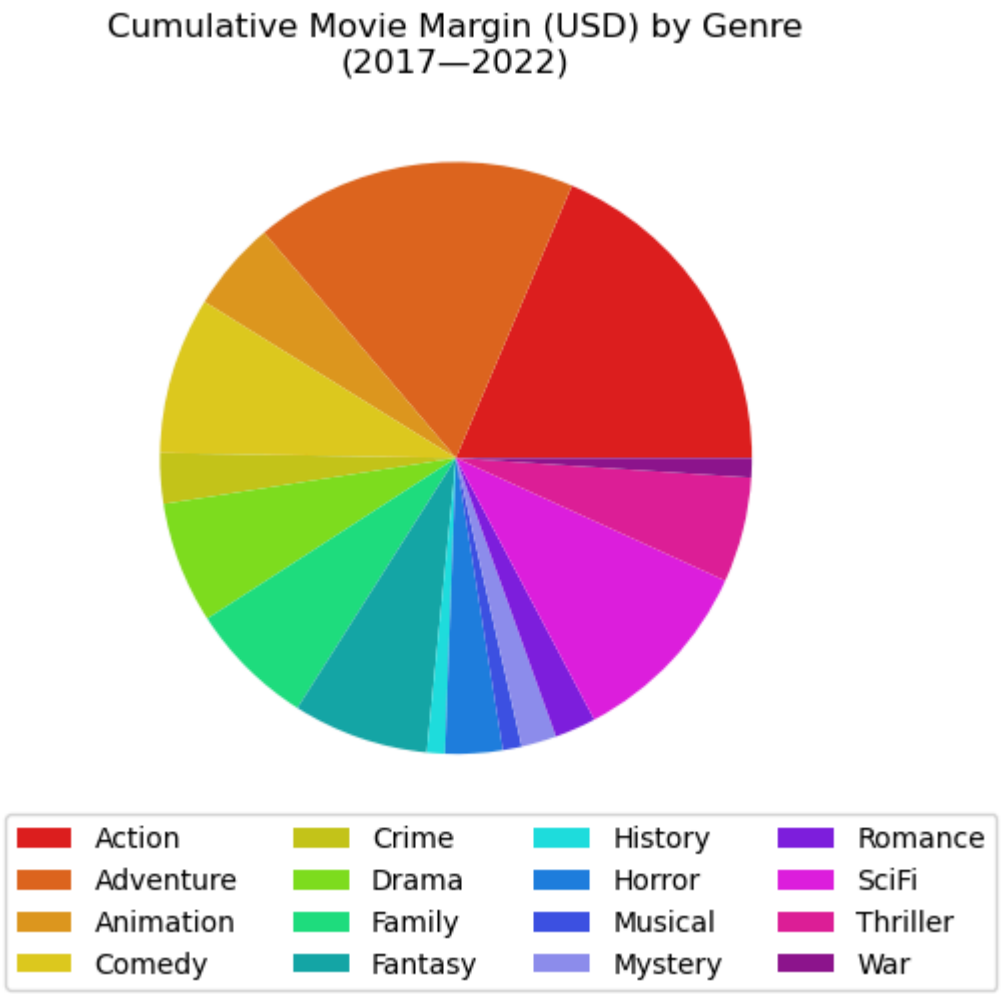
```
                      movies[movies.Musical == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                      movies[movies.Mystery == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                      movies[movies.Romance == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                      movies[movies.SciFi == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                      movies[movies.Thriller == True][['margin']].sum(axis='rows', skipna=False).iloc[0],
                      movies[movies.War == True][['margin']].sum(axis='rows', skipna=False).iloc[0]]

plt.bar(genres, genre_cum_margins, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Cumulative Movie Margin (USD) by Genre\n(2017–2022)")
```

Out[72]: Text(0.5, 1.0, 'Cumulative Movie Margin (USD) by Genre\n(2017–2022)')



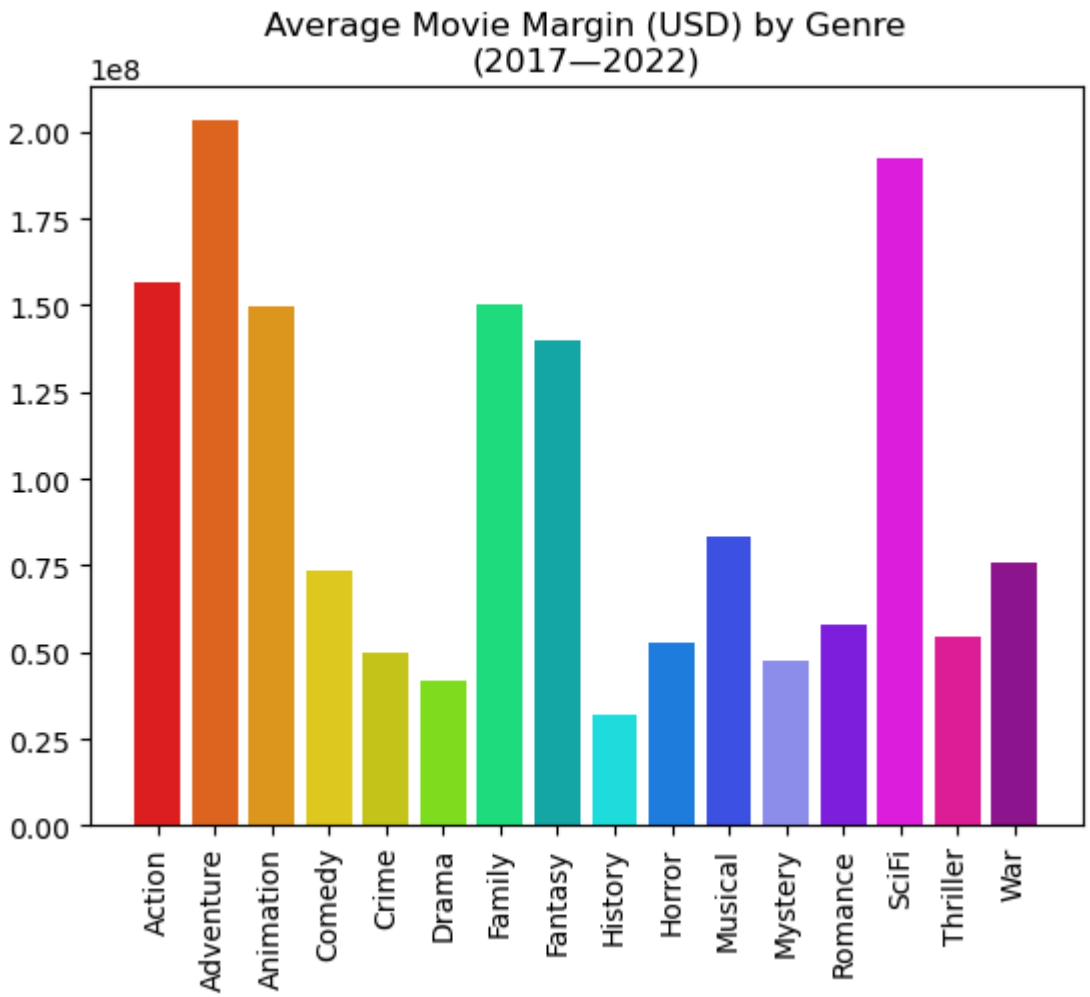
In [73]:
```
piec = plt.pie(genre_cum_margins, colors = genre_palette)
plt.title("Cumulative Movie Margin (USD) by Genre\n(2017—2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

Out[73]:
```
<matplotlib.legend.Legend at 0x70b957cabec0>
```



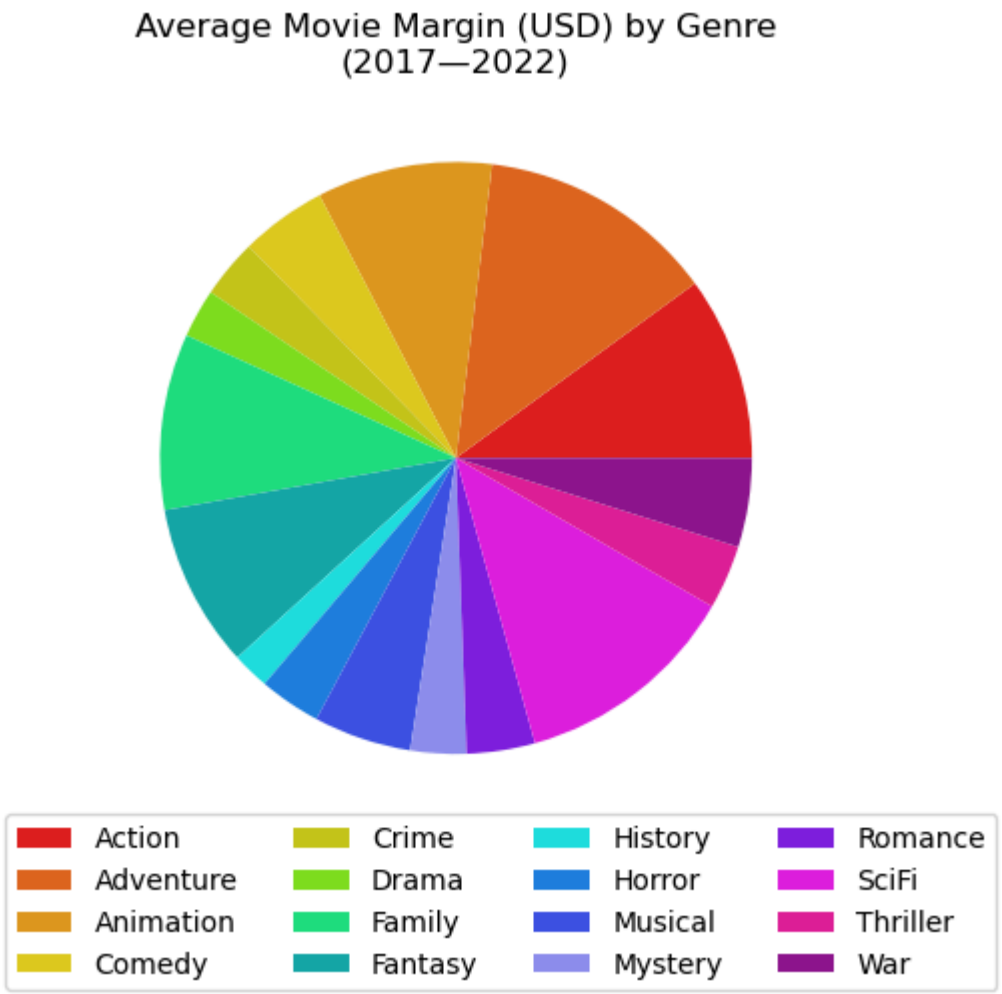
In [74]:
```
genre_avg_margins = np.divide(genre_cum_margins, genre_counts)

plt.bar(genres, genre_avg_margins, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Average Movie Margin (USD) by Genre\n(2017–2022)")
```

Out[74]:
```
Text(0.5, 1.0, 'Average Movie Margin (USD) by Genre\n(2017–2022)')
```



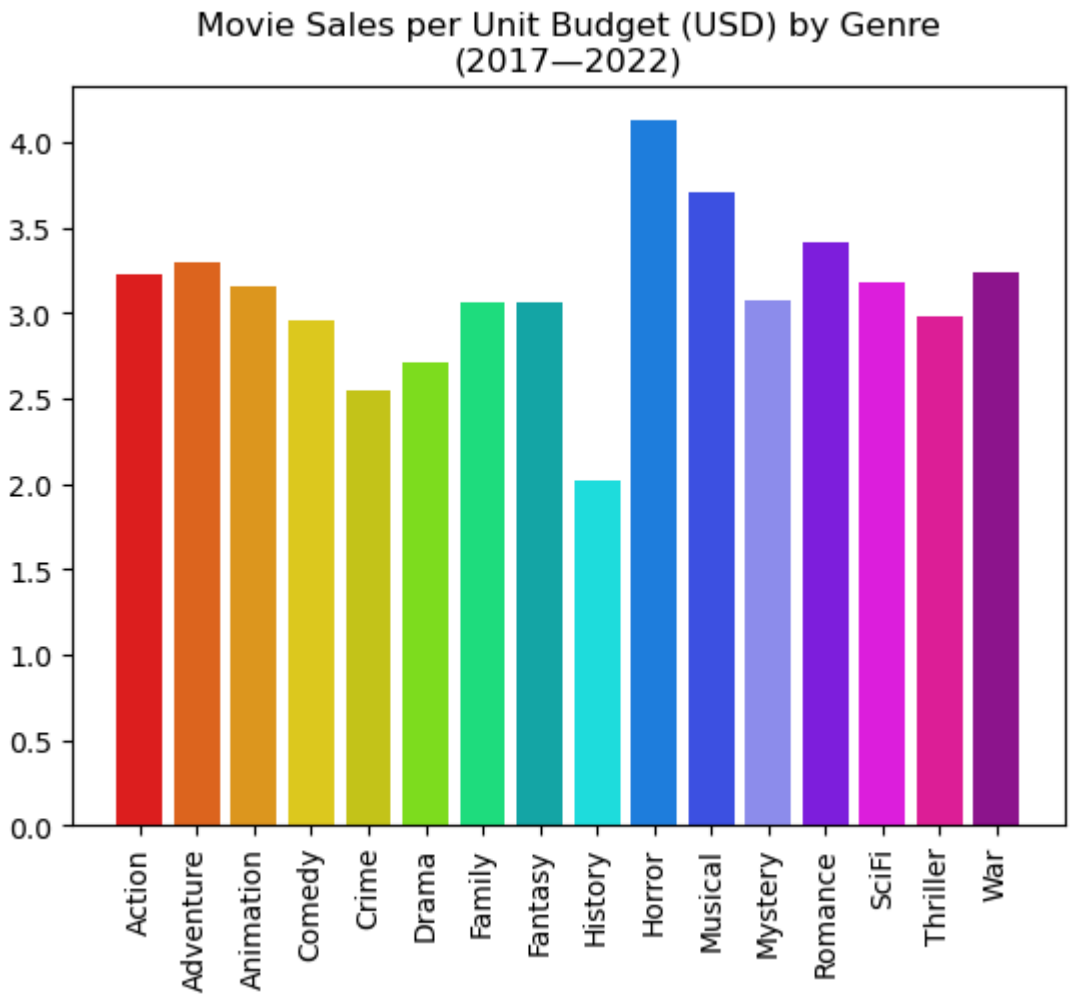
In [75]:
```
plt.pie(genre_avg_margins, colors=genre_palette)
plt.title("Average Movie Margin (USD) by Genre\n(2017–2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

Out[75]:
```
<matplotlib.legend.Legend at 0x70b95ced65a0>
```



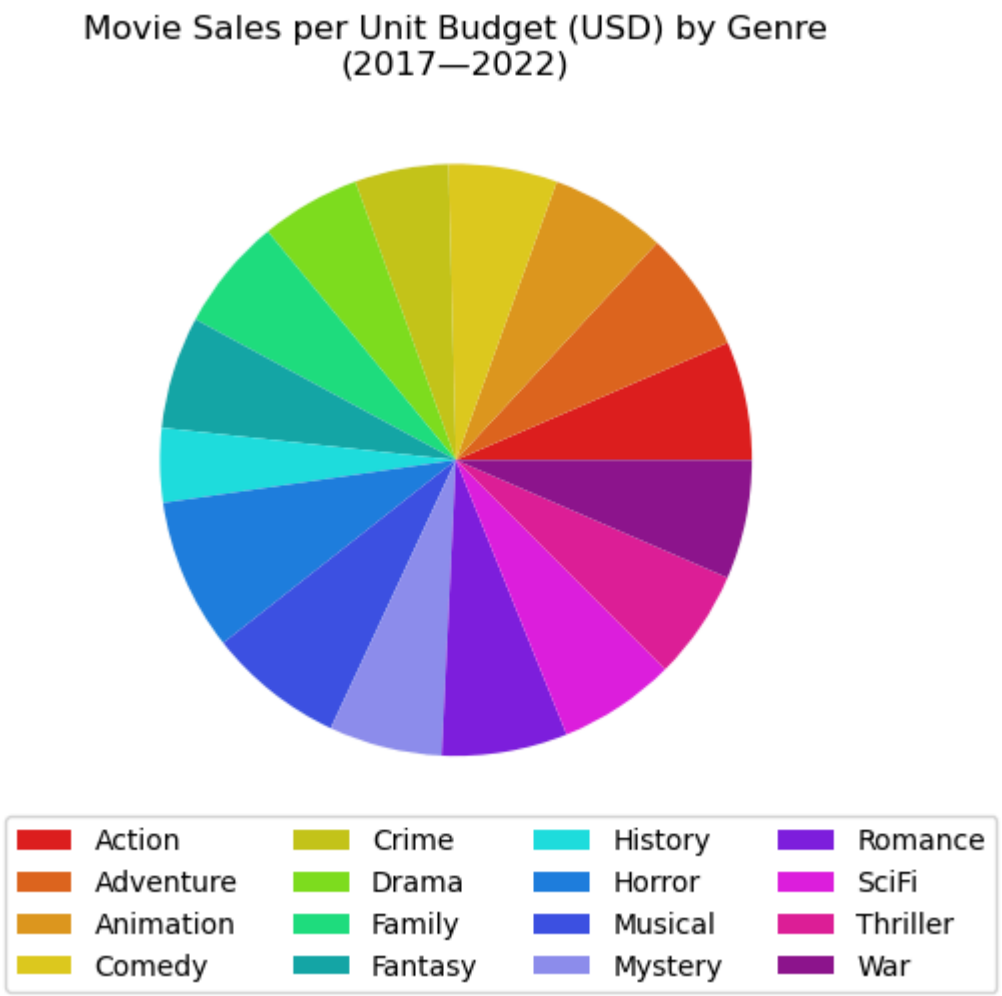
In [76]:
```
movie_sales_over_budgets = np.divide(genre_avg_sales, genre_avg_budgets)

plt.bar(genres, movie_sales_over_budgets, color = genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Movie Sales per Unit Budget (USD) by Genre\n(2017–2022)")
```

Out[76]:
```
Text(0.5, 1.0, 'Movie Sales per Unit Budget (USD) by Genre\n(2017–2022)')
```



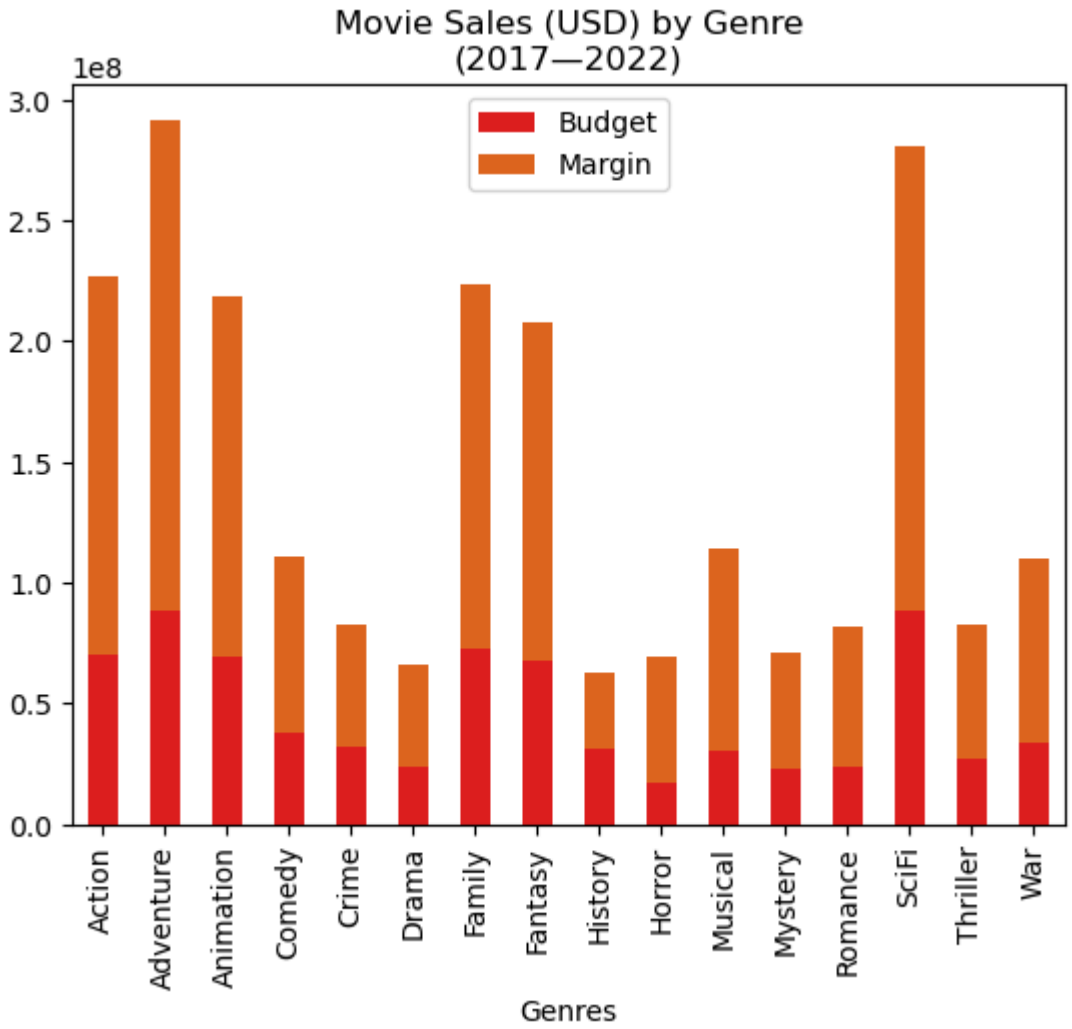
In [77]:
```
plt.pie(movie_sales_over_budgets, colors=genre_palette)
plt.title("Movie Sales per Unit Budget (USD) by Genre\n(2017—2022)")
plt.legend(loc='right', labels=genres, bbox_to_anchor=(1.25,-0.1), ncols=4)
```

Out[77]:
```
<matplotlib.legend.Legend at 0x70b95c41e870>
```



In [78]:
```
genre_sales_stack = {'Genres': genres, 'Budget' : genre_avg_budgets, 'Margin' : genre_avg_margins}
stack_frame = pd.DataFrame(genre_sales_stack)
stack_frame.set_index('Genres').plot(kind='bar', stacked=True, color=genre_palette)
plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], rotation='vertical')
plt.yticks()
plt.title("Movie Sales (USD) by Genre\n(2017–2022)")
```

Out[78]:
```
Text(0.5, 1.0, 'Movie Sales (USD) by Genre\n(2017–2022)')
```



The barplots above show that the most budget-intensive genres (like action, adventure, animation, family, fantasy, and sci-fi) produce the highest sales and margins, whereas choice of genre does not seem to affect either rating or runtime. Given the positive sales/margin results for the aforementioned six genres, surprising is that four out of six genres (animation, family, fantasy and sci-fi) account for less than 15% of the number total movies, which lags behind five other genres (comedy, crime, horror, and thriller). The count of drama movies far outweighs the counts of action and adventure movies despite comparatively poor return on investment as shown from the stacked barplot above. The distribution of movie counts across genres between the time-restricted and time-agnostic dataset is similar.

Assumptions

From the analysis above, the following trends were observed:

- Raising budget increases sales as well as margin (although the expenditures that directly cause sales/margin increase is unclear).
- Raising budget increases votes, which increases rating, which increases sales as well as margin.
- Raising budget increases runtime, which, as runtime approaches 150 minutes, increases ratings, which increases sales as well as margin.
- Sales increase as either budget, rating, runtime, or popularity increase. Where more than one of these determinants increase, the increase in sales tends to be more dramatic.
- Rating does not respond to budget increases, necessitating other measures in order to drive rating.
- Popularity is considered more of a substitute rather than a driver of rating or votes because popularity already takes rating and votes into account of its own computation.
- The correlation observed in each time-restricted regression plot was typically similar to the correlation observed in the corresponding time-agnostic regression plot, with exception of wider confidence intervals in the time-restricted plot.
- The most profitable genres (action, adventure, animation, family, fantasy, and sci-fi) are not produced as frequently as some other less profitable categories (especially drama).

In relation to these observations, the following considerations are applied:

- The budget expenditures that directly cause increases to votes and runtime are more ostensible than for sales/margin increase.
- Limitations in the dataset do not describe how increasing spending in areas such as recruiting and hiring talent, as well as advertising, drive sales.
- The global pandemic decreased responsiveness of sales to all determinants, and the result of the pandemic may be causing word of mouth to travel faster as society becomes more digitized.

Modeling

Various machine learning approaches (Naive Bayes, K-Nearest Neighbors, and Multilinear Regression) are herein utilized in order to learn possible patterns within the dataset useful for predicting future outcomes.

The Naive Bayes method evaluates a set of features (predictor variables) in order to predict whether a movie is `profitable` (either `True` or `False`). The `id`, `title` and `date` columns are not considered to be predictor variables.

In [79]:
```
profitable
```

Out[79]:
```
0         True
1         True
2         True
3         True
4         True
         ...
18119     True
18127    False
18129     True
18132    False
18156     True
Name: profitable, Length: 7022, dtype: bool
```

In [80]:
```
profitable.value_counts(normalize=True)
```

Out[80]:
```
profitable
True     0.702934
False    0.297066
Name: proportion, dtype: float64
```

In [81]:
```
movies = movies.iloc[:,3:]
movies.head()
```

Out[81]:

	runtime	votes	rating	popularity	budget	sales	margin	profitable	Action	Adventure	...	Family	Fantasy	History
6	149	27713	8.255	154.340	300000000	2052415039	1752415039	True	True	True	...	False	False	False
15	181	23857	8.263	91.756	356000000	2800000000	2444000000	True	True	True	...	False	False	False
18	122	23425	8.168	54.522	55000000	1074458282	1019458282	True	False	False	...	False	False	False
27	135	21053	7.390	43.665	200000000	1349926083	1149926083	True	True	True	...	False	False	False
32	133	20507	7.345	65.880	175000000	880166924	705166924	True	True	True	...	False	False	False

5 rows × 24 columns

The Naive Bayes model is, first, trained, and then, tested, on disparate subsets from the dataset. The model is trained by being fitted to the train data, then, tested by predicting on the test data.

In [82]:
```
np.random.seed(123)

data_randomized = movies.sample(frac=1)

train_size = round(len(data_randomized) * 0.8)

train_set = data_randomized[:train_size].reset_index(drop=True)
test_set = data_randomized[train_size:].reset_index(drop=True)
```

The train dataset and test dataset should be structurally similar.

In [83]:
```
print(train_set.shape)
print(test_set.shape)
```

```
(779, 24)
(195, 24)
```

In [84]:
```
train_set.head()
```

Out[84]:

	runtime	votes	rating	popularity	budget	sales	margin	profitable	Action	Adventure	...	Family	Fantasy	History	Ho
0	127	5199	7.418	48.284	200000000	220889446	20889446	True	True	True	...	False	True	False	F
1	106	190	6.479	9.850	16000000	20690779	4690779	True	False	False	...	False	False	False	F
2	139	127	6.134	7.172	9661157	3562793	-6098364	False	False	False	...	False	False	False	F
3	105	214	5.544	9.422	1686858	1891907	205049	True	True	False	...	False	True	False	F
4	95	294	7.881	7.981	9850000	1756552	-8093448	False	False	True	...	False	False	False	F

5 rows × 24 columns

In [85]:
```
test_set.head()
```

Out[85]:

	runtime	votes	rating	popularity	budget	sales	margin	profitable	Action	Adventure	...	Family	Fantasy	History	Horr
0	86	193	6.142	15.705	5000000	3500000	-1500000	False	False	False	...	False	False	False	Fal
1	112	1276	7.000	11.457	20000000	8654322	-11345678	False	False	False	...	False	False	False	Fal
2	110	886	4.129	19.589	95000000	73515024	-21484976	False	False	False	...	False	True	False	Fal
3	117	1312	6.196	20.880	20000000	69766483	49766483	True	False	False	...	False	False	False	Fal
4	108	1420	6.911	32.218	23000000	23076711	76711	True	True	False	...	False	False	False	Fal

5 rows × 24 columns

In [86]:
```
train_set['profitable'].value_counts(normalize=True)
```

Out[86]:
```
profitable
True     0.631579
False    0.368421
Name: proportion, dtype: float64
```

In [87]:
```
test_set['profitable'].value_counts(normalize=True)
```

Out[87]:
```
profitable
True     0.594872
False    0.405128
Name: proportion, dtype: float64
```

The features (independent variables) and labels (dependent variable) of the train dataset are assigned to separate variables for convenience working with data processing API.

In [88]:
```
train_x = train_set.iloc[:,8:]
train_y = train_set['profitable']
print((train_set.columns.size, train_x.columns.size))
```

```
(24, 16)
```

In [89]:
```
train_y
```

Out[89]:
```
0       True
1       True
2      False
3       True
4      False
       ...
774     True
775     True
776    False
777    False
778    False
Name: profitable, Length: 779, dtype: bool
```

In [90]:
```
colnames = train_x.columns
```

In [91]:
```
train_x.head()
```

Out[91]:

	Action	Adventure	Animation	Comedy	Crime	Drama	Family	Fantasy	History	Horror	Musical	Mystery	Romance	SciFi	Thriller
0	True	True	False	False	False	False	False	True	False	False	False	False	False	False	False
1	False	False	False	False	False	True	False	False	False	False	False	False	True	False	False
2	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False
3	True	False	False	False	True	False	False	True	False	False	False	False	False	False	False
4	False	True	True	False	False	False	False	False	False	False	False	True	False	False	False

In [92]:
```
train_y.head()
```

Out[92]:
```
0     True
1     True
2    False
3     True
4    False
Name: profitable, dtype: bool
```

Labels are encoded as `0` or `1` consistent with a Bernoulli RV.

In [93]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

In [94]:
```
le = LabelEncoder

train_brnli = le.fit_transform(le, train_y)

train_brnli[:5]
```

Out[94]:
```
array([1, 1, 0, 1, 0])
```

Numerical (continuous) features are converted to categorical (discrete) features by the process of binning.

In [95]:
```
enc = OrdinalEncoder()

train_x = enc.fit_transform(train_x)

train_x = pd.DataFrame(train_x, columns=colnames)

train_x.head()
```

Out[95]:

	Action	Adventure	Animation	Comedy	Crime	Drama	Family	Fantasy	History	Horror	Musical	Mystery	Romance	SciFi	Thriller
0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

The model is fit to the train data comprising the encoded features and encoded label.

In [96]:
```
model = CategoricalNB()
model.fit(train_x, train_brnli)
```

Out[96]:

▾ CategoricalNB

```
CategoricalNB()
```

The fitted model is evaluated on its accuracy of predicting train labels given a set of train features. The process of fitting the model minimizes error or variance on predicting train data which increases bias from predicting other possible (out-of-sample) data; a dichotomy which is called, bias-variance tradeoff. As a result, evaluating model accuracy on the basis of train features produces a more biased and therefore less reliable representation of model accuracy in predicting on out-of-sample data than evaluating model accuracy on the basis of test features.

In [97]:
```
train_yhat = model.predict(train_x)
```

In [98]:
```
pd.crosstab(train_yhat, train_y)
```

Out[98]:

profitable	False	True
row_0		
0	107	102
1	180	390

In [99]:
```
confusion_matrix(train_yhat, train_brnli)
```

Out[99]:
```
array([[107, 102],
       [180, 390]])
```

In [100…
```
accuracy_score(train_yhat, train_brnli)
```

Out[100…
```
0.637997432605905
```

The processes, of assigning features and label to separate variables and converting features and label from numerical to binned categorical values, are repeated for test data.

In [101…
```
test_x = test_set.iloc[:,8:]
test_y = test_set['profitable']
print((test_set.columns.size, test_x.columns.size))
```

```
(24, 16)
```

In [102…
```
colnames = test_x.columns
print((test_set.columns.size, test_x.columns.size))

test_x.head()
test_y.head()
```

```
(24, 16)
```

Out[102…
```
0    False
1    False
2    False
3     True
4     True
Name: profitable, dtype: bool
```

In [103…
```
le = LabelEncoder
test_brnli = le.fit_transform(le, test_y)

enc = OrdinalEncoder()
test_x = enc.fit_transform(test_x)

test_x = pd.DataFrame(test_x, columns=colnames)
test_x.head()
```

Out[103…

	Action	Adventure	Animation	Comedy	Crime	Drama	Family	Fantasy	History	Horror	Musical	Mystery	Romance	SciFi	Thriller
0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The fitted model is evaluated on its accuracy of predicting train labels given a set of test features

In [104…
```
test_yhat = model.predict(test_x)
test_yhat
```

Out[104…
```
array([1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1])
```

In [105…
```
conf_m = confusion_matrix(test_yhat, test_brnli)
conf_m
```

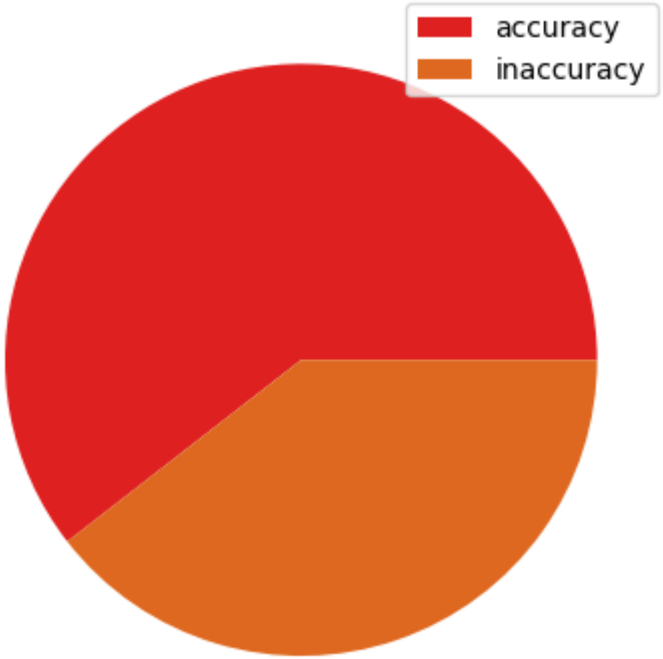
Out[105…
```
array([[27, 25],
       [52, 91]])
```

In [106…
```
score = accuracy_score(test_yhat, test_brnli)
score
```

Out[106…
```
0.6051282051282051
```

In [107…
```
plt.pie([score, 1-score])
plt.legend(labels=['accuracy', 'inaccuracy'])
```

Out[107…
```
<matplotlib.legend.Legend at 0x70b9579db440>
```

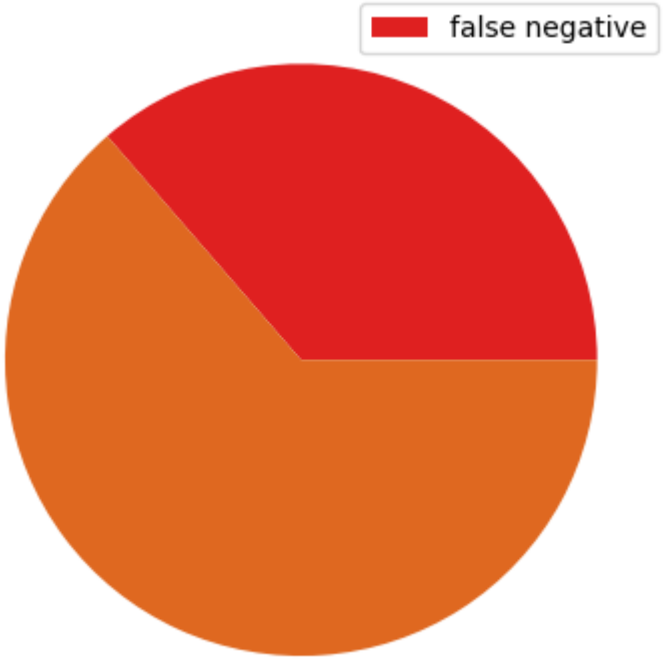


The probability of false negative rate is:

In [108…
```
prob_t2 = conf_m[1][0]/sum(conf_m[1])
```

In [109…
```
plt.pie([prob_t2, 1-prob_t2])
plt.legend(labels=['false negative'])
```

Out[109…
```
<matplotlib.legend.Legend at 0x70b9579b8920>
```



The K-Nearest Neighbors (kNN) method predicts an outcome `margin` based on a single predictor, which is `budget`. The two columns of interest are individually assigned to an independent or dependent variable, respectively, A seed is specified for reproducability of the results from the random choice generator used to generate random indeces for splitting the combined columns into train and test subsets. Vector `k_vec` is created for hyperparameter `k` in the kNN model.

In [110…
```
from sklearn.neighbors import KNeighborsRegressor as knr
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

In [111…
```
n = movies['profitable'].size
x = movies.budget
y = movies.margin

xy = pd.DataFrame({'budget':x,'margin':y})
```

```
xy.shape

np.random.seed
train_i = np.random.choice(n, int(n*.6), replace=False)

train_set = []
test_set = []

train_set = xy.iloc[train_i, :]
test_set = xy.iloc[-train_i, :]

k_vec = range(1,int(n*.5))
k_n = len(k_vec)
```

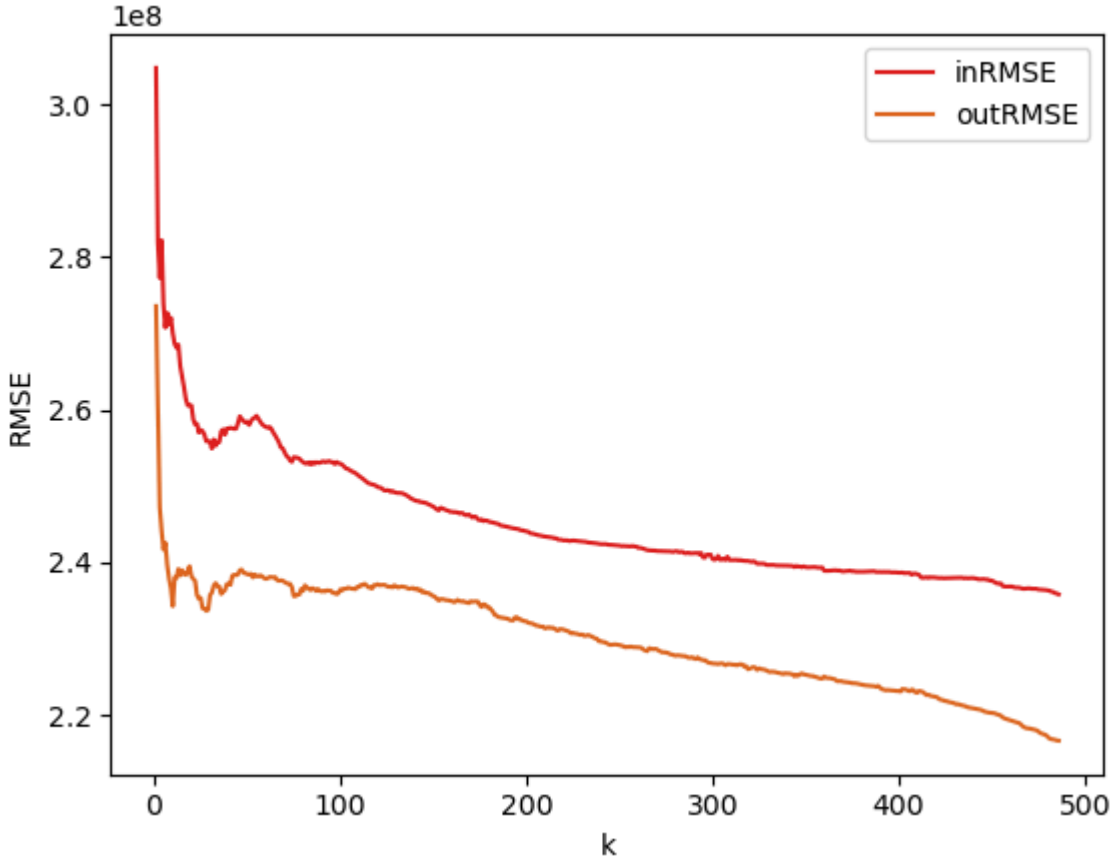
For each `k` in `k_vec`, the model is fit to the train data, then the root mean squared error (RMSE) is performed on out-of-sample and in-sample predictions.

In [112…
```
outRMSE = np.repeat(0, k_n)
inRMSE = np.repeat(0, k_n)

def rmse(yhat, y):
    return np.sqrt(np.mean((yhat - y)**2))

scaler = StandardScaler()

scaled_train_x = scaler.fit_transform(train_set.budget.to_frame())
scaled_test_x = scaler.transform(test_set.budget.to_frame())

in_rmse = [0] * k_n
out_rmse = [0] * k_n

for k_el in k_vec:
    model = knr(n_neighbors=k_el)
    in_rmse[k_el - 1] = rmse(model.fit(scaled_train_x, train_set.margin).predict(scaled_test_x), train_set.margin.t
    out_rmse[k_el - 1] = rmse(model.fit(scaled_train_x, test_set.margin).predict(scaled_test_x), test_set.margin.to
```

In [113…
```
# credit: https://www.geeksforgeeks.org/plot-multiple-lines-in-matplotlib/

plt.plot(k_vec,in_rmse,label='inRMSE')
```

```
plt.plot(k_vec,out_rmse,label='outRMSE')
plt.xlabel('k')
plt.ylabel('RMSE')
plt.legend()
```

Out[113… `<matplotlib.legend.Legend at 0x70b9575ade20>`



The `k` with the least out-of-sample RMSE is determined.

In [114…
```
k_best = np.argmin(out_rmse)
k_best
```

Out[114… `np.int64(485)`

Cross-validation evaluates the performance of competing models fit to train data and predicting on test data by randomly dividing the original dataset into `k` folds, choosing one fold as test data, training the model on the remainking `k - 1` groups, predicting on observations in the test set, and repeating the procedure for `k` folds in the data set. After RMSE is computed for each fold, average RMSE is computed for all folds.

In [115…

```
# credit: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

n_folds = 5
k = 5

model.set_params(n_neighbors=k)
kfold = KFold(n_splits=n_folds, shuffle=True, random_state=123)

scores = []
predictions = []

for train_i, val_i in kfold.split(x.to_numpy()):

    train_x = x.iloc[train_i]
    train_y = y.iloc[train_i]
    val_x = x.iloc[val_i]
    val_y = y.iloc[val_i]

    model.fit(train_x.to_frame(), train_y.to_frame())
    pred_y = model.predict(val_x.to_frame())

    score = rmse(pred_y[0], val_y)
    scores.append(score)

    predictions.extend(pred_y)

np.mean(scores)
```

Out[115… `np.float64(480473639.7467799)`

The multilinear regression approach is applied for multiple features (runtime, votes, rating, polularity, budget) with respect to a single label, `margin`. The test and train data are split, the model is fit to the train data using multilinear regression, and the model predicts labels from

the test data. RMSE is computed, and the coefficients and intercept of the equation for the regression lines are computed. This process is repeated for each multilinear regression approach (lr, ridge, lasso and elastinet).

In [116…
```
# credit: https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/

from sklearn.model_selection import train_test_split as tts
from sklearn.linear_model import LinearRegression as lr
from sklearn import metrics

x = movies.iloc[:,:5]
y = movies.margin
x
```

Out[116…

	runtime	votes	rating	popularity	budget
6	149	27713	8.255	154.340	300000000
15	181	23857	8.263	91.756	356000000
18	122	23425	8.168	54.522	55000000
27	135	21053	7.390	43.665	200000000
32	133	20507	7.345	65.880	175000000
...	...	...	...	...	...
17896	86	102	6.931	7.196	170000
17933	87	102	6.377	11.176	3000000
18047	88	101	5.297	18.488	80000000
18119	112	100	7.055	29.969	5900000
18132	113	100	7.030	10.190	20000000

974 rows × 5 columns

In [117…
```
train_x = []
test_x = []
train_y = []
```

```
test_y = []
train_x, test_x, train_y, test_y = tts(x,y,test_size=0.2,random_state=42)
```

In [118…
```
model = lr()
model.fit(train_x, train_y)
pred_y = model.predict(test_x)
```

In [119…
```
out_rmse = []
out_rmse.append(rmse(pred_y, test_y))
out_rmse
```

Out[119…
```
[np.float64(156033403.11981857)]
```

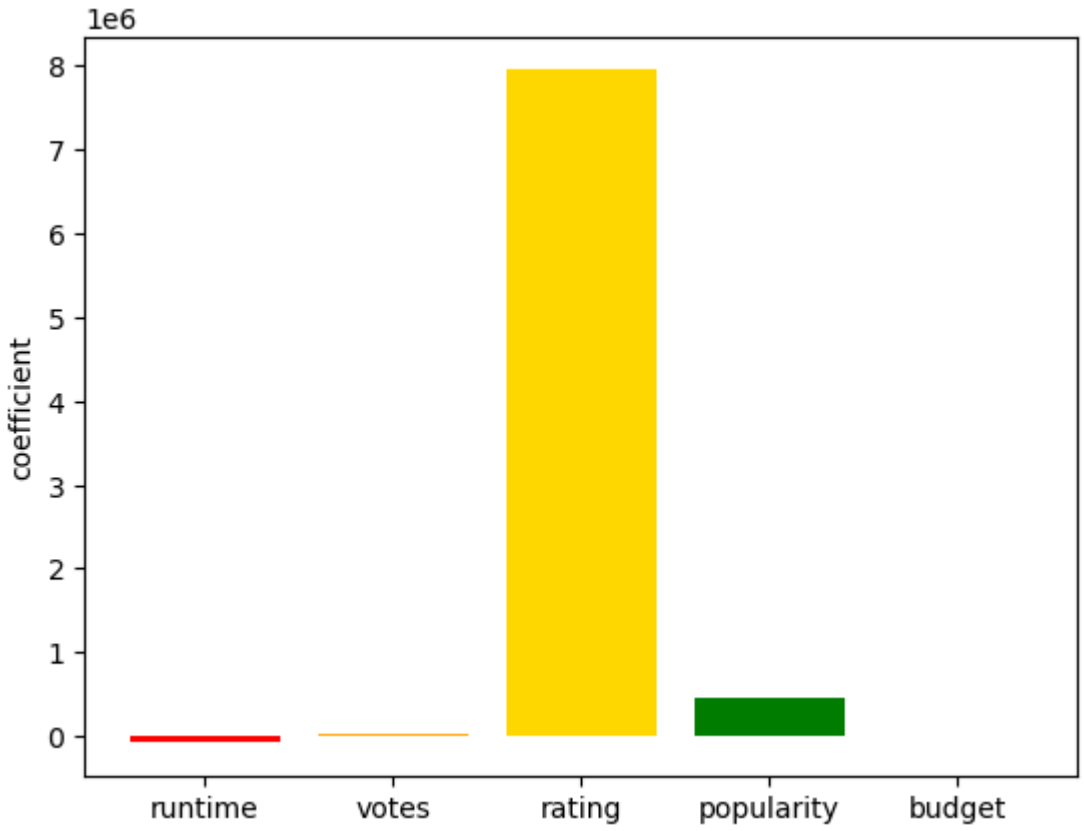
In [120…
```
m = model.coef_
b = model.intercept_
m
```

Out[120…
```
array([-7.90797556e+04,  3.40809947e+04,  7.94556184e+06,  4.62336389e+05,
        1.02996799e+00])
```

In [121…
```
feature_colors = ['red', 'orange', 'gold', 'green', 'blue']
```

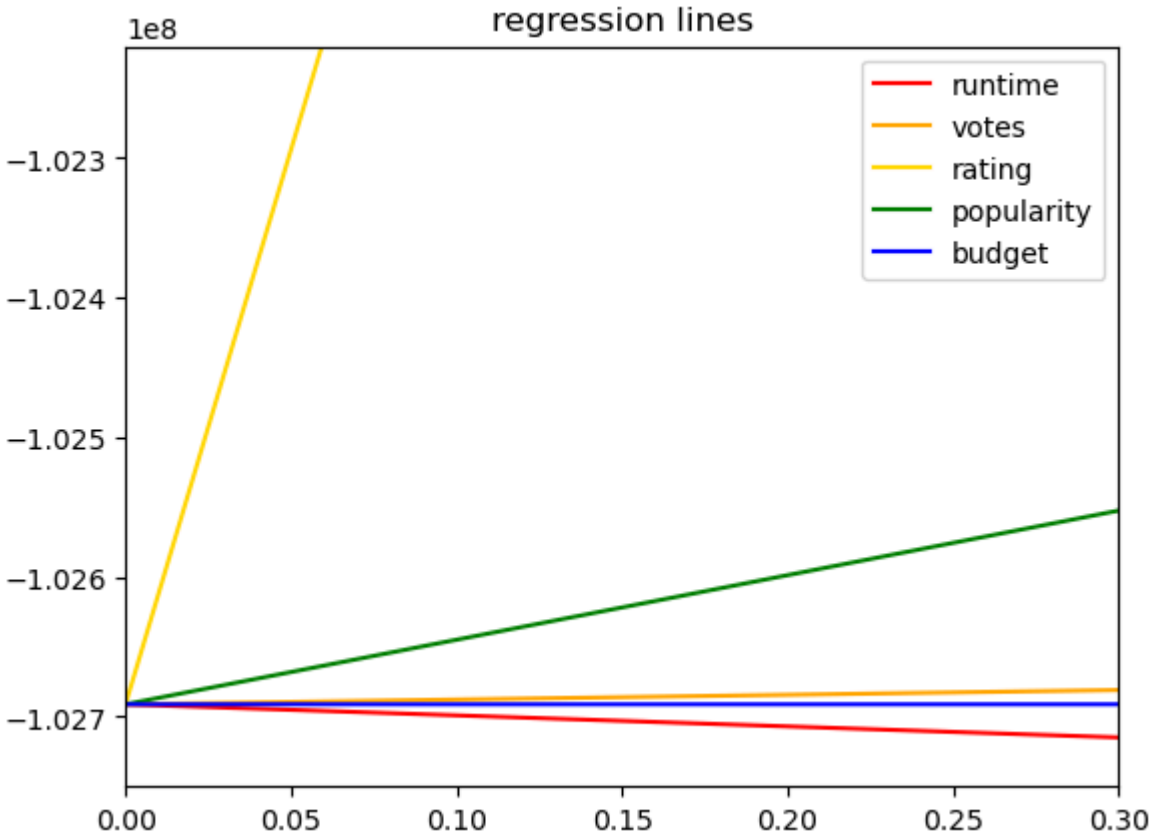
In [122…
```
range_m = range(m.size)
plt.bar(range_m, m, color=feature_colors)
plt.ylabel('coefficient')
plt.xticks(range_m, x.columns)
```

Out[122…
```
([<matplotlib.axis.XTick at 0x70b956452690>,
  <matplotlib.axis.XTick at 0x70b956442870>,
  <matplotlib.axis.XTick at 0x70b95642f3e0>,
  <matplotlib.axis.XTick at 0x70b956453200>,
  <matplotlib.axis.XTick at 0x70b954ac0a40>],
 [Text(0, 0, 'runtime'),
  Text(1, 0, 'votes'),
  Text(2, 0, 'rating'),
  Text(3, 0, 'popularity'),
  Text(4, 0, 'budget')])
```



In [123…
```
for i in range(m.size): plt.axline(xy1=(0,b), slope=m[i],color=feature_colors[i])
plt.xlim(0, 0.3)
plt.ylim(-1.0275*10**8, -1.0222*10**8)
plt.title('regression lines')
plt.legend(labels=x.columns)
```

Out[123…
```
<matplotlib.legend.Legend at 0x70b954af2fc0>
```



In [124…
```
# credit: https://scikit-learn.org/stable/api/sklearn.linear_model.html

from sklearn.linear_model import Lasso

model = Lasso(alpha=0.1)
model.fit(train_x, train_y)
pred_y = model.predict(test_x)
```

In [125…
```
out_rmse.append(rmse(pred_y, test_y))
out_rmse
```

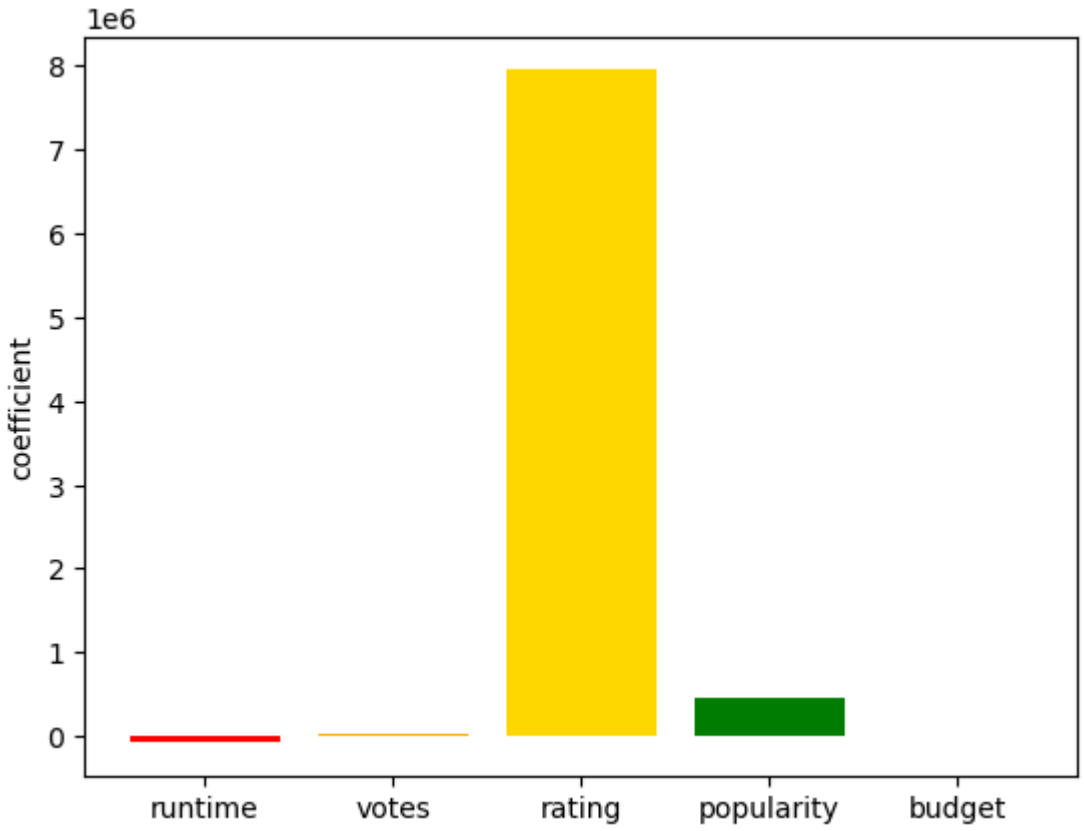
Out[125… `[np.float64(156033403.11981857), np.float64(156033403.10274816)]`

In [126…
```
m = model.coef_
b = model.intercept_
m
```

Out[126…
```
array([-7.90797524e+04,  3.40809947e+04,  7.94556161e+06,  4.62336389e+05,
        1.02996799e+00])
```

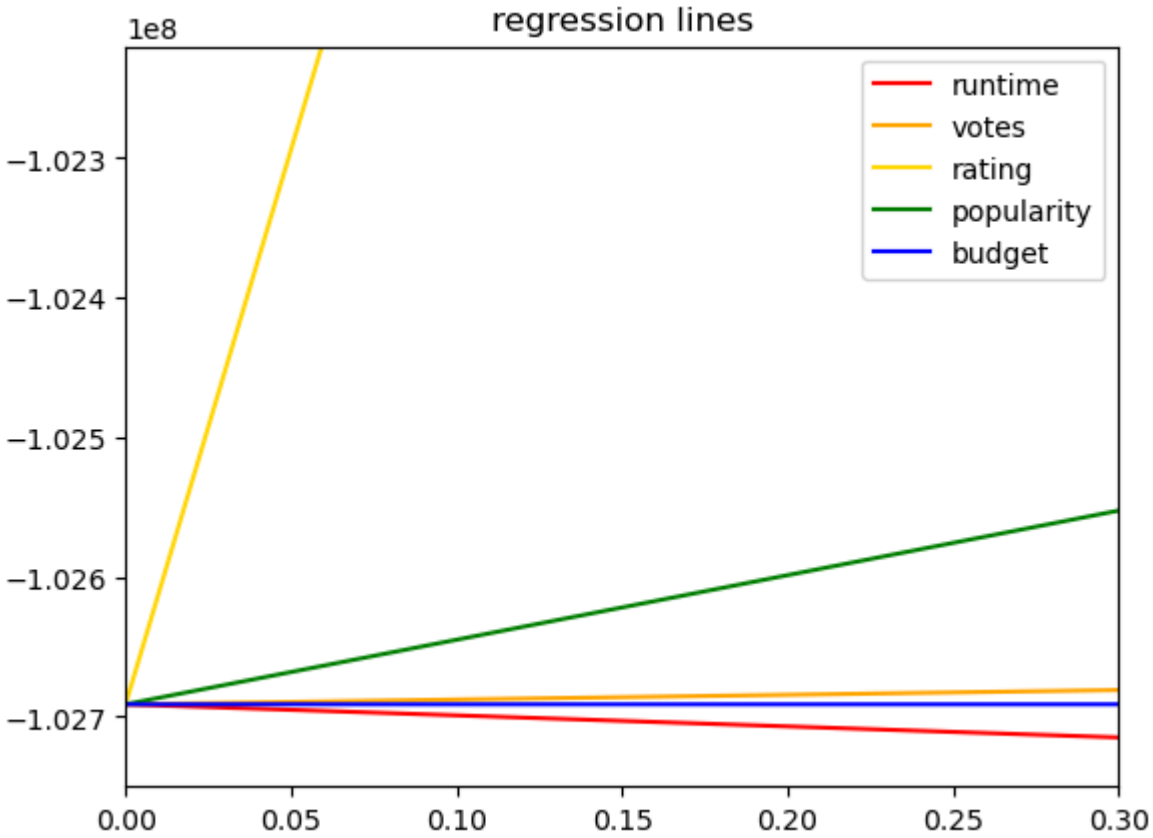
In [127…
```
range_m = range(m.size)
plt.bar(range_m, m, color=feature_colors)
plt.ylabel('coefficient')
plt.xticks(range_m, x.columns)
```

Out[127…
```
([<matplotlib.axis.XTick at 0x70b954b7d6d0>,
  <matplotlib.axis.XTick at 0x70b954b62870>,
  <matplotlib.axis.XTick at 0x70b954ac1f40>,
  <matplotlib.axis.XTick at 0x70b954b63d10>,
  <matplotlib.axis.XTick at 0x70b95499ac60>],
 [Text(0, 0, 'runtime'),
  Text(1, 0, 'votes'),
  Text(2, 0, 'rating'),
  Text(3, 0, 'popularity'),
  Text(4, 0, 'budget')])
```



In [128…
```
for i in range(m.size): plt.axline(xy1=(0,b), slope=m[i],color=feature_colors[i])
plt.xlim(0, 0.3)
plt.ylim(-1.0275*10**8, -1.0222*10**8)
plt.title('regression lines')
plt.legend(labels=x.columns)
```

Out[128…
```
<matplotlib.legend.Legend at 0x70b95499af30>
```



In [129…

```
# credit: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ri

from sklearn.linear_model import Ridge

model = Ridge(alpha=1.0)
model.fit(train_x, train_y)
pred_y = model.predict(test_x)
```

In [130…

```
out_rmse.append(rmse(pred_y, test_y))
out_rmse
```

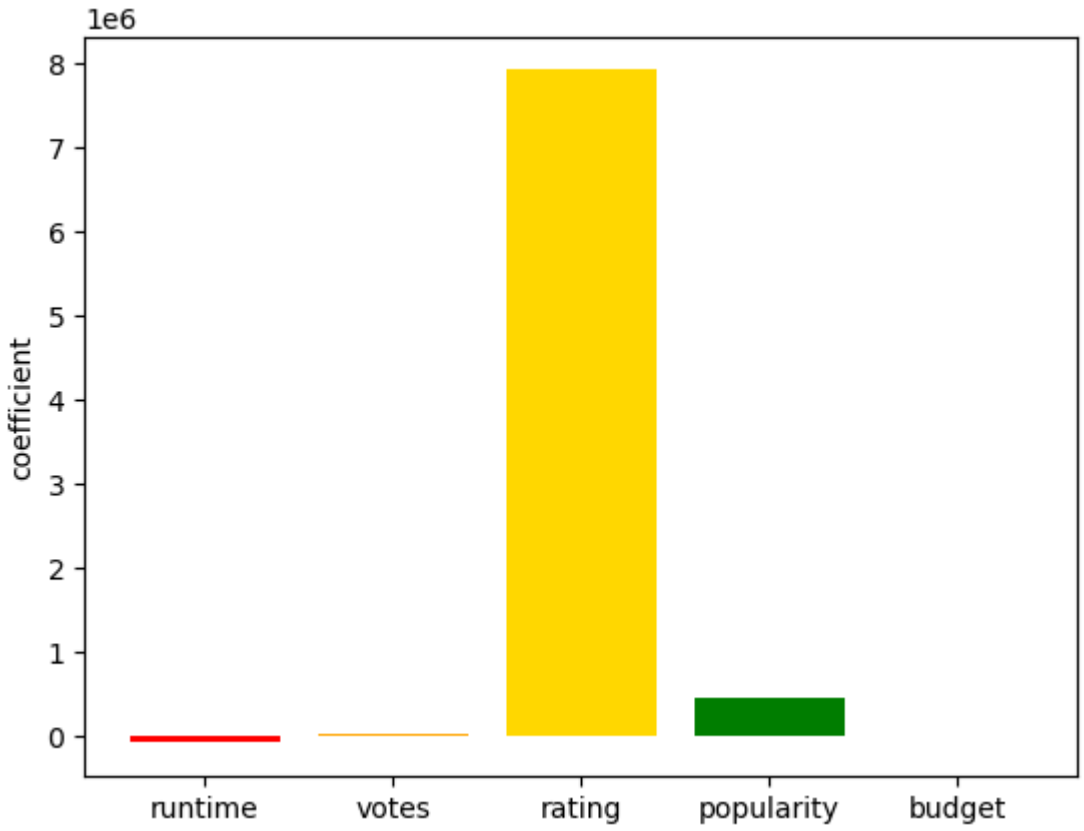
Out[130…
```
[np.float64(156033403.11981857),
 np.float64(156033403.10274816),
 np.float64(156031637.21807626)]
```

In [131…
```
m = model.coef_
b = model.intercept_
m
```

Out[131…
```
array([-7.87795265e+04,  3.40824298e+04,  7.92263406e+06,  4.62427972e+05,
        1.02988788e+00])
```

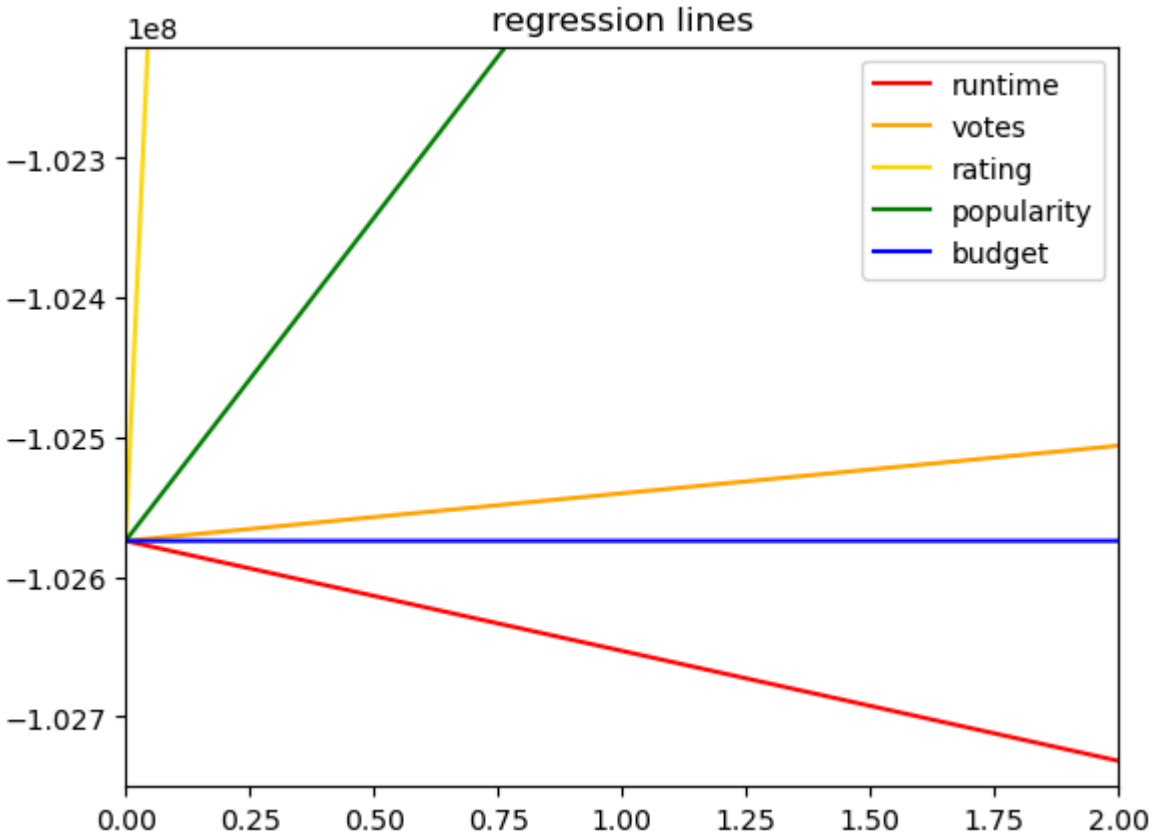
In [132…
```
range_m = range(m.size)
plt.bar(range_m, m, color=feature_colors)
plt.ylabel('coefficient')
plt.xticks(range_m, x.columns)
```

Out[132…
```
([<matplotlib.axis.XTick at 0x70b9549ec860>,
  <matplotlib.axis.XTick at 0x70b954a62870>,
  <matplotlib.axis.XTick at 0x70b954a62f60>,
  <matplotlib.axis.XTick at 0x70b954a63d10>,
  <matplotlib.axis.XTick at 0x70b9548a3740>],
 [Text(0, 0, 'runtime'),
  Text(1, 0, 'votes'),
  Text(2, 0, 'rating'),
  Text(3, 0, 'popularity'),
  Text(4, 0, 'budget')])
```



In [133…
```
for i in range(m.size): plt.axline(xy1=(0,b), slope=m[i],color=feature_colors[i])
plt.xlim(0, 2)
plt.ylim(-1.0275*10**8, -1.0222*10**8)
plt.title('regression lines')
plt.legend(labels=x.columns)
```

Out[133…
```
<matplotlib.legend.Legend at 0x70b9548edf70>
```



In [134…
```
# credit: https://medium.com/@nandiniverma78988/implementation-of-elastic-net-regression-for-predictive-modeling-in

from sklearn.linear_model import ElasticNet

model = ElasticNet(alpha=0.5, l1_ratio=0.5)
model.fit(train_x, train_y)
pred_y = model.predict(test_x)
```

In [135…
```
out_rmse.append(rmse(pred_y, test_y))
out_rmse
```

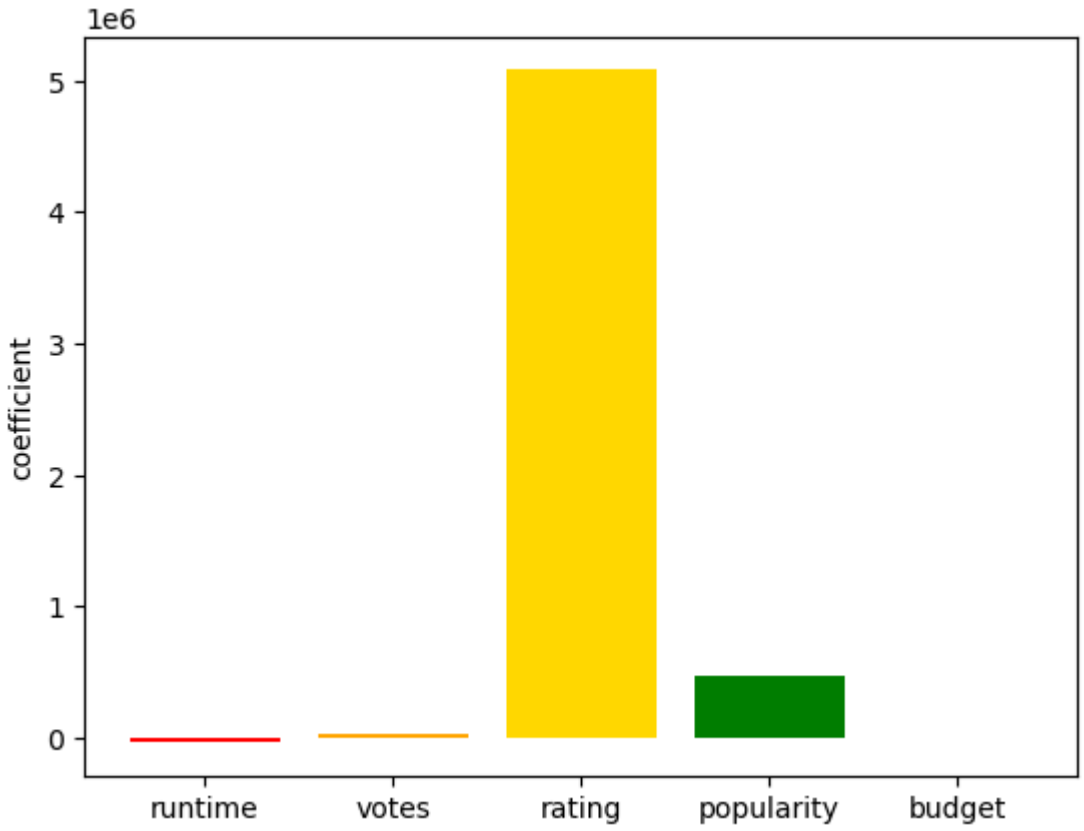
Out[135…
```
[np.float64(156033403.11981857),
 np.float64(156033403.10274816),
 np.float64(156031637.21807626),
 np.float64(155823545.68853328)]
```

In [136…
```
m = model.coef_
b = model.intercept_
m
```

Out[136…
```
array([-4.15940481e+04,  3.42604322e+04,  5.08188686e+06,  4.73703123e+05,
        1.01997479e+00])
```

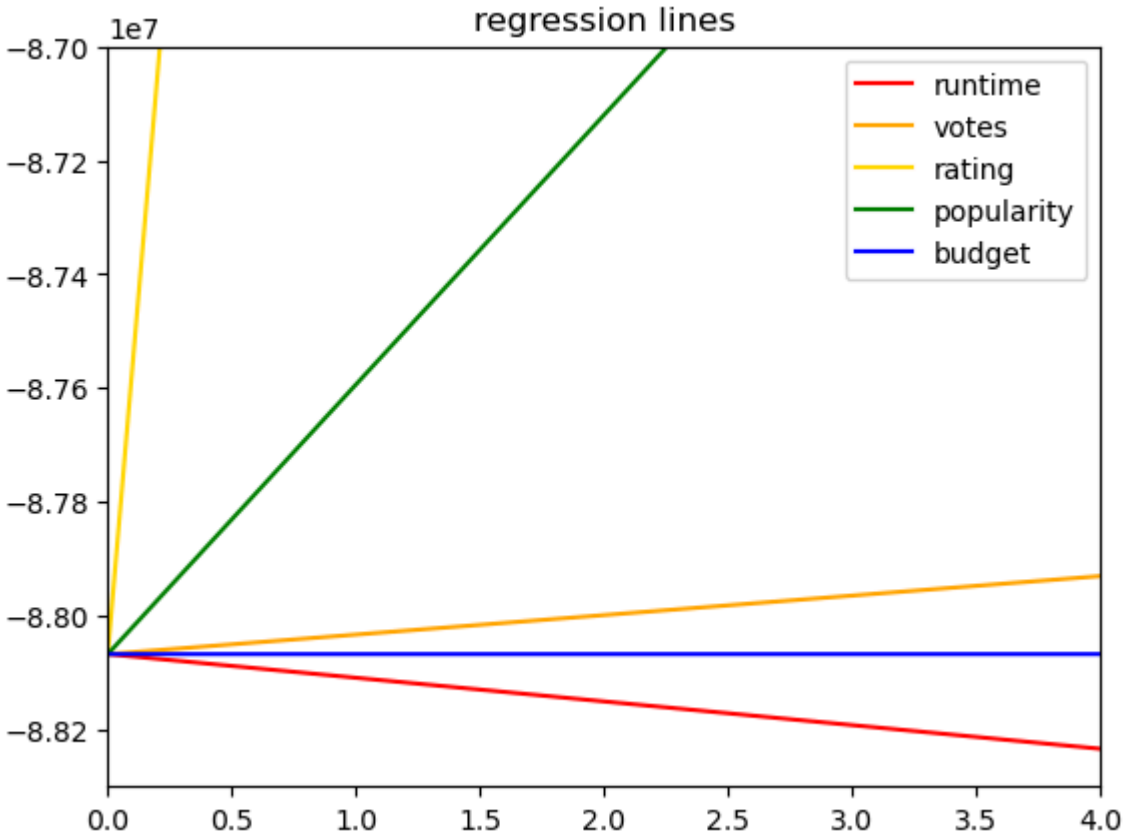
In [137…
```
range_m = range(m.size)
plt.bar(range_m, m, color=feature_colors)
plt.ylabel('coefficient')
plt.xticks(range_m, x.columns)
```

Out[137…
```
([<matplotlib.axis.XTick at 0x70b9548ed4c0>,
  <matplotlib.axis.XTick at 0x70b9548fd0a0>,
  <matplotlib.axis.XTick at 0x70b9548a2a50>,
  <matplotlib.axis.XTick at 0x70b9547bd010>,
  <matplotlib.axis.XTick at 0x70b9547bd940>],
 [Text(0, 0, 'runtime'),
  Text(1, 0, 'votes'),
  Text(2, 0, 'rating'),
  Text(3, 0, 'popularity'),
  Text(4, 0, 'budget')])
```



In [138…
```
for i in range(m.size): plt.axline(xy1=(0,b), slope=m[i],color=feature_colors[i])
plt.xlim(0, 4)
plt.ylim(-8.83*10**7, -8.7*10**7)
plt.title('regression lines')
plt.legend(labels=x.columns)
```

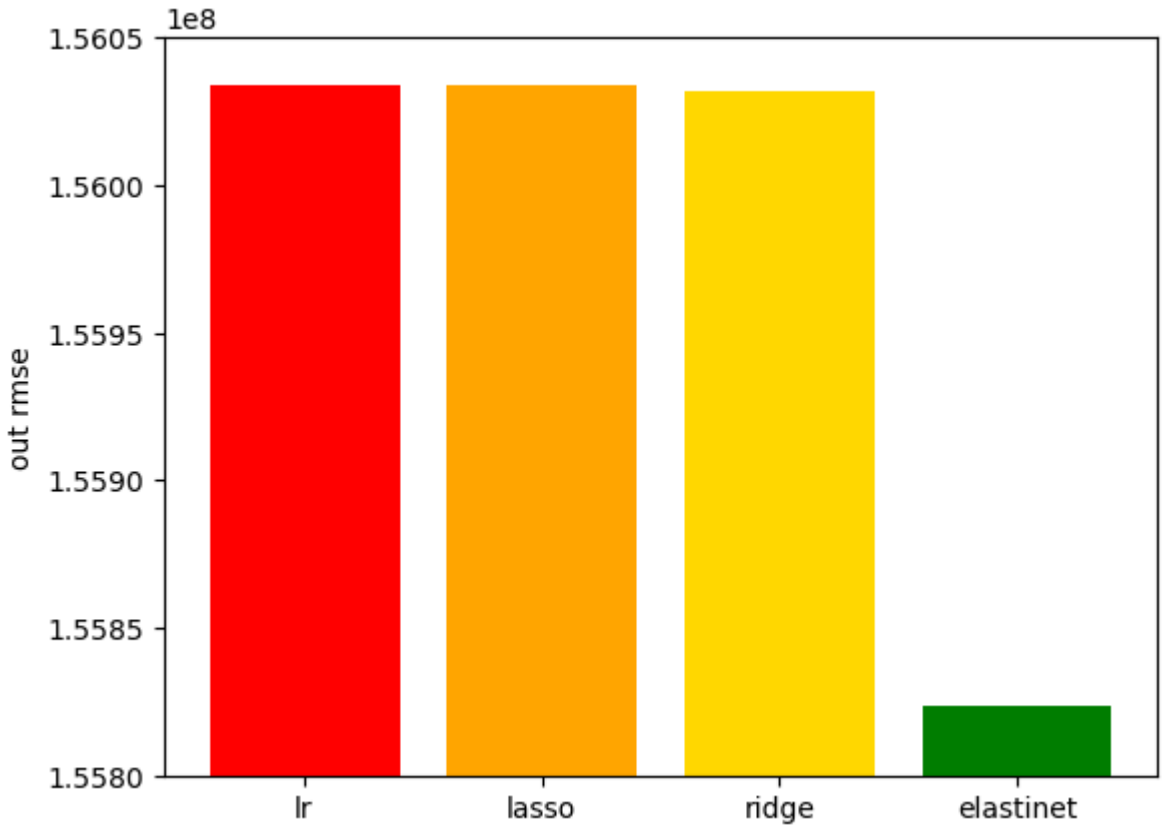
Out[138…
```
<matplotlib.legend.Legend at 0x70b9548fc0e0>
```



Consistently, rating and polularity demonstrate a strongly positive coefficient, suggesting strong positive correlation between each of these features and margin, while other features demonstrate weakly positive or negative coefficients. The root mean squared errors between multilinear regression approaches are subsequently compared.

In [139…
```
plt.bar(range(4), out_rmse, color=feature_colors)
plt.ylim(155800000, 156050000)
plt.xticks(range(4), ['lr', 'lasso', 'ridge', 'elastinet'])
plt.ylabel('out rmse')
```

Out[139…
```
Text(0, 0.5, 'out rmse')
```



RMSE is lower for ElastiNet. Using ElastiNet, the multilinear regression approach is again applied, this time for all genres with respect to the single label, margin. As before, the test and train data are split, the model is fit to the train data using multilinear regression, and the model predicts labels from the test data. RMSE is computed, and the coefficients and intercept of the equation for the regression lines are computed.

In [140…

```
x = movies.iloc[:,8:]
y = movies.margin
x
train_x = []
test_x = []
train_y = []
test_y = []
train_x, test_x, train_y, test_y = tts(x,y,test_size=0.2,random_state=42)
```

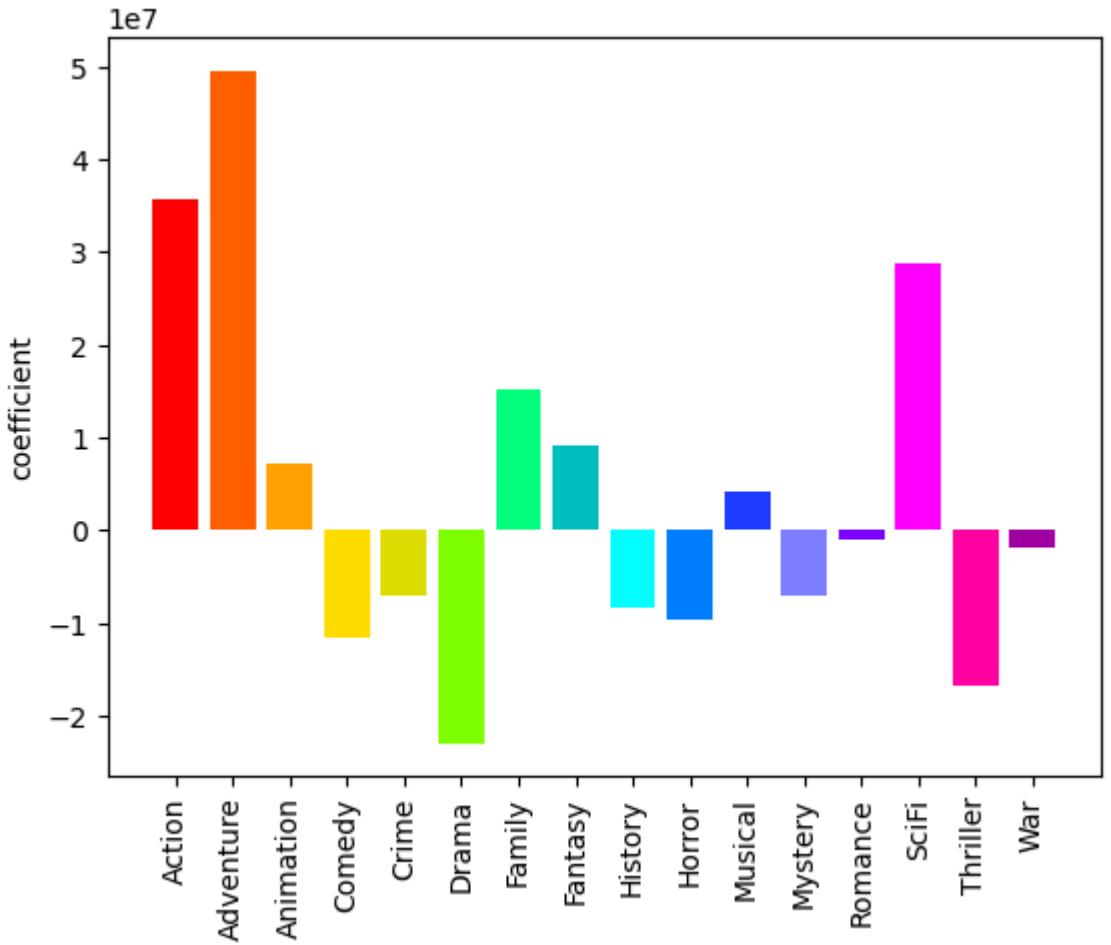
```
model = ElasticNet(alpha=0.5, l1_ratio=0.5)
model.fit(train_x, train_y)
pred_y = model.predict(test_x)
out_rmse
m = model.coef_
b = model.intercept_
m
```

Out[140…
```
array([ 35759432.86589195,  49600663.98494968,   7228203.63060208,
       -11668599.38361933,  -7069640.91658986, -22928866.65922069,
        15241498.09652513,   9236762.25108693,  -8431222.07479491,
        -9629251.89468266,   4236172.05192311,  -7136833.44420907,
        -1011284.80194708,  28713893.7142905 , -16661059.30377297,
        -1909138.88371447])
```

In [141…
```
feature_colors = genre_colors
```

In [142…
```
range_m = range(m.size)
plt.bar(range_m, m, color=feature_colors)
plt.ylabel('coefficient')
plt.xticks(range_m, x.columns, rotation='vertical')
plt.yticks()
```

Out[142…
```
(array([-30000000., -20000000., -10000000.,         0.,  10000000.,
         20000000.,  30000000.,  40000000.,  50000000.,  60000000.]),
 [Text(0, -30000000.0, '−3'),
  Text(0, -20000000.0, '−2'),
  Text(0, -10000000.0, '−1'),
  Text(0, 0.0, '0'),
  Text(0, 10000000.0, '1'),
  Text(0, 20000000.0, '2'),
  Text(0, 30000000.0, '3'),
  Text(0, 40000000.0, '4'),
  Text(0, 50000000.0, '5'),
  Text(0, 60000000.0, '6')])
```

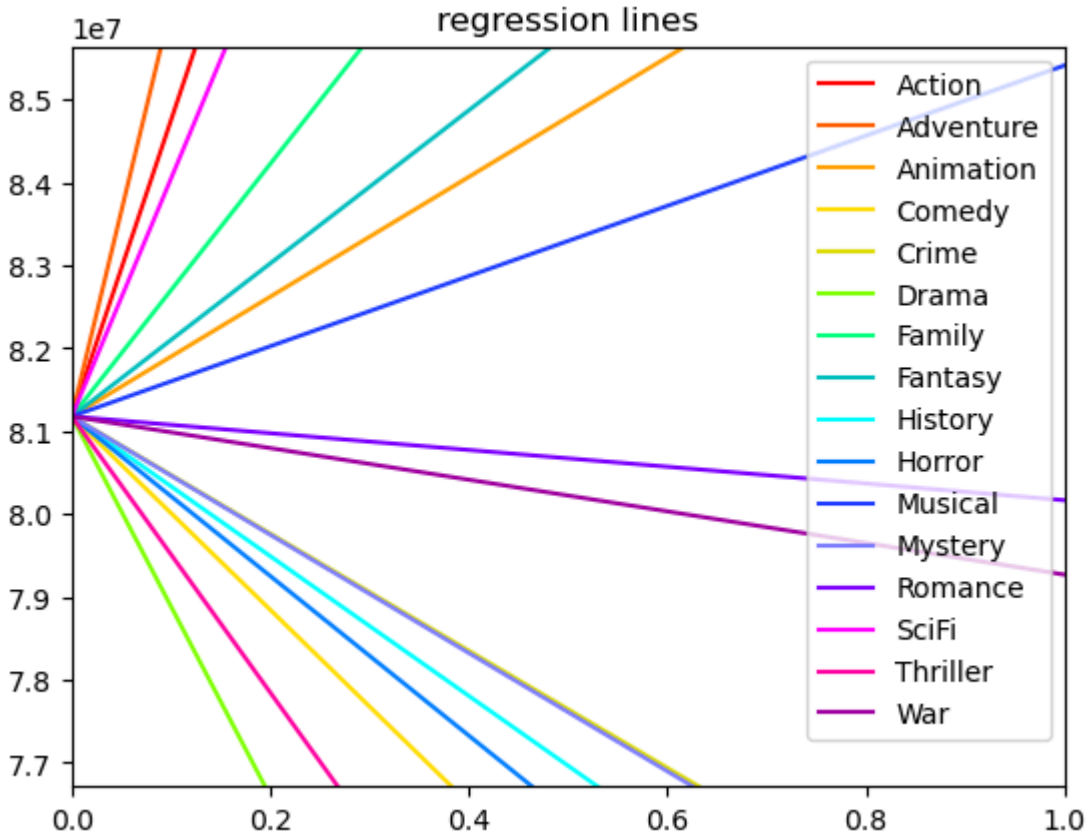


In [143…

```
for i in range(m.size): plt.axline(xy1=(0,b), slope=m[i],color=feature_colors[i])
plt.xlim(0, 1)
plt.title('regression lines')
plt.legend(labels=x.columns)
```

Out[143…

```
<matplotlib.legend.Legend at 0x70b954711df0>
```



The features of Adventure, Action, SciFi, Family, Fantasy, Animation, and Musical, in order of magnitude, demonstrate a positive coefficient, suggesting positive correlation between each of these features and margin, while other features demonstrate weakly or strongly negative coefficients.

Inferences

From the above observations and considerations, the following inferences are derived:

- Profitability is significantly impacted by choice of genre.
- Determinants of sales and margin should be analyzed with respect to individual genres in order to detect additional discrepancies.

This report therefore recommends that a production company consider a more profitable genre (especially Adventure, or SciFi, but also Family, Fantasy, or Animation) and analyze individual determinants of sales and margin with respect to the selected genre, then acquire the resources necessary to increase budget, follow industry trends in terms of where to allocate/expend budget (in order to follow the trends observed in this analysis), and aim for a movie runtime of 150 minutes. A best-performing machine learning model, such as a neural net with hyperparameters tuned to reduce overfitting, after training, testing and validation, could predict the desired outcome, such as profitability (which could also be modified to specify a different threshhold value, or alternatively a different quality altogether) provided a combination of control values such as runtime, rating, votes, popularity, budget and genre, which could help a production company decide on how to optimize inputs toward different desired outcomes given a set of constraints.