Making Movie Magic for Microsoft

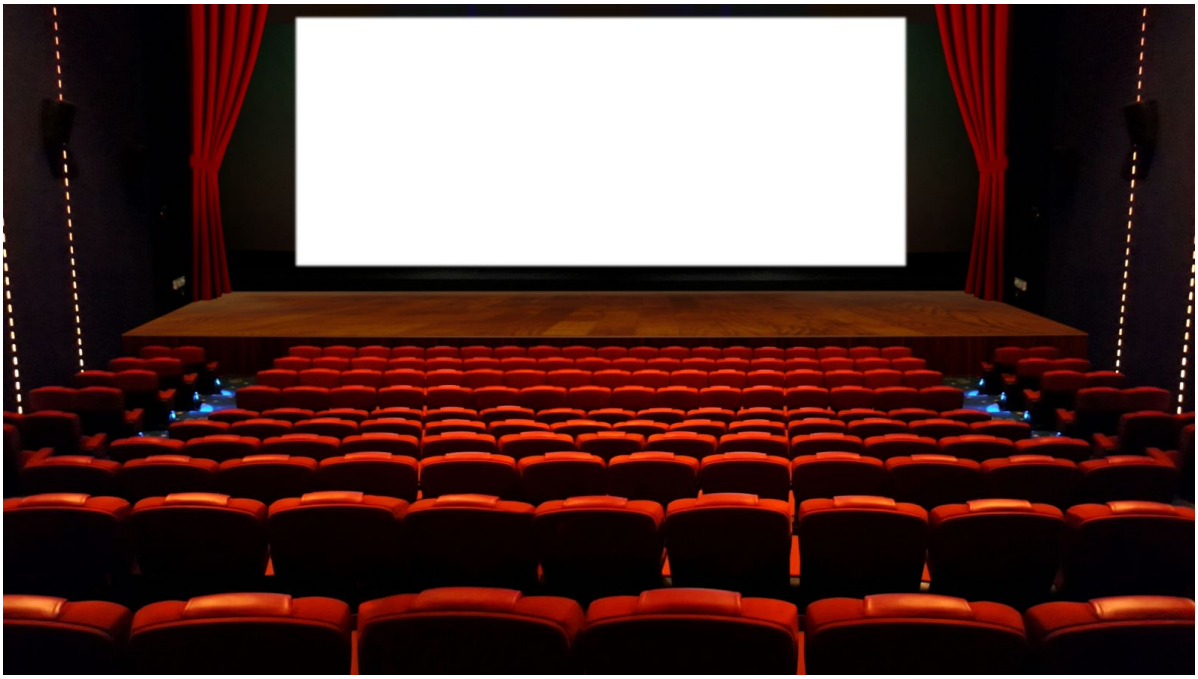
Student Name: Natalya Doris

Student Pace: Flex / 40 weeks

Scheduled Project Review Date / Time: Thursday, May 26 at 11 a.m. EST

Instructor Name: Abhineet Kulkarni

Blog Post URL: https://medium.com/@ntdoris/adding-a-single-regression-line-to-a-scatterplot-with-multiple-categories-1eb804e27ce0



Source: US Chamber of Commerce Foundation (https://www.uschamberfoundation.org/blog/post/night-movies-what-expect-next-time-you-go-theater)

Overview

This project analyzes movie data from Box Office Mojo, IMDB, Rotten Tomatoes, TheMovieDB and The Numbers to better understand what types of movies have been the most successful in the last decade, both from a profitability and popularity perspective.

Analysis of recent historical budget, revenue, genre and review data can help Microsoft determine how to best strategize their impending entrance into the movie market.

Business Problem

Microsoft is considering opening a movie studio, and faces a number of strategic choices, including genre, budget size, and timing. In this analysis, we will take a deep dive into the current landscape of the film industry to determine the most efficient use of their capital across these categories.

Data Understanding

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import sqlite3
import datetime
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import matplotlib.ticker as mtick
```

In [2]:
```
conn = sqlite3.Connection('zippedData/im.db')
table_names = pd.read_sql("""SELECT name FROM sqlite_master WHERE type = 'table
display(table_names)

#df_im_directors = pd.read_sql("""SELECT * FROM directors; """, conn)
df_im_movie_ratings = pd.read_sql("""SELECT * FROM movie_ratings; """, conn)
df_im_movie_basics = pd.read_sql("""SELECT * FROM movie_basics; """, conn)
#df_im_persons = pd.read_sql("""SELECT * FROM persons; """, conn)
#df_im_writers = pd.read_sql("""SELECT * FROM writers; """, conn)

df_im = pd.read_sql("""
SELECT *
FROM movie_basics
JOIN movie_ratings
    USING(movie_id)
""", conn)

conn.close()

df_movie_gross = pd.read_csv('zippedData/bom.movie_gross.csv')
df_movie_info = pd.read_csv('zippedData/rt.movie_info.tsv', delimiter='\t')
df_review = pd.read_csv('zippedData/rt.reviews.tsv', delimiter='\t', encoding='
df_movies = pd.read_csv('zippedData/tmdb.movies.csv')
df_movie_budgets = pd.read_csv('zippedData/tn.movie_budgets.csv')
```

	name
0	movie_basics
1	directors
2	known_for
3	movie_akas
4	movie_ratings
5	persons
6	principals
7	writers

In [3]:
```
display(df_movie_gross.head())
display(df_review.head())
display(df_movie_budgets.head())
display(df_movie_info.head())
display(df_movies.head())
display(df_im.head())
```

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.0	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.0	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.0	664300000	2010
3	Inception	WB	292600000.0	535700000	2010
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

	id	synopsis	rating	genre	director	writer	theater_date
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Ha
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Ince

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

Movie Genre & Review Data

The IMDB dataset includes ~73,000 movies released from 2010 to 2019, with variables describing the movie's genre, runtime, average rating and number of votes. The Movie Database (TMDB) dataset includes ~27,000 movies released from 1930 to 2020, with variables descrbing genre, popularity, vote average & vote count and release date.

In [4]:
```
df_im.info()
df_im['start_year'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
dtypes: float64(2), int64(2), object(4)
memory usage: 4.5+ MB
```

Out[4]:

```
count    73856.000000
mean      2014.276132
std          2.614807
min       2010.000000
25%       2012.000000
50%       2014.000000
75%       2016.000000
max       2019.000000
Name: start_year, dtype: float64
```

In [5]:

```
df_movies.info()

df_movies['year'] = pd.DatetimeIndex(df_movies['release_date']).year
df_movies['year'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

Out[5]:

```
count    26517.000000
mean      2013.953162
std          3.544641
min       1930.000000
25%       2012.000000
50%       2014.000000
75%       2016.000000
max       2020.000000
Name: year, dtype: float64
```

The IMDB dataset is larger but contains movies produced in a shorter span of time, between 2010 and 2019. The TMBD data has fewer rows but a wider span of years, from 1930 to

2020. We will see how each of these datasets merge with the budget & revenue data to determine which we will use.

Movie Budget & Revenue Data

There are also two datasets which contain revenue data. `df_movie_gross` contains ~3,400 movies released between 2010 and 2018, and includes variables for movie studio, domestic gross revenues and foreign gross revenues. `df_movie_budgets` contains ~5,800 movies released between 1915 and 2020, and includes variables for domestic gross revenues, worldwide gross revenues, production budget, and release date. I am inclined to focus on the `df_movie_budgets` dataset given my interest in budget data, but will take a closer look at each to see if there are any other factors I should consider.

In [6]:
```
# Missing lots of data in foreign_gross column, which would make it difficult t
df_movie_gross.info()

df_movie_gross['year'].describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   title           3387 non-null   object
 1   studio          3382 non-null   object
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object
 4   year            3387 non-null   int64 
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

Out[6]:
```
count    3387.000000
mean     2013.958075
std         2.478141
min      2010.000000
25%      2012.000000
50%      2014.000000
75%      2016.000000
max      2018.000000
Name: year, dtype: float64
```

In [7]:
```
df_movie_budgets.info() # much more complete dataset

# create year and month columns
df_movie_budgets['year'] = pd.DatetimeIndex(df_movie_budgets['release_date']).y
df_movie_budgets['month'] = pd.DatetimeIndex(df_movie_budgets['release_date']).

df_movie_budgets['year'].describe() # contains movies from 1915 to 2020
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

Out[7]:

```
count    5782.000000
mean     2003.967139
std        12.724386
min      1915.000000
25%      2000.000000
50%      2007.000000
75%      2012.000000
max      2020.000000
Name: year, dtype: float64
```

`df_movie_budgets` is cleaner, contains more rows, and includes production budget data. `df_movie_gross` only includes movies produced between 2010 and 2018 and over 1000 items are missing foreign gross revenue data, which means we can only look at worldwide gross for about 2/3 of the dataset. Given the cleaner data, inclusion of budget data and larger number of records, I will merge `df_movie_budgets` with the genre data.

Merge Testing

Here I test out different merge combinations to see which yields the most complete/relevant dataset for the majority of our analysis.

In [8]:

```
test = pd.merge(df_im, df_movie_budgets, left_on='primary_title', right_on='mov
test.info()

print("Movies with same title and different start years: ",
      len(test[test['start_year']!=test['year']])) # lots of movies with same t
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2875 entries, 0 to 2874
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   movie_id           2875 non-null   object 
 1   primary_title      2875 non-null   object 
 2   original_title     2875 non-null   object 
 3   start_year         2875 non-null   int64  
 4   runtime_minutes    2757 non-null   float64
 5   genres             2867 non-null   object 
 6   averagerating      2875 non-null   float64
 7   numvotes           2875 non-null   int64  
 8   id                 2875 non-null   int64  
 9   release_date       2875 non-null   object 
 10  movie              2875 non-null   object 
 11  production_budget  2875 non-null   object 
 12  domestic_gross     2875 non-null   object 
 13  worldwide_gross    2875 non-null   object 
 14  year               2875 non-null   int64  
 15  month              2875 non-null   int64  
dtypes: float64(2), int64(5), object(9)
memory usage: 381.8+ KB
Movies with same title and different start years:  1377
```

If two movies have the same name but were released in different years, they will still match, so we need to create a key which includes the year in which the movie was produced to yield a more accurately joined dataset.

In [9]:
```
df_im['test_key'] =  df_im['start_year'].astype(str) + " " + df_im['primary_tit
df_movie_budgets['test_key'] = df_movie_budgets['year'].astype(str) + " " + df_
df_movies['test_key'] = df_movies['year'].astype(str) + " " + df_movies['title'
df_movie_gross['test_key'] = df_movie_gross['year'].astype(str) + " " + df_movi

test = pd.merge(df_im, df_movie_budgets, on='test_key')
display(test.info()) # 1498 length

test2 = pd.merge(df_movies, df_movie_budgets, on='test_key')
display(test2.info()) # 1758 length
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1498 entries, 0 to 1497
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   movie_id           1498 non-null   object 
 1   primary_title      1498 non-null   object 
 2   original_title     1498 non-null   object 
 3   start_year         1498 non-null   int64  
 4   runtime_minutes    1490 non-null   float64
 5   genres             1496 non-null   object 
 6   averagerating      1498 non-null   float64
 7   numvotes           1498 non-null   int64  
 8   test_key           1498 non-null   object 
 9   id                 1498 non-null   int64  
 10  release_date       1498 non-null   object 
 11  movie              1498 non-null   object 
 12  production_budget  1498 non-null   object 
 13  domestic_gross     1498 non-null   object 
 14  worldwide_gross    1498 non-null   object 
 15  year               1498 non-null   int64  
 16  month              1498 non-null   int64  
dtypes: float64(2), int64(5), object(10)
memory usage: 210.7+ KB
None
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1758 entries, 0 to 1757
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         1758 non-null   int64  
 1   genre_ids          1758 non-null   object 
 2   id_x               1758 non-null   int64  
 3   original_language  1758 non-null   object 
 4   original_title     1758 non-null   object 
 5   popularity         1758 non-null   float64
 6   release_date_x     1758 non-null   object 
 7   title              1758 non-null   object 
 8   vote_average       1758 non-null   float64
 9   vote_count         1758 non-null   int64  
 10  year_x             1758 non-null   int64  
 11  test_key           1758 non-null   object 
 12  id_y               1758 non-null   int64  
 13  release_date_y     1758 non-null   object 
 14  movie              1758 non-null   object 
 15  production_budget  1758 non-null   object 
 16  domestic_gross     1758 non-null   object 
 17  worldwide_gross    1758 non-null   object 
 18  year_y             1758 non-null   int64  
 19  month              1758 non-null   int64  
dtypes: float64(2), int64(7), object(11)
memory usage: 288.4+ KB
None
```

The merge between the `df_movies` and `df_movie_budgets` datasets yields the most records and includes the information we are interested in analyzing: domestic/foreign gross revenue, production budget, release year, genre, reviews and popularity. The data is also fairly clean, with some minor improvements needed, which we will tackle in the next section.

Data Preparation

Data Cleaning

Here, we transform financial data into integers from strings so that we can run mathematical and statistical formulas.

In [10]:
```
df_movie_budgets['production_budget'] = [int(budget[1:].replace(',','')) for bu
                                                    df_movie_budgets['production_bu
df_movie_budgets['domestic_gross'] = [int(budget[1:].replace(',','')) for budge
                                                  df_movie_budgets['domestic_gross']
df_movie_budgets['worldwide_gross'] = [int(budget[1:].replace(',','')) for budg
                                                   df_movie_budgets['worldwide_gross
```

Here, we transform date information into a datetime object from a string so that we can look at trends over time and better organize our data.

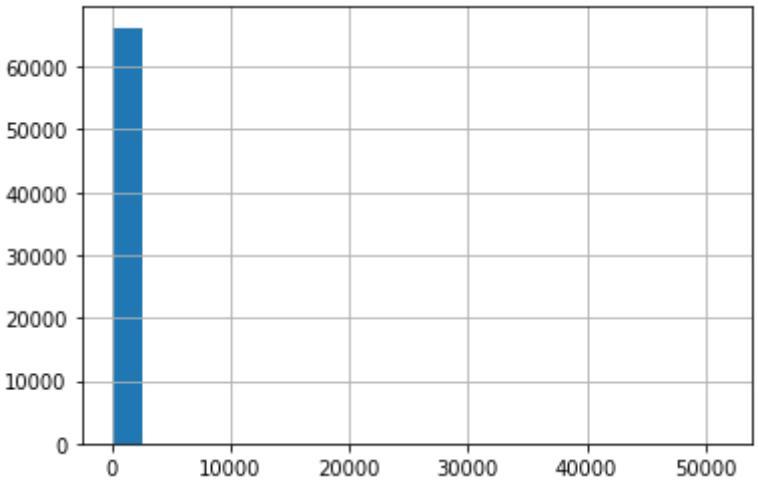
In [11]:
```
display(df_movie_budgets['release_date'][0]) # date format is %b %d, %Y
display(df_movies['release_date'][0]) # date format is %Y-%m-%d

df_movie_budgets['release_date'] = [datetime.datetime.strptime(date, '%b %d, %Y
                                    for date in df_movie_budgets['release_date'
df_movies['release_date'] = [datetime.datetime.strptime(date, '%Y-%m-%d') for d
```

```
'Dec 18, 2009'
'2010-11-19'
```

Here, we deal with missing values in the runtime minutes variable of the IMDB dataset. If it is a fairly normal distribution will assign the mean to missing values. Otherwise we will do further analysis.

In [12]:
```
df_im.info()

df_im['runtime_minutes'].hist(bins=20) # there is clearly some outlier skewing

display(len(df_im[df_im['runtime_minutes']>300])) # 43 movies have a runtime lo
```

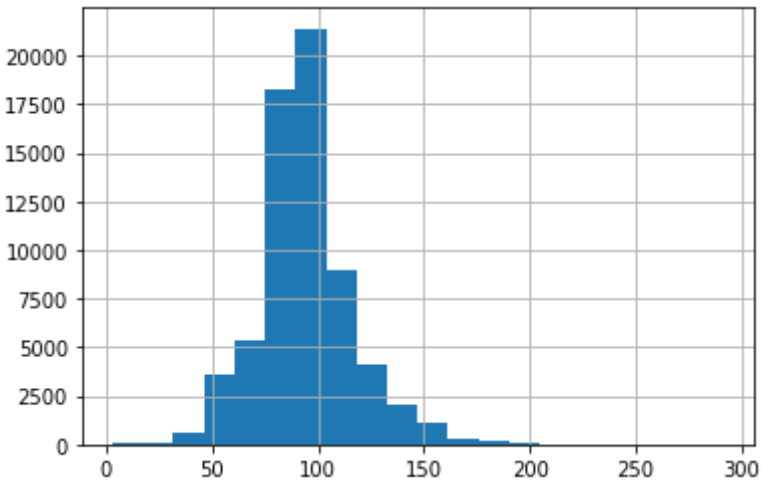
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
 8   test_key         73856 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 5.1+ MB
43
```



When we exclude the 43 outliers, we get a much more normal distribution. We will assign the mean of the <300min length movies to the missing values.

In [13]:
```
df_im['runtime_minutes'][df_im['runtime_minutes']<300].hist(bins=20)

repl = df_im['runtime_minutes'][df_im['runtime_minutes']<300].mean()
df_im['runtime_minutes'] = df_im['runtime_minutes'].fillna(repl)
df_im.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  73856 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
 8   test_key         73856 non-null  object 
dtypes: float64(2), int64(2), object(5)
memory usage: 5.1+ MB
```



Feature Engineering

Since any given movie can have multiple genres, I create a column for each genre. If a movie is of that genre, the value in that column will be 1, otherwise 0.

In [14]:
```
# Genre dictionary for Movie DB dataset
genre_dict = {'Action': '28',
              'Adventure': '12',
              'Animation': '16',
              'Comedy': '35',
              'Crime': '80',
              'Documentary': '99',
              'Drama': '18',
              'Family': '10751',
              'Fantasy': '14',
              'History': '36',
              'Horror': '27',
              'Music': '10402',
              'Mystery': '9648',
              'Romance': '10749',
              'Science Fiction': '878',
              'TV Movie': '10770',
              'Thriller': '53',
              'War': '10752',
              'Western': '37'}

# Transform genre ID to genre
def get_key(val):
    for key, value in genre_dict.items():
        if val == value:
            return key

df_movies['genre_ids'] = [genres.strip('][').split(', ') for genres in df_movie
df_movies['genre_list'] = [[get_key(val) for val in genres] for genres in df_mo

genre_list = list(genre_dict.keys())
import numpy as np
stringsCheck = genre_list
df_movies['genres'] = [','.join(map(str, l)) for l in df_movies['genre_list']]
for i in stringsCheck:
    df_movies[f'{i}'] = np.where(df_movies.genres.str.contains(f'{i}'), 1, 0)

display(df_movies.head())
```

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Ha
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Ince

5 rows × 33 columns

I will perform the same action for the IMDB database, in case we decide to use it later on in the analysis. We will also drop the rows with missing genre given they are a small percentage of the total dataset.

In [15]:
```
df_im.dropna(subset='genres', inplace=True) # drop rows with missing genre

df_im['genre_list'] = [genres.split(',') for genres in df_im['genres']]

genre_list = [item for sublist in df_im['genre_list'] for item in sublist]
genre_list = list(set(genre_list))

# create columns for genre category
import numpy as np
stringsCheck = genre_list
for i in stringsCheck:
    df_im[f'{i}'] = np.where(df_im.genres.str.contains(f'{i}'), 1, 0)
```

Here I create new variables to measure foreign revenues & profitability.

In [16]:
```
df_movie_budgets['foreign_gross'] = df_movie_budgets['worldwide_gross'] - df_mo
df_movie_budgets['profit'] = df_movie_budgets['worldwide_gross'] - df_movie_bud
df_movie_budgets['profit_margin'] = df_movie_budgets['profit'] / df_movie_budge
df_movie_budgets['roi'] = df_movie_budgets['profit'] / df_movie_budgets['produc
df_movie_budgets['pct_foreign'] = df_movie_budgets['foreign_gross'] / df_movie_
df_movie_budgets['pct_foreign'] = df_movie_budgets['pct_foreign'].fillna(0) # r

df_movie_budgets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   release_date       5782 non-null   datetime64[ns]
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   int64
 4   domestic_gross     5782 non-null   int64
 5   worldwide_gross    5782 non-null   int64
 6   year               5782 non-null   int64
 7   month              5782 non-null   int64
 8   test_key           5782 non-null   object
 9   foreign_gross      5782 non-null   int64
 10  profit             5782 non-null   int64
 11  profit_margin      5782 non-null   float64
 12  roi                5782 non-null   float64
 13  pct_foreign        5782 non-null   float64
dtypes: datetime64[ns](1), float64(3), int64(8), object(2)
memory usage: 632.5+ KB
```

Merge Datasets

For reasons stated earlier, we will merge the `df_movies` and `df_movie_budgets` datasets.

In [17]:
```
df_movies.columns
```

Out[17]:
```
Index(['Unnamed: 0', 'genre_ids', 'id', 'original_language', 'original_title',
       'popularity', 'release_date', 'title', 'vote_average', 'vote_count',
       'year', 'test_key', 'genre_list', 'genres', 'Action', 'Adventure',
       'Animation', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Family',
       'Fantasy', 'History', 'Horror', 'Music', 'Mystery', 'Romance',
       'Science Fiction', 'TV Movie', 'Thriller', 'War', 'Western'],
      dtype='object')
```

In [18]:
```
df_movie_budgets.columns
```

Out[18]:
```
Index(['id', 'release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross', 'year', 'month', 'test_key', 'foreign_gross',
       'profit', 'profit_margin', 'roi', 'pct_foreign'],
      dtype='object')
```

In [19]:
```
# create merge key to avoid potential duplicate titles
df_movies['match_key'] = df_movies['year'].astype(str) + " " + df_movies['origi
df_movie_budgets['match_key'] = df_movie_budgets['year'].astype(str) + " " + df

# only keeping columns we need
df_movies = df_movies[['popularity', 'release_date', 'original_language',
                       'vote_average', 'vote_count', 'genre_list', 'genres', 'A
                       'Animation', 'Comedy', 'Crime', 'Documentary', 'Drama',
                       'Horror', 'Music', 'Mystery', 'Romance', 'Science Fictio
                       'Western', 'match_key']]
df_movie_budgets = df_movie_budgets[['movie', 'year', 'production_budget', 'dom
                                     'worldwide_gross', 'month', 'profit', 'pro
                                     'pct_foreign', 'match_key']]
```

```
# merge on key
df_final = pd.merge(df_movie_budgets, df_movies, on='match_key')
```

In [20]:
```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1759 entries, 0 to 1758
Data columns (total 38 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   movie               1759 non-null   object        
 1   year                1759 non-null   int64         
 2   production_budget   1759 non-null   int64         
 3   domestic_gross      1759 non-null   int64         
 4   foreign_gross       1759 non-null   int64         
 5   worldwide_gross     1759 non-null   int64         
 6   month               1759 non-null   int64         
 7   profit              1759 non-null   int64         
 8   profit_margin       1759 non-null   float64       
 9   roi                 1759 non-null   float64       
 10  pct_foreign         1759 non-null   float64       
 11  match_key           1759 non-null   object        
 12  popularity          1759 non-null   float64       
 13  release_date        1759 non-null   datetime64[ns]
 14  original_language   1759 non-null   object        
 15  vote_average        1759 non-null   float64       
 16  vote_count          1759 non-null   int64         
 17  genre_list          1759 non-null   object        
 18  genres              1759 non-null   object        
 19  Action              1759 non-null   int64         
 20  Adventure           1759 non-null   int64         
 21  Animation           1759 non-null   int64         
 22  Comedy              1759 non-null   int64         
 23  Crime               1759 non-null   int64         
 24  Documentary         1759 non-null   int64         
 25  Drama               1759 non-null   int64         
 26  Family              1759 non-null   int64         
 27  Fantasy             1759 non-null   int64         
 28  History             1759 non-null   int64         
 29  Horror              1759 non-null   int64         
 30  Music               1759 non-null   int64         
 31  Mystery             1759 non-null   int64         
 32  Romance             1759 non-null   int64         
 33  Science Fiction     1759 non-null   int64         
 34  TV Movie            1759 non-null   int64         
 35  Thriller            1759 non-null   int64         
 36  War                 1759 non-null   int64         
 37  Western             1759 non-null   int64         
dtypes: datetime64[ns](1), float64(5), int64(27), object(5)
memory usage: 535.9+ KB
```

In [21]:
```
df_final['year'].describe()
```

Out[21]:
```
count    1759.000000
mean     2013.191018
std         4.934025
min      1946.000000
25%      2011.000000
50%      2014.000000
75%      2016.000000
max      2019.000000
Name: year, dtype: float64
```

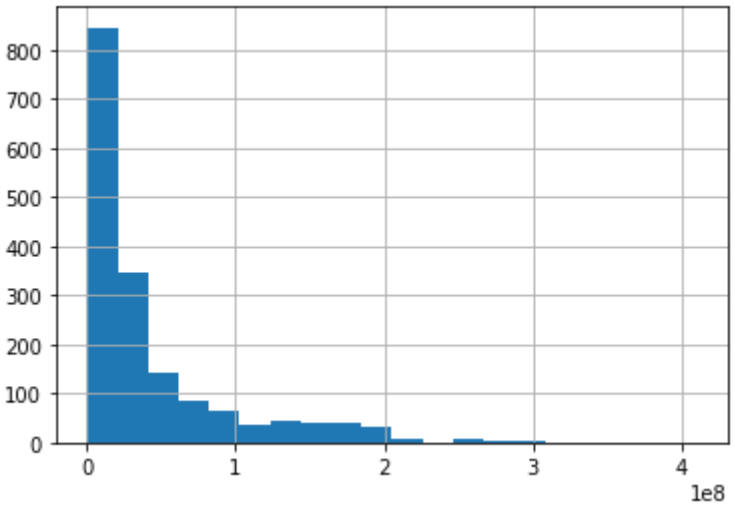
In [22]:
```
# Create CSV file with final dataset
df_final.to_csv('final_dataset.csv')
```

Data Analysis

Given Microsoft's size, I make the assumption that they will not seek to create low budget films. I will also focus on films released in 2010 or later since we are more interested in how recent films have been performing. To determine what we should consider a low budget film, I calculate the interquartile range of production budgets and look at the first quartile.

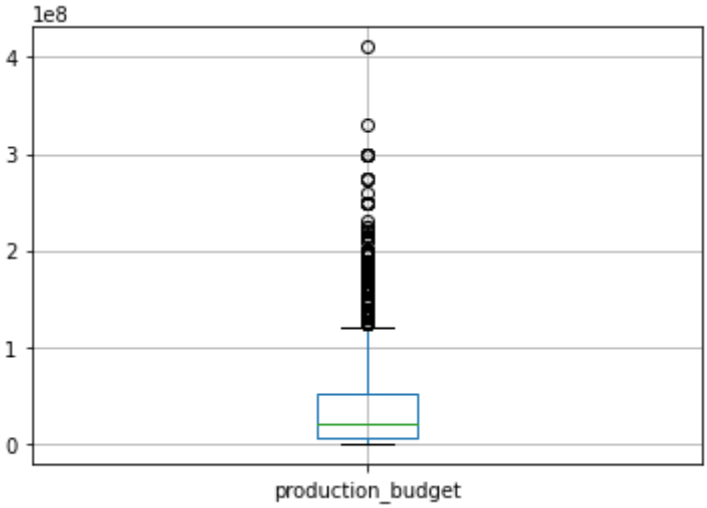
In [23]:
```
df_final[(df_final['year'] >= 2010)]['production_budget'].hist(bins=20) # distr
# clear right skew - lots of movies at moderage budgets, few with very high buc
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:
```
df_final[(df_final['year'] >= 2010)].boxplot(column='production_budget')
```

Out[24]:
```
<AxesSubplot:>
```



Since this data is skewed, we will use the interquartile range & quartiles to categorize the data. Based on this analysis, the Low Budget Film Category should be any movie with a budget less than $7,000,000

In [25]:
```
q1, q2, q3, q4 = np.percentile(df_final[(df_final['year'] >= 2010)]['productior
iqr = q3 - q1

print("Interquartile range:", iqr)
print("First Quartile:", q1)
print("Second Quartile:", q2)
print("Third Quartile:", q3)
print("Fourth Quartile:", q4)
```

```
Interquartile range: 45250000.0
First Quartile: 7000000.0
Second Quartile: 21000000.0
Third Quartile: 52250000.0
Fourth Quartile: 410600000.0
```

Creating a dataset for movies released after 2010 which excludes low budget films.

In [26]:
```
df_final_subset = df_final[(df_final['year'] >= 2010) & (df_final['production_b
```

Adding a categorical variable for budget. Since we removed low budget films, we create medium, high, and very high categories.

In [27]:
```
for k,v in df_final_subset.iterrows():
    if df_final_subset.loc[k, 'production_budget'] < 21000000:
        df_final_subset.loc[k, 'budget_category'] = 'medium'
    elif ((df_final_subset.loc[k, 'production_budget'] >= 21000000) &
          (df_final_subset.loc[k, 'production_budget'] < 52250000)):
        df_final_subset.loc[k, 'budget_category'] = 'high'
    else:
        df_final_subset.loc[k, 'budget_category'] = 'very high'
```

```
/var/folders/wl/4cw_k4nj07d773kdv1fw53tc0000gn/T/ipykernel_26978/3950909464.p
y:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_final_subset.loc[k, 'budget_category'] = 'very high'
```

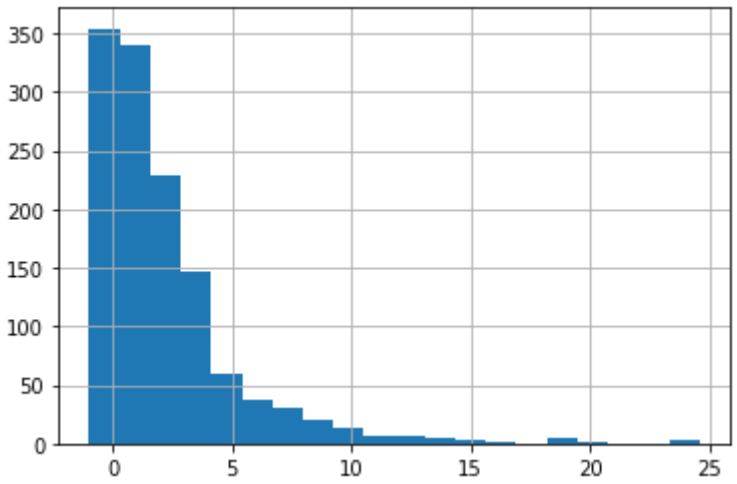
In [28]:
```
df_final_subset['year'].describe()
```

Out[28]:
```
count    1260.000000
mean     2013.977778
std         2.526960
min      2010.000000
25%      2012.000000
50%      2014.000000
75%      2016.000000
max      2019.000000
Name: year, dtype: float64
```

Taking a look at the distribution of some of the other variables.

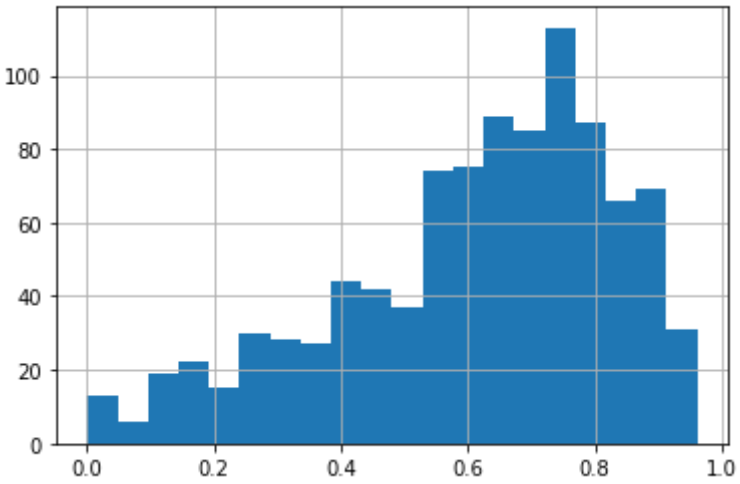
In [29]:
```
df_final_subset['roi'].hist(bins=20) # right skew
```

Out[29]:
```
<AxesSubplot:>
```



In [30]:
```
df_final_subset[df_final_subset['profit_margin'] >= 0]['profit_margin'].hist(bi
```

Out[30]:
```
<AxesSubplot:>
```

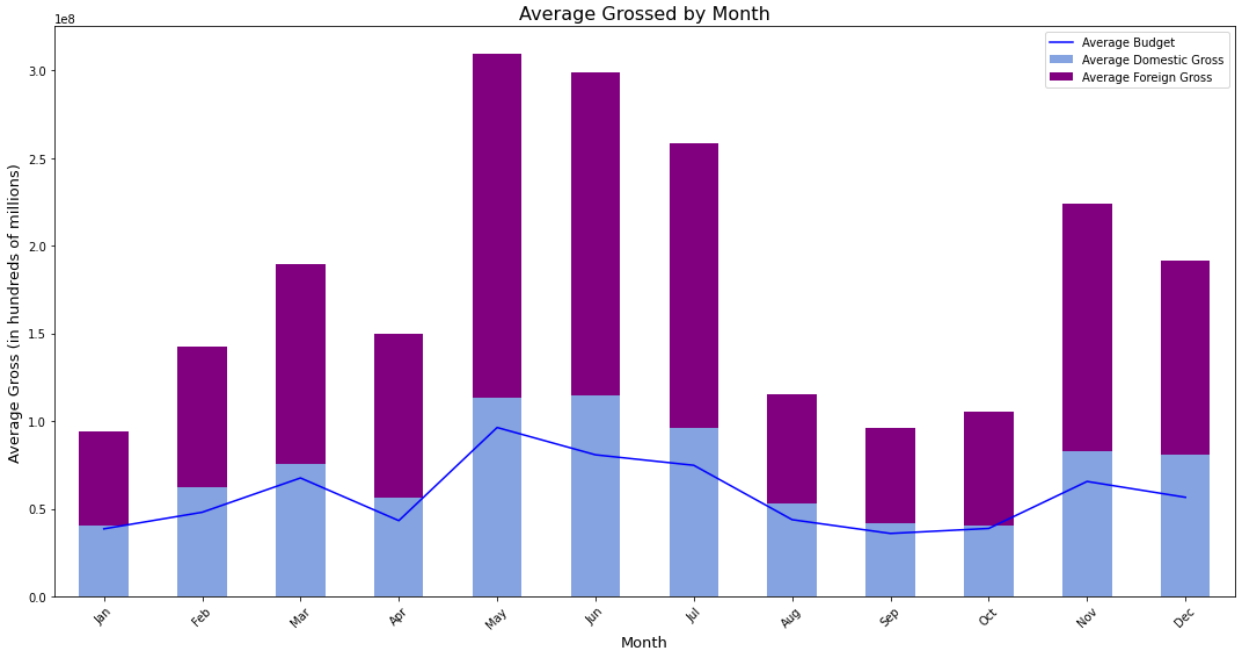


Seasonality

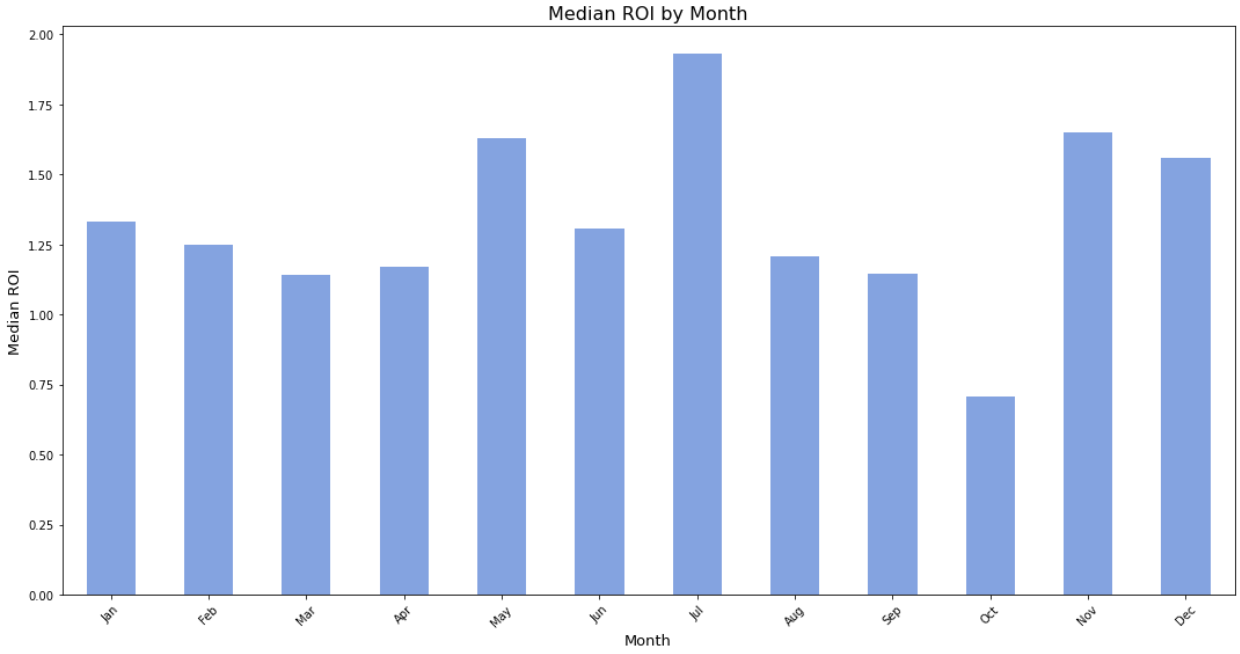
Both gross revenue and ROI data suggest that the summer months (May, June, July) and winter holidays (Nov, Dec) are the best time to release a movie. Foreign revenues tend to represent a larger percentage of revenues during these months as well.

In [31]:
```
season = pd.DataFrame()

season['month'] = list(set(df_final_subset['month']))
season['month_str'] = [datetime.datetime.strptime(str(month), "%m").strftime("%
season['avg_dom_gross'] = [(df_final_subset['domestic_gross'][df_final_subset['
                            for month in season['month']]
season['avg_fn_gross'] = [(df_final_subset['foreign_gross'][df_final_subset['mo
                            for month in season['month']]
season['avg_ww_gross'] = [(df_final_subset['worldwide_gross'][df_final_subset['
                            for month in season['month']]
season['avg_budget'] = [(df_final_subset['production_budget'][df_final_subset['
                            for month in season['month']]
season['roi'] = [(df_final_subset['roi'][df_final_subset['month']==month].media
                            for month in season['month']]
season['pct_foreign'] = [(df_final_subset['pct_foreign'][df_final_subset['month
                            for month in season['month']]

season = season.sort_values('month')
```

In [32]:
```
ax = season[['month_str', 'avg_dom_gross', 'avg_fn_gross']].plot(x='month_str',
                                                                 figsize=(15,8)
                                                                 color=[(0.2, 0
season.plot(y='avg_budget', x='month_str', ax=ax, color='blue')
plt.xlabel('Month', fontsize=13)
plt.xticks(rotation = 45)
plt.ylabel('Average Gross (in hundreds of millions)', fontsize=13)
plt.title('Average Grossed by Month', fontsize=16)
plt.legend(labels=['Average Budget', 'Average Domestic Gross', 'Average Foreign
plt.tight_layout()
plt.savefig("seasons.png", format='png',dpi=150)
```

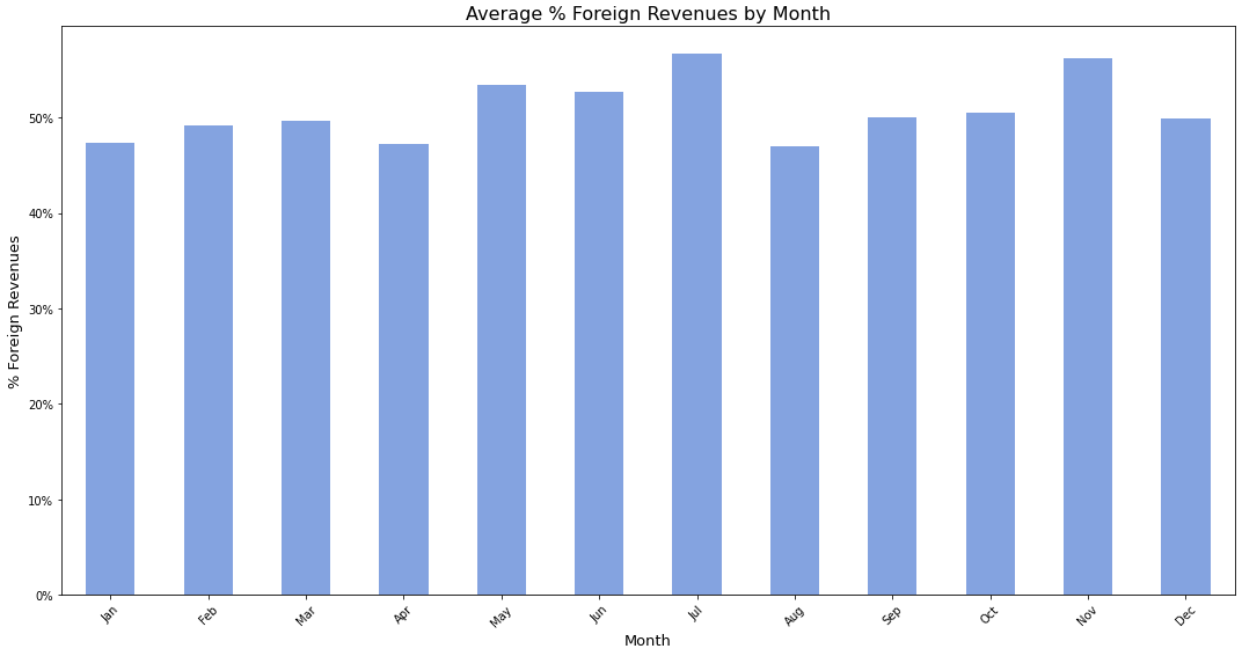


In [33]:
```
ax = season[['month_str', 'roi']].plot(x='month_str', kind='bar', figsize=(15,8
plt.xlabel('Month', fontsize=13)
plt.ylabel('Median ROI', fontsize=13)
plt.title('Median ROI by Month', fontsize=16)
plt.legend(labels=['ROI'])
plt.xticks(rotation = 45)
ax.get_legend().remove()
plt.tight_layout()
plt.savefig("seasons2.png", format='png',dpi=150)
```



In [34]:
```
ax = season[['month_str', 'pct_foreign']].plot(x='month_str', kind='bar', figsi
plt.xlabel('Month', fontsize=13)
plt.ylabel('% Foreign Revenues', fontsize=13)
plt.title('Average % Foreign Revenues by Month', fontsize=16)
plt.xticks(rotation = 45)
ax.yaxis.set_major_formatter(mtick.PercentFormatter(1))
ax.get_legend().remove()
```

```
plt.tight_layout()
plt.savefig("seasons3.png", format='png',dpi=150)
```



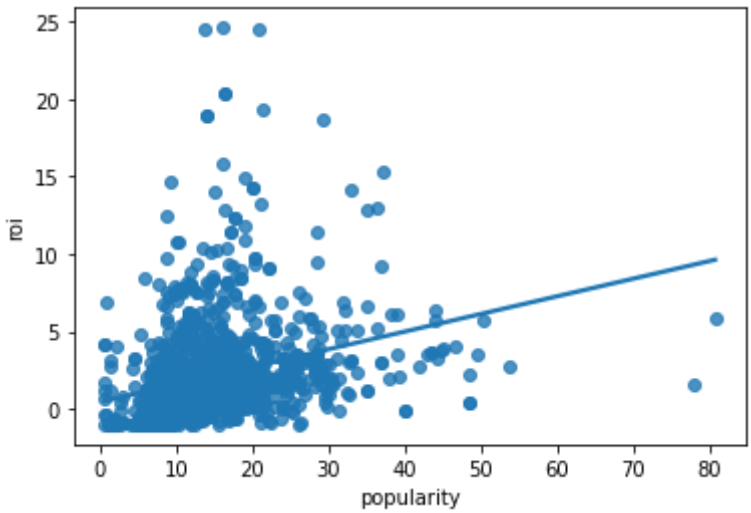
Exploring Relationships

The data implies a positive correlation between movie popularity & ROI and vote average & ROI.

In [35]:
```
sns.regplot(df_final_subset['popularity'], df_final_subset['roi'], ci=None)
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[35]:
```
<AxesSubplot:xlabel='popularity', ylabel='roi'>
```



In [36]:
```
df_final_subset['roi'].describe()
```

Out[36]:
```
count    1260.000000
mean        2.078917
std         3.140772
min        -1.000000
25%         0.125157
50%         1.324828
75%         2.952290
max        24.597236
Name: roi, dtype: float64
```

As we saw earlier, the ROI data is fairly skewed, so we will exclude some of the outliers to get a better representation of the data.

In [37]:
```
q1, q3 = np.percentile(df_final_subset['roi'], [25, 75])
iqr = q3 - q1

p95 = np.percentile(df_final_subset['roi'], 95)

print("Interquartile range:", iqr)
print("First Quartile:", q1)
print("Third Quartile:", q3)
print("95th Percentile:", p95)
```

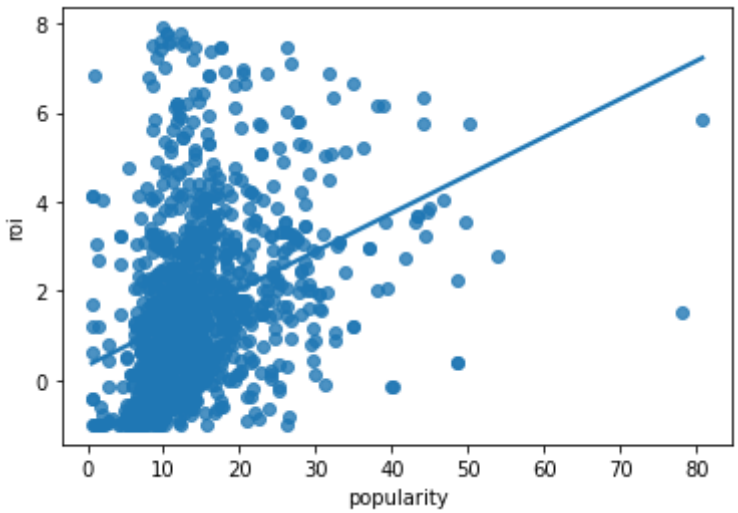
```
Interquartile range: 2.8271326874213836
First Quartile: 0.12515717924528302
Third Quartile: 2.952289866666667
95th Percentile: 8.0500023675
```

The positive slope of the regression line implies a positive relationship between a movie's popularity score and ROI.

In [38]:
```
sns.regplot(df_final_subset['popularity'][df_final_subset['roi'] < 8],
            df_final_subset['roi'][df_final_subset['roi'] < 8], ci=None)
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[38]:
```
<AxesSubplot:xlabel='popularity', ylabel='roi'>
```



Adding categories for budget shows us that medium budget films tend to have the lowest popularity scores, while high and very high budget films have greater popularity. There is also a big cluster of medium budget films among lower ROIs, but each budget category seems to include a full range of ROIs.

In [39]:
```
fig = plt.figure(figsize=(12, 8), dpi=80)
fig, ax = plt.subplots(figsize=(12, 8), dpi= 80, facecolor='w', edgecolor='k')

plot = sns.scatterplot(x='popularity', y='roi', data=df_final_subset[df_final_s
                       hue=df_final_subset.budget_category, legend='full', alp
                       palette='mako')

# adding regression line
sns.regplot(data=df_final_subset[df_final_subset['roi'] < 8], x='popularity', y
            ax=ax, ci=False, color='g', line_kws={'alpha':0.4})

fig.suptitle("Return on Investment vs Movie Popularity", fontsize=18)
ax.set_xlabel("Popularity Score", fontsize=15)
ax.set_ylabel("ROI", fontsize=15)
ax.get_legend().set_title("Budget Size")

plt.tight_layout()
plt.savefig("roi_popularity.png", format='png',dpi=150)
```

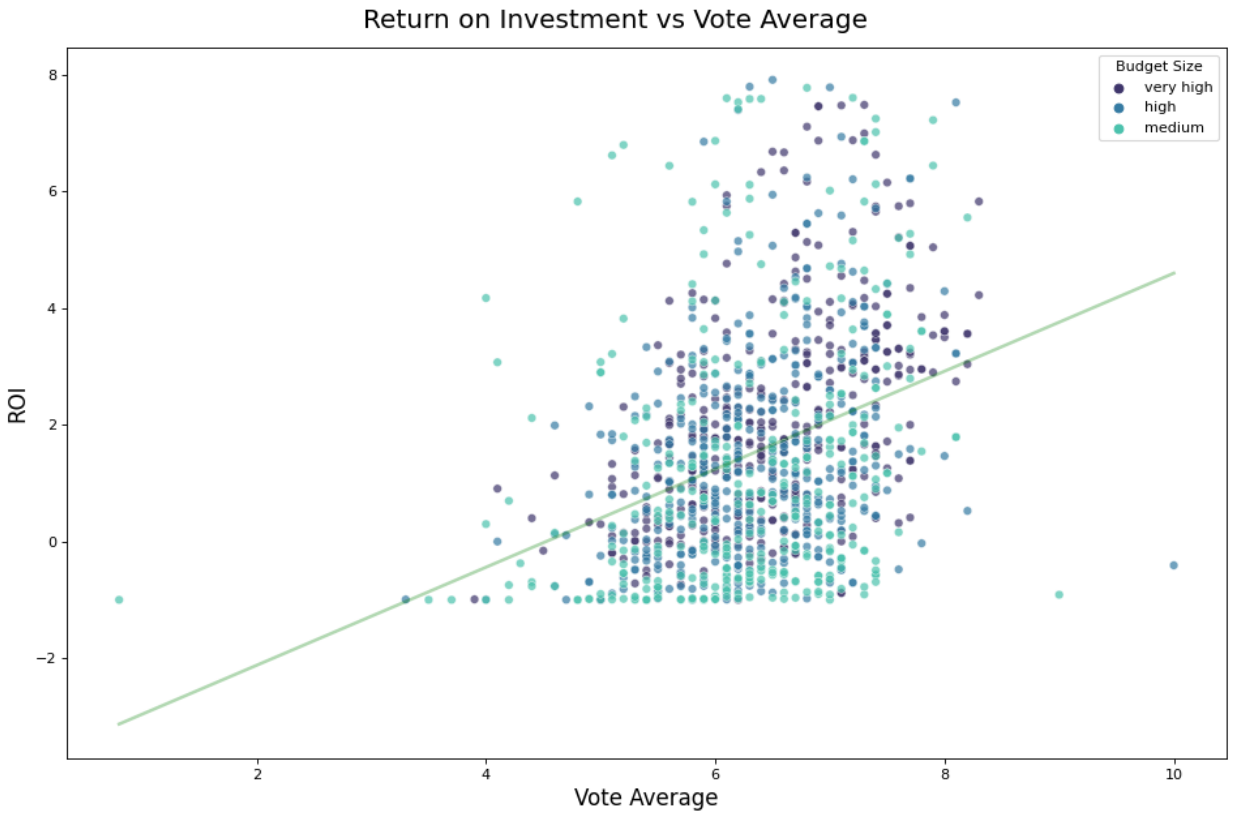
```
<Figure size 960x640 with 0 Axes>
```

Now we will look at the relationship between a movie's voted rating and ROI.

In [40]:
```
fig = plt.figure(figsize=(12, 8), dpi=80)

fig, ax = plt.subplots(figsize=(12, 8), dpi= 80, facecolor='w', edgecolor='k')
```

```
plot = sns.scatterplot(x='vote_average', y='roi', data = df_final_subset[(df_fi
                       hue=df_final_subset.budget_category, legend='full', alph
sns.regplot(data=df_final_subset[df_final_subset['roi'] < 8], x='vote_average',
            ax=ax, ci=False, color='g', line_kws={'alpha':0.3})

fig.suptitle("Return on Investment vs Vote Average", fontsize=18)
ax.set_xlabel("Vote Average", fontsize=15)
ax.set_ylabel("ROI", fontsize=15)
ax.get_legend().set_title("Budget Size")

plt.tight_layout()
```

```
<Figure size 960x640 with 0 Axes>
```



Excluding movies with ROI of -1 to make the plot a bit cleaner:

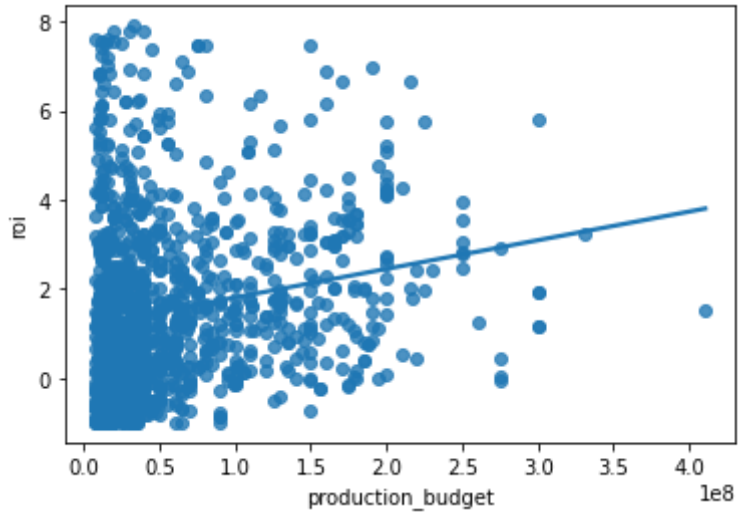
In [41]:
```
fig = plt.figure(figsize=(12, 8), dpi=80)

fig, ax = plt.subplots(figsize=(12, 8), dpi= 80, facecolor='w', edgecolor='k')

plot = sns.scatterplot(x='vote_average', y='roi',
                       data = df_final_subset[(df_final_subset['roi'] < 8) & (d
                       hue=df_final_subset.budget_category, legend='full', alph
sns.regplot(data=df_final_subset[(df_final_subset['roi'] < 8) & (df_final_subse
            x='vote_average', y='roi', scatter=False, ax=ax, ci=False, color='g

fig.suptitle("Return on Investment vs Vote Average", fontsize=18)
ax.set_xlabel("Vote Average", fontsize=15)
ax.set_ylabel("ROI", fontsize=15)
ax.get_legend().set_title("Budget Size")

plt.tight_layout()
plt.savefig("roi_vote.png", format='png',dpi=150)
```

```
<Figure size 960x640 with 0 Axes>
```

In [42]:
```
# budget vs roi - not the cleanest looking plot, slightly positive slope

sns.regplot(df_final_subset['production_budget'][df_final_subset['roi'] < 8],
            df_final_subset['roi'][df_final_subset['roi'] < 8], ci=None)
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[42]:
```
<AxesSubplot:xlabel='production_budget', ylabel='roi'>
```



Genre Analysis

Animation is the top scoring genre (incorporating popularity, vote average, ROI and profit margin). Horror movies tend to have very high returns on investment but yield lesser reviews.

In [43]:
```
print("Movies with more than one genre: " + str(len(df_final_subset[df_final_su
print("Movies with one genre: " + str(len(df_final_subset[df_final_subset['genr
```

```
Movies with more than one genre: 1067
Movies with one genre: 193
```

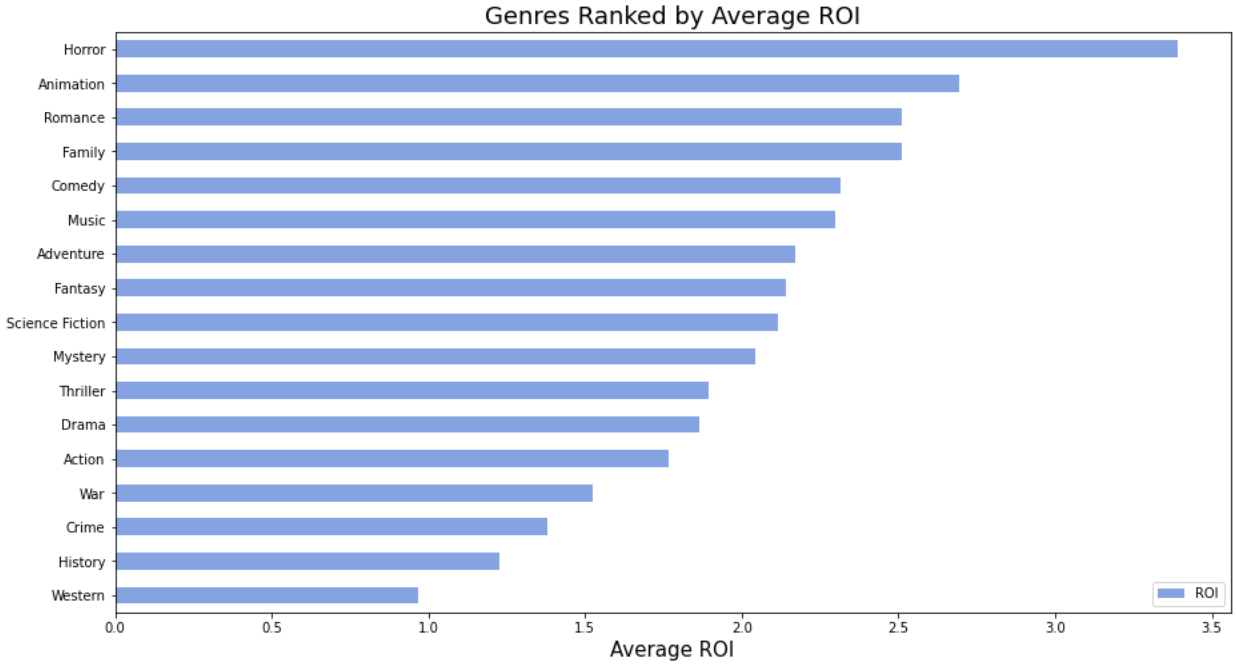
In [44]:
```
genres = pd.DataFrame()
genres['genre'] = list(set([item for sublist in df_final_subset['genre_list'] f

# dropping empty values
genres = genres[~genres['genre'].isnull()].reset_index(drop=True)

genres['count'] = [sum(df_final_subset[f'{genre}']) for genre in genres['genre'
genres['gross'] = [sum(df_final_subset['worldwide_gross'][df_final_subset[f'{ge
                   for genre in genres['genre']]
genres['profit'] = [sum(df_final_subset['profit'][df_final_subset[f'{genre}']==
                    for genre in genres['genre']]
genres['avg_profit_per_movie'] = genres['profit'] / genres['count']
genres['avg_budget'] = [df_final_subset['production_budget'][df_final_subset[f'
                        for genre in genres['genre']]
genres['roi'] = [df_final_subset['roi'][df_final_subset[f'{genre}']==1].mean()
genres['pct_foreign'] = [df_final_subset['pct_foreign'][df_final_subset[f'{genr
                         for genre in genres['genre']]
genres['profit_margin'] = genres['profit'] / genres['gross']
genres['vote_avg'] = [df_final_subset['vote_average'][df_final_subset[f'{genre}
                      for genre in genres['genre']]
genres['popularity'] = [df_final_subset['popularity'][df_final_subset[f'{genre}
                        for genre in genres['genre']]

# budget category
for k,v in genres.iterrows():
    if genres.loc[k, 'avg_budget'] < 21000000:
        genres.loc[k, 'budget_category'] = 'medium'
    elif ((genres.loc[k, 'avg_budget'] >= 21000000) &
          (genres.loc[k, 'avg_budget'] < 52250000)):
        genres.loc[k, 'budget_category'] = 'high'
    else:
        genres.loc[k, 'budget_category'] = 'very high'
```

In [45]:
```
display(genres.sort_values('roi', ascending=False))

# will exclude documentary category since there are only 6 in the data
genres = genres[genres['genre'] != 'Documentary'].reset_index(drop=True)
```

	genre	count	gross	profit	avg_profit_per_movie	avg_budget	
5	Horror	103	11433894154	8026594154	7.792810e+07	3.308058e+07	3.3
2	Documentary	6	756632425	478632425	7.977207e+07	4.633333e+07	2.8
0	Animation	109	41456366568	30681866568	2.814850e+08	9.884862e+07	2.6
16	Romance	156	18251426642	12619526642	8.089440e+07	3.610192e+07	2.5
7	Family	170	55355999945	39706599945	2.335682e+08	9.205529e+07	2.
15	Comedy	432	73348622494	51983322494	1.203318e+08	4.945671e+07	2.
4	Music	35	4227519643	2948019643	8.422913e+07	3.655714e+07	2.2
13	Adventure	287	114859571883	80914471883	2.819320e+08	1.182756e+08	2.
8	Fantasy	153	57029415024	39420715024	2.576517e+08	1.150895e+08	2.
14	Science Fiction	175	58577281983	40610081983	2.320576e+08	1.026697e+08	2.
9	Mystery	91	10244912167	6639412167	7.296057e+07	3.962088e+07	2.0
17	Thriller	340	43050542607	27572342607	8.109513e+07	4.552412e+07	1.8
6	Drama	586	60581202870	39113902870	6.674727e+07	3.663362e+07	1.8
12	Action	393	110840804013	75148704013	1.912181e+08	9.081959e+07	1.7
3	War	42	5865804654	3730004654	8.880963e+07	5.085238e+07	1.
11	Crime	194	22965852528	14569252528	7.509924e+07	4.328144e+07	1.3
10	History	70	4821672962	2364672962	3.378104e+07	3.510000e+07	1.2
1	Western	18	2927798322	1580798322	8.782213e+07	7.483333e+07	0.9

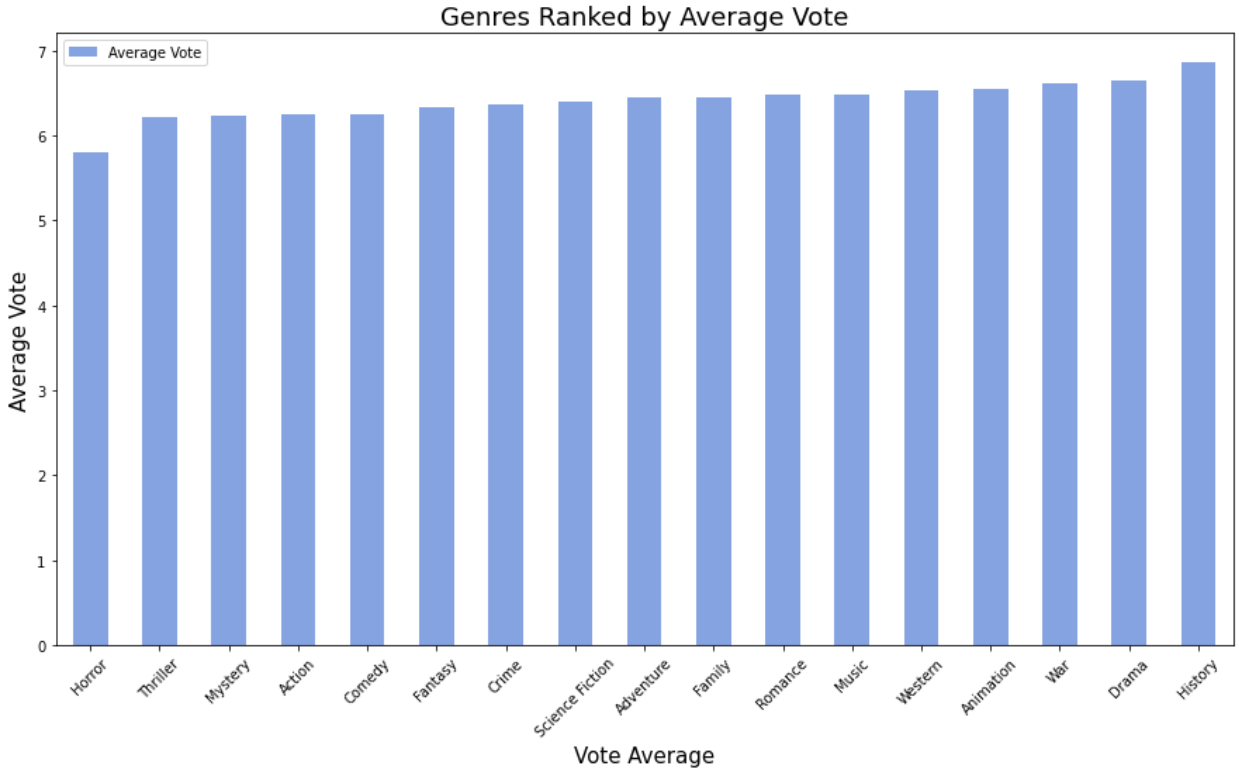
In [46]:
```
# Genre ranked by ROI
genres[['genre', 'roi']].sort_values('roi', ascending=True).plot(x='genre', kir
                                                                 stacked=True,
plt.xlabel('Average ROI', fontsize=15)
plt.ylabel('Genre', fontsize=15).set_visible(False)
plt.title('Genres Ranked by Average ROI',fontsize=18)
plt.legend(labels=['ROI'])
```

Out[46]:
```
<matplotlib.legend.Legend at 0x127c9eb30>
```



In [47]:
```
# Genre ranked by vote average - not a lot of differentiation
genres[['genre', 'vote_avg']].sort_values('vote_avg', ascending=True).plot(x='g
                                                                            figs
                                                                            colo

plt.xlabel('Vote Average', fontsize=15)
plt.ylabel('Average Vote', fontsize=15)
plt.xticks(rotation = 45)
plt.title('Genres Ranked by Average Vote',fontsize=18)
plt.legend(labels=['Average Vote'])
```

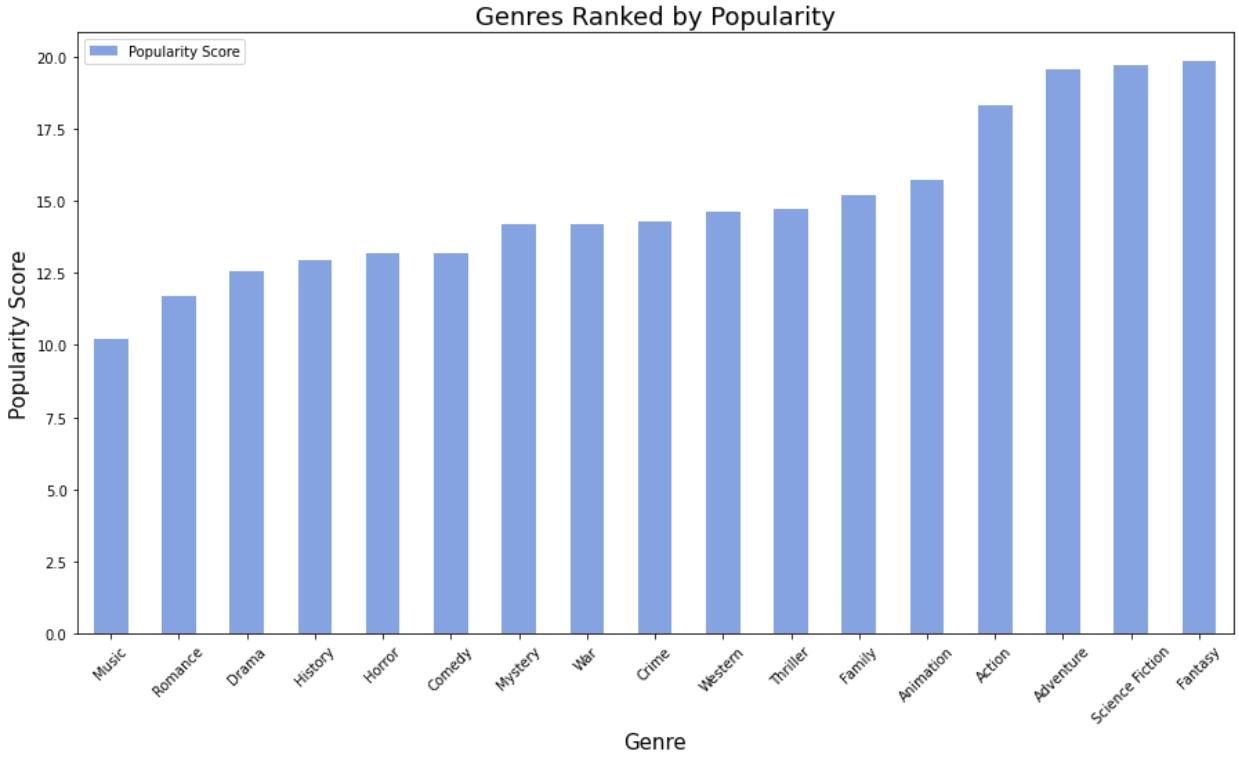
Out[47]:
```
<matplotlib.legend.Legend at 0x126858a30>
```



In [48]:
```
# Genre ranked by popularity
```

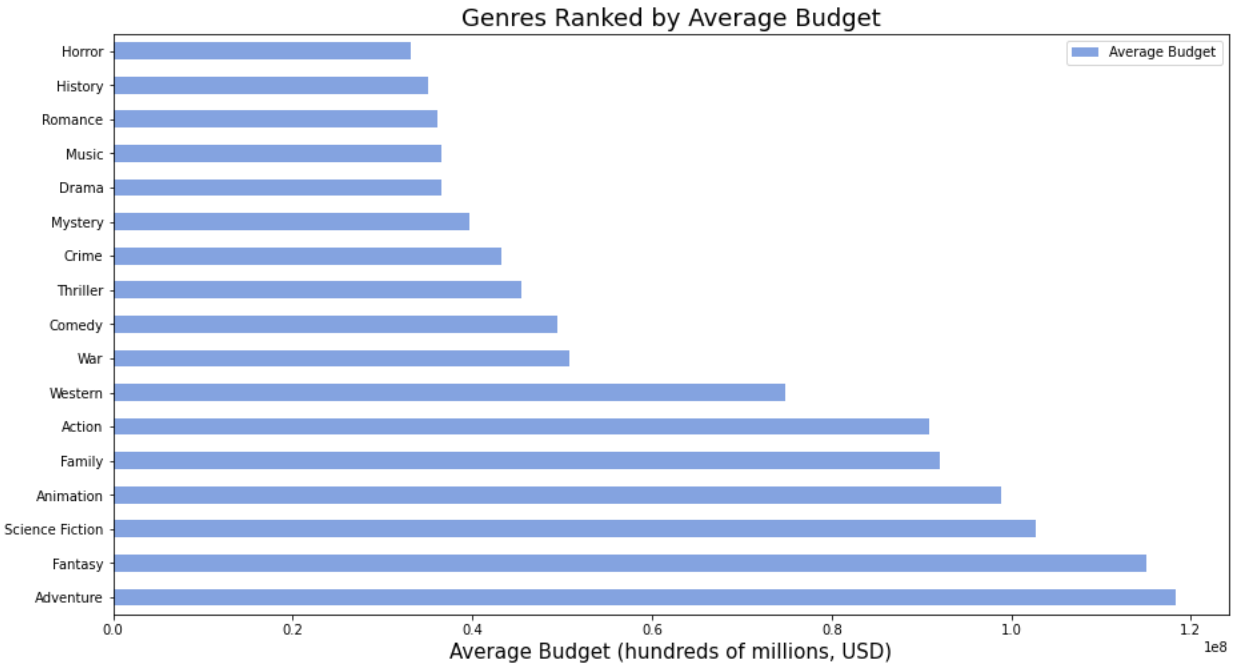
```
genres[['genre', 'popularity']].sort_values('popularity', ascending=True).plot(


plt.xlabel('Genre', fontsize=15)
plt.ylabel('Popularity Score', fontsize=15)
plt.xticks(rotation = 45)
plt.title('Genres Ranked by Popularity',fontsize=18)
plt.legend(labels=['Popularity Score'])
```

Out[48]:
```
<matplotlib.legend.Legend at 0x127c9f8b0>
```

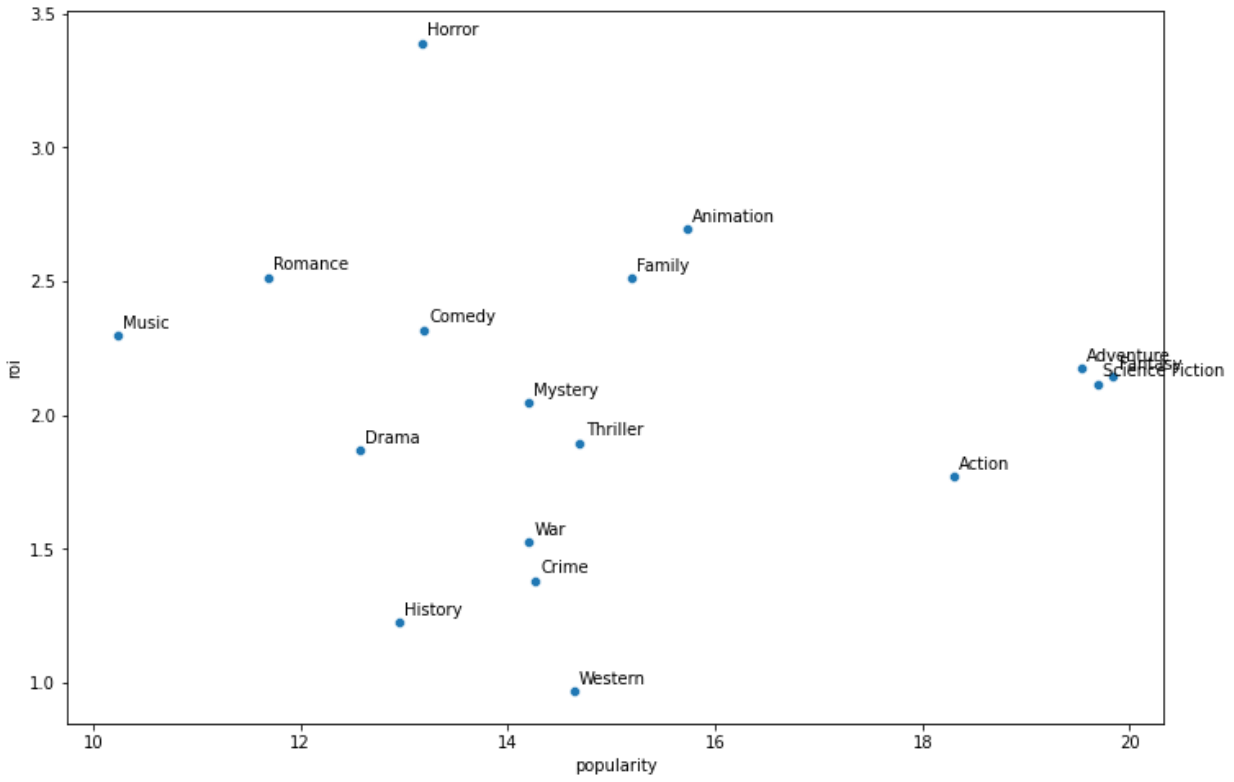


In [49]:
```
# Genre ranked by average budget
genres[['genre', 'avg_budget']].sort_values('avg_budget', ascending=False).plot


plt.ylabel('Genre', fontsize=15).set_visible(False)
plt.xlabel('Average Budget (hundreds of millions, USD)', fontsize=15)
plt.title('Genres Ranked by Average Budget',fontsize=18)
plt.legend(labels=['Average Budget'])
```

Out[49]:
```
<matplotlib.legend.Legend at 0x126824ac0>
```



In [50]:
```
# Define variables for chart
gen = genres['genre']
popularity = genres['popularity']
vote_avg = genres['vote_avg']
roi = genres['roi']

plt.figure(figsize=(12,8))
plot = sns.scatterplot(x = 'popularity', y = 'roi', data=genres)

# Annotate label points
for i, g in enumerate(gen):
    plt.annotate(g, (popularity[i]+0.05, roi[i]+0.03))
```



I want to standardize the ROI, popularity and vote average values so that I can create a composite score & rank genres by that score.

In [51]:
```
# standarizing the data with its standard deviation from the mean
std_scaler = StandardScaler()

#fit the values to the function
genres['roi_scaled'] = std_scaler.fit_transform(genres[['roi']].values)
genres['pm_scaled'] = std_scaler.fit_transform(genres[['profit_margin']].values
genres['pop_scaled'] = std_scaler.fit_transform(genres[['popularity']].values)
genres['vote_scaled'] = std_scaler.fit_transform(genres[['vote_avg']].values)

# source: https://www.journaldev.com/54166/data-scaling-in-python
```

In [52]:
```
genres['score'] = ((1/4) * genres['pm_scaled'] + (1/4) * genres['vote_scaled']
                   (1/4) * genres['pop_scaled'] + (1/4) * genres['roi_scaled'])
```

In [53]:
```
genres2 = genres[['genre', 'count', 'gross', 'avg_profit_per_movie', 'avg_budge
                  'profit_margin', 'vote_avg', 'popularity', 'budget_category',
display(genres2.sort_values('score', ascending=False).style.hide_index())
```

```
/var/folders/wl/4cw_k4nj07d773kdv1fw53tc0000gn/T/ipykernel_26978/610118494.py:
3: FutureWarning: this method is deprecated in favour of `Styler.hide(axis='in
dex')`
  display(genres2.sort_values('score', ascending=False).style.hide_index())
```

genre	count	gross	avg_profit_per_movie	avg_budget	roi	pct_foreig
Animation	109	41456366568	281485014.385321	98848623.853211	2.695279	0.6124
Adventure	287	114859571883	281931957.780488	118275609.756098	2.172222	0.6203
Science Fiction	175	58577281983	232057611.331429	102669714.285714	2.115982	0.6043
Fantasy	153	57029415024	257651732.183007	115089542.483660	2.141698	0.6192
Family	170	55355999945	233568234.970588	92055294.117647	2.512180	0.5793
Romance	156	18251426642	80894401.551282	36101923.076923	2.512485	0.5019
Action	393	110840804013	191218076.368957	90819592.875318	1.767700	0.5777
Comedy	432	73348622494	120331765.032407	49456712.962963	2.316872	0.4527
Horror	103	11433894154	77928098.582524	33080582.524272	3.389840	0.5410
Drama	586	60581202870	66747274.522184	36633617.747440	1.866465	0.4882
Music	35	4227519643	84229132.657143	36557142.857143	2.297203	0.3593
War	42	5865804654	88809634.619048	50852380.952381	1.522781	0.4826
Mystery	91	10244912167	72960573.263736	39620879.120879	2.044564	0.5707
Thriller	340	43050542607	81095125.314706	45524117.647059	1.893624	0.5539
Crime	194	22965852528	75099239.835052	43281443.298969	1.380990	0.5154
History	70	4821672962	33781042.314286	35100000.000000	1.224445	0.5151
Western	18	2927798322	87822129.000000	74833333.333333	0.967484	0.4662

Studio Analysis

How many movies have the top studios produced? What genres do they specialize in? What does their typical movie budget look like?

In [54]:
```
df_movie_gross.info()
df_movie_info.info()

studio = df_movie_gross[['title','studio','year']].dropna(subset='studio')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
 5   test_key        3387 non-null   object 
dtypes: float64(1), int64(1), object(4)
memory usage: 158.9+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

In [55]:
```
studio['match_key'] = studio['year'].astype(str) + " " + studio['title']
df_final_studio = pd.merge(df_final_subset, studio, on='match_key')
```

In [56]:
```
df_final_studio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1026 entries, 0 to 1025
Data columns (total 42 columns):
 #   Column              Non-Null Count  Dtype         
---  ------              --------------  -----         
 0   movie               1026 non-null   object        
 1   year_x              1026 non-null   int64         
 2   production_budget   1026 non-null   int64         
 3   domestic_gross      1026 non-null   int64         
 4   foreign_gross       1026 non-null   int64         
 5   worldwide_gross     1026 non-null   int64         
 6   month               1026 non-null   int64         
 7   profit              1026 non-null   int64         
 8   profit_margin       1026 non-null   float64       
 9   roi                 1026 non-null   float64       
 10  pct_foreign         1026 non-null   float64       
 11  match_key           1026 non-null   object        
 12  popularity          1026 non-null   float64       
 13  release_date        1026 non-null   datetime64[ns]
 14  original_language   1026 non-null   object        
 15  vote_average        1026 non-null   float64       
 16  vote_count          1026 non-null   int64         
 17  genre_list          1026 non-null   object        
 18  genres              1026 non-null   object        
 19  Action              1026 non-null   int64         
 20  Adventure           1026 non-null   int64         
 21  Animation           1026 non-null   int64         
 22  Comedy              1026 non-null   int64         
 23  Crime               1026 non-null   int64         
 24  Documentary         1026 non-null   int64         
 25  Drama               1026 non-null   int64         
 26  Family              1026 non-null   int64         
 27  Fantasy             1026 non-null   int64         
 28  History             1026 non-null   int64         
 29  Horror              1026 non-null   int64         
 30  Music               1026 non-null   int64         
 31  Mystery             1026 non-null   int64         
 32  Romance             1026 non-null   int64         
 33  Science Fiction     1026 non-null   int64         
 34  TV Movie            1026 non-null   int64         
 35  Thriller            1026 non-null   int64         
 36  War                 1026 non-null   int64         
 37  Western             1026 non-null   int64         
 38  budget_category     1026 non-null   object        
 39  title               1026 non-null   object        
 40  studio              1026 non-null   object        
 41  year_y              1026 non-null   int64         
dtypes: datetime64[ns](1), float64(5), int64(28), object(8)
memory usage: 344.7+ KB
```

In [57]:
```
df_final_studio.columns
```

Out[57]:
```
Index(['movie', 'year_x', 'production_budget', 'domestic_gross',
       'foreign_gross', 'worldwide_gross', 'month', 'profit', 'profit_margin',
       'roi', 'pct_foreign', 'match_key', 'popularity', 'release_date',
       'original_language', 'vote_average', 'vote_count', 'genre_list',
       'genres', 'Action', 'Adventure', 'Animation', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Family', 'Fantasy', 'History', 'Horror',
       'Music', 'Mystery', 'Romance', 'Science Fiction', 'TV Movie',
       'Thriller', 'War', 'Western', 'budget_category', 'title', 'studio',
       'year_y'],
      dtype='object')
```

In [58]:
```
# copy of budget variable to see both sum and mean
df_final_studio['avg_budget'] = df_final_studio['production_budget']
studios = pd.DataFrame(df_final_studio.groupby(by='studio').agg({'movie': 'cour
                                                 'production_budget': np.sum,
                                                'avg_budget': np.mean,
                                                'domestic_gross': np.sum,
                                                'foreign_gross': np.sum,
                                                'worldwide_gross': np.sum,
                                                 'profit': np.mean,
                                                 'vote_average': np.mean,
                                                 'vote_count': np.mean,
                                                 'popularity': np.mean,
                                                 'Action': np.sum,
                                                 'Adventure': np.sum,
                                                 'Animation': np.sum,
                                                 'Comedy': np.sum,
                                                 'Crime': np.sum,
                                                 'Documentary': np.sum,
                                                 'Drama': np.sum,
                                                 'Family': np.sum,
                                                 'Fantasy': np.sum,
                                                 'History': np.sum,
                                                 'Horror': np.sum,
                                                 'Music': np.sum,
                                                 'Mystery': np.sum,
                                                 'Romance': np.sum,
                                                 'Science Fiction': np.sum,
                                                 'TV Movie': np.sum,
                                                 'Thriller': np.sum,
                                                 'War': np.sum,
                                                 'Western': np.sum}))

studios['profit_margin'] = (studios['worldwide_gross'] - studios['production_bu
studios['roi'] = (studios['worldwide_gross'] - studios['production_budget']) /
```

In [59]:
```
# budget category
for k,v in studios.iterrows():
    if studios.loc[k, 'avg_budget'] < 21000000:
        studios.loc[k, 'budget_category'] = 'medium'
    elif ((studios.loc[k, 'avg_budget'] >= 21000000) &
          (studios.loc[k, 'avg_budget'] < 52250000)):
        studios.loc[k, 'budget_category'] = 'high'
    else:
        studios.loc[k, 'budget_category'] = 'very high'
```

In [60]:
```
# score
studios['roi_scaled'] = std_scaler.fit_transform(studios[['roi']].values)
studios['pm_scaled'] = std_scaler.fit_transform(studios[['profit_margin']].valu
```

```
studios['pop_scaled'] = std_scaler.fit_transform(studios[['popularity']].values
studios['vote_scaled'] = std_scaler.fit_transform(studios[['vote_average']].val

studios['score'] = ((1/4) * studios['pm_scaled'] + (1/4) * studios['vote_scaled
                    (1/4) * studios['pop_scaled'] + (1/4) * studios['roi_scaled
```

In [61]:
```
# adding a variable for number of genres
studios['gen_count'] = (studios[['Action', 'Adventure', 'Animation', 'Comedy',
                                 'Documentary', 'Drama', 'Family', 'Fantasy', '
                                 'Music', 'Mystery', 'Romance', 'Science Fictio
                                 'Thriller', 'War', 'Western']] != 0).astype(in
```

In [62]:
```
studios.columns
```

Out[62]:
```
Index(['movie', 'production_budget', 'avg_budget', 'domestic_gross',
       'foreign_gross', 'worldwide_gross', 'profit', 'vote_average',
       'vote_count', 'popularity', 'Action', 'Adventure', 'Animation',
       'Comedy', 'Crime', 'Documentary', 'Drama', 'Family', 'Fantasy',
       'History', 'Horror', 'Music', 'Mystery', 'Romance', 'Science Fiction',
       'TV Movie', 'Thriller', 'War', 'Western', 'profit_margin', 'roi',
       'budget_category', 'roi_scaled', 'pm_scaled', 'pop_scaled',
       'vote_scaled', 'score', 'gen_count'],
      dtype='object')
```

In [63]:
```
# selecting relevant columns
studios2 = studios[['movie', 'gen_count','production_budget', 'worldwide_gross'
                    'profit_margin', 'vote_average', 'vote_count', 'popularity'

# only looking at studios that produced more than 5 movies
display(studios2[studios2['movie'] > 5].sort_values('score', ascending=False))
studios[studios['movie'] > 5].sort_values('score', ascending=False).to_csv('stu
```

studio	movie	gen_count	production_budget	worldwide_gross	roi	profit_margin	vote
BV	79	17	10677300000	38248690075	2.582244	0.720845	
FoxS	32	12	488100000	2369639346	3.854823	0.794019	
WB (NL)	37	13	2245000000	9049872986	3.031124	0.751930	
Wein.	36	12	818100000	2767132410	2.382389	0.704351	
Fox	117	17	8535000000	29734509875	2.483832	0.712960	
P/DW	10	8	1334000000	5078027601	2.806617	0.737300	
A24	16	9	183500000	460260186	1.508230	0.601312	
Uni.	113	16	6749200000	27445930630	3.066546	0.754091	
WB	97	17	8773000000	23527640561	1.681824	0.627119	
LGF	54	16	2279600000	7875861491	2.454931	0.710559	
STX	17	11	604100000	1453150394	1.405480	0.584283	
Sony	74	16	5377500000	19380490279	2.603996	0.722530	
LG/S	27	14	1608000000	3567604679	1.218660	0.549277	
Par.	71	18	4993000000	14056291609	1.815200	0.644785	
TriS	13	11	454800000	1383485283	2.041964	0.671265	
Focus	33	14	772800000	2064112073	1.670952	0.625602	
Sum.	11	12	349800000	1242197975	2.551166	0.718402	
CBS	8	9	182000000	473844600	1.603542	0.615908	
SPC	12	8	179200000	506099122	1.824214	0.645919	
ORF	24	14	569700000	1035331792	0.817328	0.449742	
SGem	24	13	721500000	2356561693	2.266198	0.693834	
RAtt.	11	6	184400000	306435564	0.661798	0.398242	
BST	7	9	132000000	140904444	0.067458	0.063195	
Rela.	23	15	679000000	1328915479	0.957166	0.489057	
IFC	6	5	83000000	19844214	-0.760913	-3.182579	
BG	6	9	119000000	98387434	-0.173215	-0.209504	

Key Findings

There are many different avenues by which Microsoft can create a successful movie studio. Here are my recommendations based on this project's analysis:

- Aim to release in summer or winter, as movies that hit the theatre during this times yield the biggest revenues and returns on investment.

- On the lower end of the budget spectrum, focus on horror movies to yield the largest returns. Romance and family earned higher scores than horror, however, when taking into account both returns and popularity/rating. If budget size isn't a barrier, animation is the most successful genre in terms of ROI and popularity/voted rating. Other successful genres in the very high budget category include adventure, sci-fi, fantasy and family.
- Successful studios tended to produce movies across a variety of genres and budgets, implying that there are a number of potential strategies that yield success in the film industry

Next Steps

- While popularity tends to increase with budget size, a movie's average vote seems to be evenly distributed among budget sizes, implying other factors (movie quality, for example) not visible in this analysis impact voted rating. I would like to explore how rating (G / PG / PG-13 / R / Not-Rated), runtime, director quality, among other factors, impact average vote
- I would like to better understand how genre success has evolved over time - are certain genres showing higher growth rates than others? Can we use this data to predict what genres show promise in the future?
- I would also like to see more detailed geographical revenue data - are there certain regions in the world that have produce more ROI & gross revenues than others?

Appendix - Top 20 Movies

In [64]:
```
top_20_gross_mb = df_movie_budgets.sort_values('worldwide_gross', ascending=Fal
top_20_gross_mb = top_20_gross_mb.sort_values('worldwide_gross', ascending=True
top_20_gross_mb
```

Out[64]:

	movie	year	production_budget	domestic_gross	foreign_gross	worldwide_gross	m
425	The Lord of the Rings: The Return of the King	2003	94000000	377845905	763557436	1141403341	
135	Aquaman	2018	160000000	335061807	811832833	1146894640	
672	Minions	2015	74000000	336045770	824290403	1160336173	
47	Iron Man 3	2013	200000000	408992272	806400000	1215392272	
22	The Fate of the Furious	2017	250000000	225764765	1009081502	1234846267	
43	Incredibles 2	2018	200000000	608581744	633938967	1242520711	
134	Beauty and the Beast	2017	160000000	504014165	755185541	1259199706	
155	Frozen	2013	150000000	400738009	871731901	1272469910	
112	Jurassic World: Fallen Kingdom	2018	170000000	417719760	888053039	1305772799	
4	Star Wars Ep. VIII: The Last Jedi	2017	317000000	620181382	696540365	1316721747	
260	Harry Potter and the Deathly Hallows: Part II	2011	125000000	381193157	960500000	1341693157	
41	Black Panther	2018	200000000	700059566	648198658	1348258224	
3	Avengers: Age of Ultron	2015	330600000	459005868	944008095	1403013963	
26	The Avengers	2012	225000000	623279547	894656350	1517935897	
66	Furious 7	2015	190000000	353007020	1165715774	1518722794	
33	Jurassic World	2015	215000000	652270625	996584239	1648854864	

	movie	year	production_budget	domestic_gross	foreign_gross	worldwide_gross	m
6	Avengers: Infinity War	2018	300000000	678815482	1369318718	2048134200	
5	Star Wars Ep. VII: The Force Awakens	2015	306000000	936662225	1116648995	2053311220	
42	Titanic	1997	200000000	659363944	1548844451	2208208395	
0	Avatar	2009	425000000	760507625	2015837654	2776345279	

In [65]:
```
top_20_gross_mb[['movie', 'domestic_gross', 'foreign_gross']].plot(x='movie', k
                                                                   stacked=True

plt.xlabel('Total Gross in USD (billions)', fontsize=15)
plt.ylabel('Movie Title', fontsize=15).set_visible(False)
plt.title('Top 20 Grossing Movies',fontsize=18)
plt.legend(labels=['Domestic Gross', 'Foreign Gross'])
plt.tight_layout()
plt.savefig("top20.png", format='png',dpi=150)
plt.show()
```