Final Project Submission

Please fill out:

- Student name: **Andrew Bennett**
- Student pace: **self paced** / part time / full time
- Scheduled project review date/time:
- Instructor name: **Morgan Jones**
- Blog post URL: https://dev.to/bennettandrewm/the-weird-world-of-2p2j-temp-slug-7842137?preview=42e6157b169f72e1debad20fd30a440a51ae69baf58f452647d9c78730da6fbb27a6d2732e7

BREAKING DOWN THE BLOCK BUSTERS

Analysis of Top Grossing Movies in Theaters of Last 20 years

Author: Andrew Bennett

The business of film is to make dreams into reality, and allow audiences to escape for a few hours.

But... how do we begin in the Film Business?

There are numerous decisions to make to begin the process of making movies. A film studio has to capture the imagination of an audience but also make money. Why? So they can do it again the next day! To aid Microsoft in this voyage, this analysis will examine various factors that go into making a movie and determine if there are any trends as they relate to box office ticket sales. We'll look at the following:

- genre
- budget

- directors

These factors represent some of the simplest, more foundational decisions. Genre determines which scripts and creative teams you'll need. The budget is the money you'll need upfront to greenlight a project. And the director is the creative lead who will shepard the day-to-day movie-making and ultimately deliver the project to you and your audience.

So... where do we start?

with DATA

To solve this problem we used two well-known industry databases. The IMDB movie database (https://www.imdb.com/) is one of the most complete, publically box office available databases. It's a well known industry staple and incredibly robust. However, We also need budget information, and that's what we'll get from The Numbers (https://www.the-numbers.com/) which reports daily and weekly box office sales for movies currently in release. From there, we examine the movie titles, genres, production budgets, directors and their respective gross box office receipts. The goal is to find which of these "input" trends have an effect, if any, on box office number "output".

So... shall we get started?

First, let's import some of our standard packages.

Import standard packages

In [13]:
```
#importing standard libraries and packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

Data Preparation

First we're going to poke around the IMDB database. Luckily this database can be accessed using `sqlite3` . I'm going to pull the table names just to get a flavor of the data we have.

In [14]:
```
#import sqlite3
import sqlite3
from zipfile import ZipFile
with ZipFile('zippedData/im.db.zip', 'r') as zf:
    zf.extractall(path = 'zippedData/')
```

In [15]:
```
#create a connection
conn = sqlite3.connect("zippedData/im.db")

#execute query for table names
df = pd.read_sql("""SELECT name
                    FROM sqlite_master
                    WHERE type = 'table'
;""", conn)
```

```
df

#sqlite_master
```

Out[15]:

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

Good... These are the tables that were shown in the diagram provided in the Project Description, so I think it's safe to use the keys listed in said diagram.

Let's look for potential row names that will help with either, budget, genres, and director. `movie_basics` table looks promising, as does `directors` and `persons`. There's no budget data here (we'll address that later), but let's pull the relevant genre and director info that we do care about. I'm also going to pull `start_year` and `primary_title`.

I will do this by doing a sqlite3 query and joining some tables in the process.

In [16]:
```
#query relevant data
df1 = pd.read_sql("""SELECT DISTINCT movie_id, primary_title, start_year, genres, perso
                     FROM movie_basics
                     JOIN directors
                         USING(movie_id)
                     JOIN persons
                         USING(person_id)
;""", conn)

#reset index to primary_title
df1.set_index('primary_title', inplace = True)
df1
```

Out[16]:

primary_title	movie_id	start_year	genres	person_id	primary_name
Sunghursh	tt0063540	2013	Action,Crime,Drama	nm0712540	Harnam Singh Rawail
One Day Before the Rainy Season	tt0066787	2019	Biography,Drama	nm0002411	Mani Kaul
The Other Side of the Wind	tt0069049	2018	Drama	nm0000080	Orson Welles
Sabse Bada Sukh	tt0069204	2018	Comedy,Drama	nm0611531	Hrishikesh Mukherjee

primary_title	movie_id	start_year	genres	person_id	primary_name
The Wandering Soap Opera	tt0100275	2017	Comedy,Drama,Fantasy	nm0749914	Raoul Ruiz
...	...	...	...	...	...
Rodolpho Teóphilo - O Legado de um Pioneiro	tt9916622	2015	Documentary	nm9272491	Ana Célia de Oliveira
Dankyavar Danka	tt9916706	2013	Comedy	nm7764440	Kanchan Nayak
6 Gunn	tt9916730	2017	None	nm10538612	Kiran Gawade
Chico Albuquerque - Revelações	tt9916754	2013	Documentary	nm8349149	Vinicius Augusto Bozzo
Chico Albuquerque - Revelações	tt9916754	2013	Documentary	nm9272490	Angela Gurgel

163533 rows × 5 columns

Good news! We've got some good looking info here. Lot's of entries. And it looks like some of the movies have two entries, based on a one-to-many join with the directors table. In other words, some movies have multiple directors, and that creates multiple entries for one movie, which isn't good. So let's do some...

Data Manipulation

- with Directors

We need to remove duplicated entries from the list due to multiple directors. We can see from above that "Chico Albuquerque - Revelações" was duplicated because of credits for both Vinicius Augusto Bozzo and Angela Gurgel. This is NO BUENO.

I'm going to see how many occurrences there are with two directors vs one director. While I'm at it I'll also check for more than two.

To do this - I'm going to run a very shitty, inefficient loop and count the maximum number of director credits for any one movie, the total number of single director movies, double director movies, and 3 or more director movies.

In [17]:
```
#setting up variables
index=0;
ctr=0;
numdirectors=0;
directors=[]
one_director=0;
two_directors=0;
many_directors=0;

#for loop that iterates through each movie with an opening conditional to stop before w
for movie in df1['movie_id']:
    if index < (len(df1['movie_id'])-1):
```

```
        #If the movie is the equal to the next element in the list, we increase the ctr
        if (df1.iloc[index,0] == df1.iloc[index+1,0]):
            ctr+=1
        #If the ctr is greater than the largest number of directors on a movie (numdire
            if ctr>numdirectors:
                numdirectors=ctr
                directors.append(df1.iloc[index,0])
        #else, the movie is not equal to the next element, that update the count of one
        else:
            if ctr==0:
                 one_director+=1
            if ctr==1:
                 two_directors+=1
            if ctr>=2:
                 many_directors+=1
            ctr=0
        index+=1

print("The most directors on a single move are ", numdirectors)
print("Total entries: ", index)
print("My counter check: ",ctr)
print(one_director, "movie entries have only one director")
print(two_directors, "movie entries have two directors")
print(many_directors,"entries have three or more directors")
```

```
The most directors on a single move are  82
Total entries:  163532
My counter check:  1
124689 movie entries have only one director
13226 movie entries have two directors
2500 entries have three or more directors
```

What? 82 directors on a movie? They're handing out credits like condoms these days!

We also see that approximately 124,689 movie entries have one director, and 13,226 have two. So we can add a second director column but that's IT. There are 2500 entries with 3 or more, or less than 3%. We can truncate these entries and only inlcude the first two names on the list. Let's go ahead and add another column for direct_2 and fill it in with the second director listed.

Unfortunately, I'm going to do another, kind of shitty loop, here. I apologize in advance.

In [18]:

```
#setting up variables

df1['director_2']=None;
index=0;
ctr=0;
numdirectors=0;
directors=[]
many_director_ctr=0;

#for loop that iterates through each movie with an opening conditional to stop before w
for movie in df1['movie_id']:
    if index < (len(df1['movie_id'])-1):

#If the movie is the equal to the next element in the list, we increase the ctr. If the
        if (df1.iloc[index,0] == df1.iloc[index+1,0]):
            ctr+=1
            if ctr==1:
                df1.iloc[index,5] = df1.iloc[index+1,4]
```

```
        else:
            ctr=0
        index+=1
df1
```

Out[18]:

primary_title	movie_id	start_year	genres	person_id	primary_name	director_2
Sunghursh	tt0063540	2013	Action,Crime,Drama	nm0712540	Harnam Singh Rawail	None
One Day Before the Rainy Season	tt0066787	2019	Biography,Drama	nm0002411	Mani Kaul	None
The Other Side of the Wind	tt0069049	2018	Drama	nm0000080	Orson Welles	None
Sabse Bada Sukh	tt0069204	2018	Comedy,Drama	nm0611531	Hrishikesh Mukherjee	None
The Wandering Soap Opera	tt0100275	2017	Comedy,Drama,Fantasy	nm0749914	Raoul Ruiz	Valeria Sarmiento
...	...	...	...	...	...	...
Rodolpho Teóphilo - O Legado de um Pioneiro	tt9916622	2015	Documentary	nm9272491	Ana Célia de Oliveira	None
Dankyavar Danka	tt9916706	2013	Comedy	nm7764440	Kanchan Nayak	None
6 Gunn	tt9916730	2017	None	nm10538612	Kiran Gawade	None
Chico Albuquerque - Revelações	tt9916754	2013	Documentary	nm8349149	Vinicius Augusto Bozzo	Angela Gurgel
Chico Albuquerque - Revelações	tt9916754	2013	Documentary	nm9272490	Angela Gurgel	None

163533 rows × 6 columns

Great. So we added a second director column and populated it. We will now get rid of the duplicates.

In [19]:
```
#drop duplicates that have the same 'movie_id'
df1 = df1.drop_duplicates(subset = 'movie_id',keep = 'first')
df1
```

Out[19]:

primary_title	movie_id	start_year	genres	person_id	primary_name	director_2
Sunghursh	tt0063540	2013	Action,Crime,Drama	nm0712540	Harnam Singh Rawail	None

primary_title	movie_id	start_year	genres	person_id	primary_name	director_2
One Day Before the Rainy Season	tt0066787	2019	Biography,Drama	nm0002411	Mani Kaul	None
The Other Side of the Wind	tt0069049	2018	Drama	nm0000080	Orson Welles	None
Sabse Bada Sukh	tt0069204	2018	Comedy,Drama	nm0611531	Hrishikesh Mukherjee	None
The Wandering Soap Opera	tt0100275	2017	Comedy,Drama,Fantasy	nm0749914	Raoul Ruiz	Valeria Sarmiento
...	...	...	...	...	...	...
Kuambil Lagi Hatiku	tt9916538	2019	Drama	nm8185151	Azhar Kinoi Lubis	None
Rodolpho Teóphilo - O Legado de um Pioneiro	tt9916622	2015	Documentary	nm9272490	Angela Gurgel	Ana Célia de Oliveira
Dankyavar Danka	tt9916706	2013	Comedy	nm7764440	Kanchan Nayak	None
6 Gunn	tt9916730	2017	None	nm10538612	Kiran Gawade	None
Chico Albuquerque - Revelações	tt9916754	2013	Documentary	nm8349149	Vinicius Augusto Bozzo	Angela Gurgel

140416 rows × 6 columns

Great, we really simplified this director issue. We have the director information, and the genre. Okay. let's see if we can find some budget information and then we'll do another round of data cleaning.

Budget Data Import

As we mentioned above, a critical component of this is to have the financial information.

Luckily we have some of that information right here from The Numbers website. So let's go ahead and import that.

In [20]:
```
#import csv file and let's make the index the movie
df = pd.read_csv("Data/zippedData/tn.movie_budgets.csv.gz")
df.set_index('movie', inplace =True)
df
```

Out[20]:

movie	id	release_date	production_budget	domestic_gross	worldwide_gross
Avatar	1	Dec 18, 2009	$425,000,000	$760,507,625	$2,776,345,279

movie	id	release_date	production_budget	domestic_gross	worldwide_gross
Pirates of the Caribbean: On Stranger Tides	2	May 20, 2011	$410,600,000	$241,063,875	$1,045,663,875
Dark Phoenix	3	Jun 7, 2019	$350,000,000	$42,762,350	$149,762,350
Avengers: Age of Ultron	4	May 1, 2015	$330,600,000	$459,005,868	$1,403,013,963
Star Wars Ep. VIII: The Last Jedi	5	Dec 15, 2017	$317,000,000	$620,181,382	$1,316,721,747
...	...	...	...	...	...
Red 11	78	Dec 31, 2018	$7,000	$0	$0
Following	79	Apr 2, 1999	$6,000	$48,482	$240,495
Return to the Land of Wonders	80	Jul 13, 2005	$5,000	$1,338	$1,338
A Plague So Pleasant	81	Sep 29, 2015	$1,400	$0	$0
My Date With Drew	82	Aug 5, 2005	$1,100	$181,041	$181,041

5782 rows × 5 columns

Nice! This table was imported smoothly. So let's join these two tables together and get our MEGA table! I'll do an inner join because I only care about the movies that I can sync all three (genre, directors, and budget/financial info).

In [21]:
```
#join both tables
genre_and_budget = df.join(df1, how = "inner")
genre_and_budget
```

Out[21]:

	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_year
#Horror	16	Nov 20, 2015	$1,500,000	$0	$0	tt3526286	2015
10 Cloverfield Lane	54	Mar 11, 2016	$5,000,000	$72,082,999	$108,286,422	tt1179933	2016
10 Days in a Madhouse	48	Nov 11, 2015	$12,000,000	$14,616	$14,616	tt3453052	2015
12 Rounds	37	Mar 27, 2009	$20,000,000	$12,234,694	$17,306,648	tt3517850	2017
12 Strong	64	Jan 19, 2018	$35,000,000	$45,819,713	$71,118,378	tt1413492	2018
...	...	...	...	...	...	...	...
Zoom	26	Aug 11, 2006	$35,000,000	$11,989,328	$12,506,188	tt6117454	2016
Zoom	26	Aug 11, 2006	$35,000,000	$11,989,328	$12,506,188	tt6667868	2016

	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_year
Zootopia	57	Mar 4, 2016	$150,000,000	$341,268,248	$1,019,429,616	tt2948356	2016
Zulu	82	Dec 31, 2013	$16,000,000	$0	$1,844,228	tt2249221	2013
xXx: Return of Xander Cage	15	Jan 20, 2017	$85,000,000	$44,898,413	$345,033,359	tt1293847	2017

3727 rows × 11 columns

Okay. Now we have our MEGA table. Let's Clean Up This Data. Already we see some repititions.

Data Cleaning

First, let's reset the index and remove the duplicates.

In [22]:
```
#remove duplicates
genre_and_budget = genre_and_budget.reset_index()

#remove duplicates based on movie_id
genre_and_budget = genre_and_budget.drop_duplicates(subset = 'movie_id',keep = 'first')

#remove duplicates based on the name and release date
genre_and_budget = genre_and_budget.drop_duplicates(subset = ['index', 'release_date'],
genre_and_budget.head(20)
```

Out[22]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_y
0	#Horror	16	Nov 20, 2015	$1,500,000	$0	$0	tt3526286	2
1	10 Cloverfield Lane	54	Mar 11, 2016	$5,000,000	$72,082,999	$108,286,422	tt1179933	2
2	10 Days in a Madhouse	48	Nov 11, 2015	$12,000,000	$14,616	$14,616	tt3453052	2
3	12 Rounds	37	Mar 27, 2009	$20,000,000	$12,234,694	$17,306,648	tt3517850	2
4	12 Strong	64	Jan 19, 2018	$35,000,000	$45,819,713	$71,118,378	tt1413492	2
5	12 Years a Slave	18	Oct 18, 2013	$20,000,000	$56,671,993	$181,025,343	tt2024544	2
6	127 Hours	6	Nov 5, 2010	$18,000,000	$18,335,230	$60,217,171	tt1542344	2
7	13 Sins	51	Apr 18, 2014	$4,000,000	$9,134	$47,552	tt2059171	2
8	1982	23	Mar 1, 2016	$1,000,000	$0	$0	tt2388621	2
10	2 Guns	39	Aug 2, 2013	$61,000,000	$75,612,460	$132,493,015	tt1272878	2

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_y
11	21	39	Mar 28, 2008	$35,000,000	$81,159,365	$159,846,429	tt5097012	2
12	21 Jump Street	44	Mar 16, 2012	$42,000,000	$138,447,667	$202,812,429	tt1232829	2
13	22 Jump Street	85	Jun 13, 2014	$50,000,000	$191,719,337	$331,333,876	tt2294449	2
14	23 Blast	64	Oct 24, 2014	$1,000,000	$549,185	$549,185	tt2304459	2
15	3	85	Sep 16, 2011	$7,200,000	$59,774	$295,492	tt1517177	2
21	3 Backyards	51	Mar 11, 2011	$300,000	$39,475	$39,475	tt1314190	2
22	3 Days to Kill	64	Feb 21, 2014	$28,000,000	$30,697,999	$38,959,900	tt2172934	2
23	30 Minutes or Less	60	Aug 12, 2011	$28,000,000	$37,053,924	$40,966,716	tt1622547	2
24	300: Rise of an Empire	26	Mar 7, 2014	$110,000,000	$106,580,051	$330,780,051	tt1253863	2
25	31	85	Oct 21, 2016	$1,500,000	$779,820	$922,727	tt3835080	2

Convert Financial Info to Integers

Good, now let's convert the box office sales to integers so that we can manipulate them. I'm going to create a function called `conversion` and map each to each column. The `conversion` function will strip away "," and "$" and convert the string to and integer.

In [23]:
```
#create function to strip '$' and ','
def conversion(value):
    value=value.strip('$')
    value=value.replace(",","")
    return int(value)

#map each column with financials to the function
genre_and_budget['production_budget']=genre_and_budget['production_budget'].map(convers
genre_and_budget['domestic_gross']=genre_and_budget['domestic_gross'].map(conversion)
genre_and_budget['worldwide_gross']=genre_and_budget['worldwide_gross'].map(conversion)

genre_and_budget
```

Out[23]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
0	#Horror	16	Nov 20, 2015	1500000	0	0	tt3526286	
1	10 Cloverfield Lane	54	Mar 11, 2016	5000000	72082999	108286422	tt1179933	

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
2	10 Days in a Madhouse	48	Nov 11, 2015	12000000	14616	14616	tt3453052	
3	12 Rounds	37	Mar 27, 2009	20000000	12234694	17306648	tt3517850	
4	12 Strong	64	Jan 19, 2018	35000000	45819713	71118378	tt1413492	
...	...	...	...	...	...	...	...	
3719	Zoolander 2	64	Feb 12, 2016	50000000	28848693	55348693	tt1608290	
3720	Zoom	26	Aug 11, 2006	35000000	11989328	12506188	tt3763866	
3724	Zootopia	57	Mar 4, 2016	150000000	341268248	1019429616	tt2948356	
3725	Zulu	82	Dec 31, 2013	16000000	0	1844228	tt2249221	
3726	xXx: Return of Xander Cage	15	Jan 20, 2017	85000000	44898413	345033359	tt1293847	

2298 rows × 12 columns

Convert Genre Info to String

And now, for genre. It appears that the genre category doesn't have one entry. And, it's not a string... Okay. Let's first convert to a string. I'm going to do something similar as I did above. I'll create a function to convert genres to a string.

In [24]:
```
#create function to convert an unknown object to a string
def convert_string(value):
    new_value = str(value)
    return new_value

#map each financial series to the function
genre_and_budget['genres'] = genre_and_budget['genres'].map(convert_string)
genre_and_budget
```

Out[24]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
0	#Horror	16	Nov 20, 2015	1500000	0	0	tt3526286	
1	10 Cloverfield Lane	54	Mar 11, 2016	5000000	72082999	108286422	tt1179933	
2	10 Days in a Madhouse	48	Nov 11, 2015	12000000	14616	14616	tt3453052	

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
3	12 Rounds	37	Mar 27, 2009	20000000	12234694	17306648	tt3517850	
4	12 Strong	64	Jan 19, 2018	35000000	45819713	71118378	tt1413492	
...	...	...	...	...	...	...	...	
3719	Zoolander 2	64	Feb 12, 2016	50000000	28848693	55348693	tt1608290	
3720	Zoom	26	Aug 11, 2006	35000000	11989328	12506188	tt3763866	
3724	Zootopia	57	Mar 4, 2016	150000000	341268248	1019429616	tt2948356	
3725	Zulu	82	Dec 31, 2013	16000000	0	1844228	tt2249221	
3726	xXx: Return of Xander Cage	15	Jan 20, 2017	85000000	44898413	345033359	tt1293847	

2298 rows × 12 columns

Convert Genre Date to a Date

Let's see the oldest and newest movie to be released. First, we'll convert the date to datatime.

In [25]:
```
#convert date
genre_and_budget['release_date'] = pd.to_datetime(genre_and_budget['release_date'], inf

#sort by date
genre_and_budget.sort_values('release_date').head(70)
```

Out[25]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_
3645	Wings	85	1927-08-12	2000000	0	0	tt2284790	
1756	Mata Hari	24	1931-12-26	558000	900000	900000	tt8788464	
2430	Snow White and the Seven Dwarfs	18	1937-12-21	1488000	184925486	184925486	tt7821084	
2156	Rebecca	47	1940-03-21	1288000	6000000	6002370	tt10430534	
932	Fantasia	69	1940-11-13	2280000	83320000	83320000	tt3591950	
...	...	...	...	...	...	...	...	
2478	Stand by Me	68	1986-08-08	8000000	52287414	52287414	tt8383596	

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	start_
2077	Playing for Keeps	49	1986-10-03	35000000	2000000	2000000	tt1540128	
2879	The Golden Child	47	1986-12-12	12000000	79817937	79817937	tt3506476	
2229	River's Edge	4	1987-05-08	1900000	4600000	4600000	tt7476946	
88	Action Jackson	8	1988-02-12	7000000	20257000	20257000	tt0403935	

70 rows × 12 columns

VOILA - We have our data in a good place! Now let's clean our table up a little more.

We're going to add some additional financial info as well as eliminate some older movies. Movies before 2000 may not help us much.

Let's go ahead and slice the data and keep movies that are 2000 and later.

In [26]:
```
#slice movies > than 2000
genre_and_budget = genre_and_budget.loc[genre_and_budget['release_date'] > '2000-01-01'
genre_and_budget.sort_values('release_date')
```

Out[26]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	sta
3281	The Terrorist	48	2000-01-14	25000	195043	195043	tt9248762	
2551	Supernova	41	2000-01-14	60000000	14218868	14816494	tt10378772	
1148	Gun Shy	86	2000-02-04	10000000	1638202	1638202	tt3910736	
2427	Snow Day	51	2000-02-11	13000000	60008303	62452927	tt4163946	
3676	Wonder Boys	13	2000-02-23	35000000	19389454	33422485	tt9095294	
...	...	...	...	...	...	...	...	
3204	The Rhythm Section	8	2019-11-22	50000000	0	0	tt7134096	
2257	Rogue City	13	2019-12-31	13000000	0	0	tt10329540	
873	Eli	16	2019-12-31	11000000	0	0	tt4786638	
2154	Reagan	30	2019-12-31	25000000	0	0	tt1822382	

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	sta
499	Call of the Wild	36	2020-02-21	82000000	0	0	tt7504726	

2083 rows × 12 columns

Financial Data Manipulation

It seems like we have the numbers converted and the strings converted to strings. Let's add a few more columns to show ROI and Gross Margin. I've defined Gross Margin as the difference between how much money was grossed worldwide and how much was spent in production, or production budet.

We'll define ROI (Return on Investment) as just Worldwide Budget divided by Production Budget. So, an ROI of 1 means a movie "broke even" or made it's money back.

We should take a second to note that we have not factored in marketing or distribution costs into either of these numbers. This data is not as readily accessible on this scale.

In [27]:
```
#create new columns as mathematical operations of existing columns
genre_and_budget['ROI'] = round (genre_and_budget['worldwide_gross'] / genre_and_budget
genre_and_budget['Gross_Margin'] = round (genre_and_budget['worldwide_gross'] - genre_a
genre_and_budget
```

```
<ipython-input-27-fd757df59553>:2: 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/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  genre_and_budget['ROI'] = round (genre_and_budget['worldwide_gross'] / genre_and_budge
t['production_budget'],2)
<ipython-input-27-fd757df59553>:3: 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/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  genre_and_budget['Gross_Margin'] = round (genre_and_budget['worldwide_gross'] - genre_
and_budget['production_budget'],2)
```

Out[27]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
0	#Horror	16	2015-11-20	1500000	0	0	tt3526286	
1	10 Cloverfield Lane	54	2016-03-11	5000000	72082999	108286422	tt1179933	
2	10 Days in a Madhouse	48	2015-11-11	12000000	14616	14616	tt3453052	
3	12 Rounds	37	2009-03-27	20000000	12234694	17306648	tt3517850	
4	12 Strong	64	2018-01-19	35000000	45819713	71118378	tt1413492	

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
...	...	...	...	...	...	...	...	
3719	Zoolander 2	64	2016-02-12	50000000	28848693	55348693	tt1608290	
3720	Zoom	26	2006-08-11	35000000	11989328	12506188	tt3763866	
3724	Zootopia	57	2016-03-04	150000000	341268248	1019429616	tt2948356	
3725	Zulu	82	2013-12-31	16000000	0	1844228	tt2249221	
3726	xXx: Return of Xander Cage	15	2017-01-20	85000000	44898413	345033359	tt1293847	

2083 rows × 14 columns

Yes!!!! We have mega table. We're very close to analyzing the data, we just need to separate the genre information so we can individualize the categories.

Question 1 - Which genre of movies do the best at the box office?

Okay, to answer this question, we need to separate all of the genres into their own columns, so each movie entry will have every genre category. What do I mean? Let's say a moive is a mix of `Action`, `Comedy`, and `Adventure`. That entry then would have a 1 in the `Action` column, a 1 in the `Comedy` column, and a 1 in the `Adventure` column. The other columns would have a 0, meaning that genre is not represented in this movie.

Sounds good right! Let's do some genre cleaning.

GENRE TITLE CLEANING

First, we have to determine how many unique genre categories exist. And then we can create separate columns for each of the genres. This way the genres will be easily categorized later.

In [28]:
```
#using list comprehension, make a list of every genre word using a lambda function and
genre_list = [item for genres in list(genre_and_budget['genres'].map(lambda x: x.split(

# then make the list unique.
list(set(genre_list))
```

Out[28]:
```
['Drama',
 'Western',
 'History',
 'Sport',
 'Music',
 'Documentary',
 'Action',
 'Fantasy',
 'Animation',
 'War',
```

```
 'Sci-Fi',
 'Musical',
 'Adventure',
 'Comedy',
 'Romance',
 'Biography',
 'Mystery',
 'Horror',
 'None',
 'Crime',
 'Family',
 'Thriller']
```

Great! although carefully looking at the list we see a category `None` as well as two similar categories, `Music` and `Musical`. Let's go ahead and clean those up.

In [29]:
```
#create a list from the set of 'genre_list'
genre_list = list(set(genre_list))
genre_list.remove('Musical')
genre_list.remove('None')
genre_list
```

Out[29]:
```
['Drama',
 'Western',
 'History',
 'Sport',
 'Music',
 'Documentary',
 'Action',
 'Fantasy',
 'Animation',
 'War',
 'Sci-Fi',
 'Adventure',
 'Comedy',
 'Romance',
 'Biography',
 'Mystery',
 'Horror',
 'Crime',
 'Family',
 'Thriller']
```

In [30]:
```
#sort the list
genre_list.sort()
genre_list
```

Out[30]:
```
['Action',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Family',
 'Fantasy',
 'History',
 'Horror',
 'Music',
 'Mystery',
 'Romance',
 'Sci-Fi',
```

```
 'Sport',
 'Thriller',
 'War',
 'Western']
```

Now that looks good! We're ready to add the genre categories as column names with a simple 0,1 to indicate whether that genre category is present in our original genre category.

In [31]:
```
#write for loop to iterate through each genre in the list we just created
for genre in genre_list:

#for each genre, create a new column with that name and create lambda function to assig
    genre_and_budget[genre]=genre_and_budget['genres'].map(lambda x: 1 if genre in x el

genre_and_budget
```

```
<ipython-input-31-94cd1ec2c572>:5: 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/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  genre_and_budget[genre]=genre_and_budget['genres'].map(lambda x: 1 if genre in x else
0)
```

Out[31]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
0	#Horror	16	2015-11-20	1500000	0	0	tt3526286	
1	10 Cloverfield Lane	54	2016-03-11	5000000	72082999	108286422	tt1179933	
2	10 Days in a Madhouse	48	2015-11-11	12000000	14616	14616	tt3453052	
3	12 Rounds	37	2009-03-27	20000000	12234694	17306648	tt3517850	
4	12 Strong	64	2018-01-19	35000000	45819713	71118378	tt1413492	
...	...	...	...	...	...	...	...	
3719	Zoolander 2	64	2016-02-12	50000000	28848693	55348693	tt1608290	
3720	Zoom	26	2006-08-11	35000000	11989328	12506188	tt3763866	
3724	Zootopia	57	2016-03-04	150000000	341268248	1019429616	tt2948356	
3725	Zulu	82	2013-12-31	16000000	0	1844228	tt2249221	
3726	xXx: Return of Xander Cage	15	2017-01-20	85000000	44898413	345033359	tt1293847	

2083 rows × 34 columns

Data Analysis

Question 1: Genre, doesn't it have any affect on financial output

Okay, let's start manipulating data and looking to answer some questions. First, how about `Gross_Margin` . Let's create a table with just `Gross_Margin` and the genre categories

In [32]:
```
#slice table to create only columns we need
gross_margin = genre_and_budget.loc[:,'Gross_Margin':'Western']

#sort columns in ascending order
gross_margin = gross_margin.sort_values('Gross_Margin', ascending = False)
gross_margin
```

Out[32]:

	Gross_Margin	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama
258	2351345279	0	0	0	0	0	0	0	0
260	1748134200	1	1	0	0	0	0	0	0
1519	1433854864	1	1	0	0	0	0	0	0
1057	1328722794	1	0	0	0	0	1	0	0
2634	1292935897	1	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...
499	-82000000	0	1	1	0	0	0	0	0
431	-90000000	1	0	0	0	0	1	0	0
1775	-106900000	1	1	0	0	1	0	0	0
1750	-110450242	0	1	1	0	0	0	0	0
707	-200237650	1	1	0	0	0	0	0	0

2083 rows × 21 columns

Let's just jump right to it. Which genre if any correlates most strongly with high margins.

Let's start with genre and box office receipts

In [33]:
```
#create a table correlating all of the variables.
positivecorr = gross_margin.corr().drop('Gross_Margin')

#Now, just show the correlations associate with 'Gross_Margin' in ascending order
question_1a_corr = positivecorr['Gross_Margin'].sort_values(ascending = True)
question_1a_corr
```

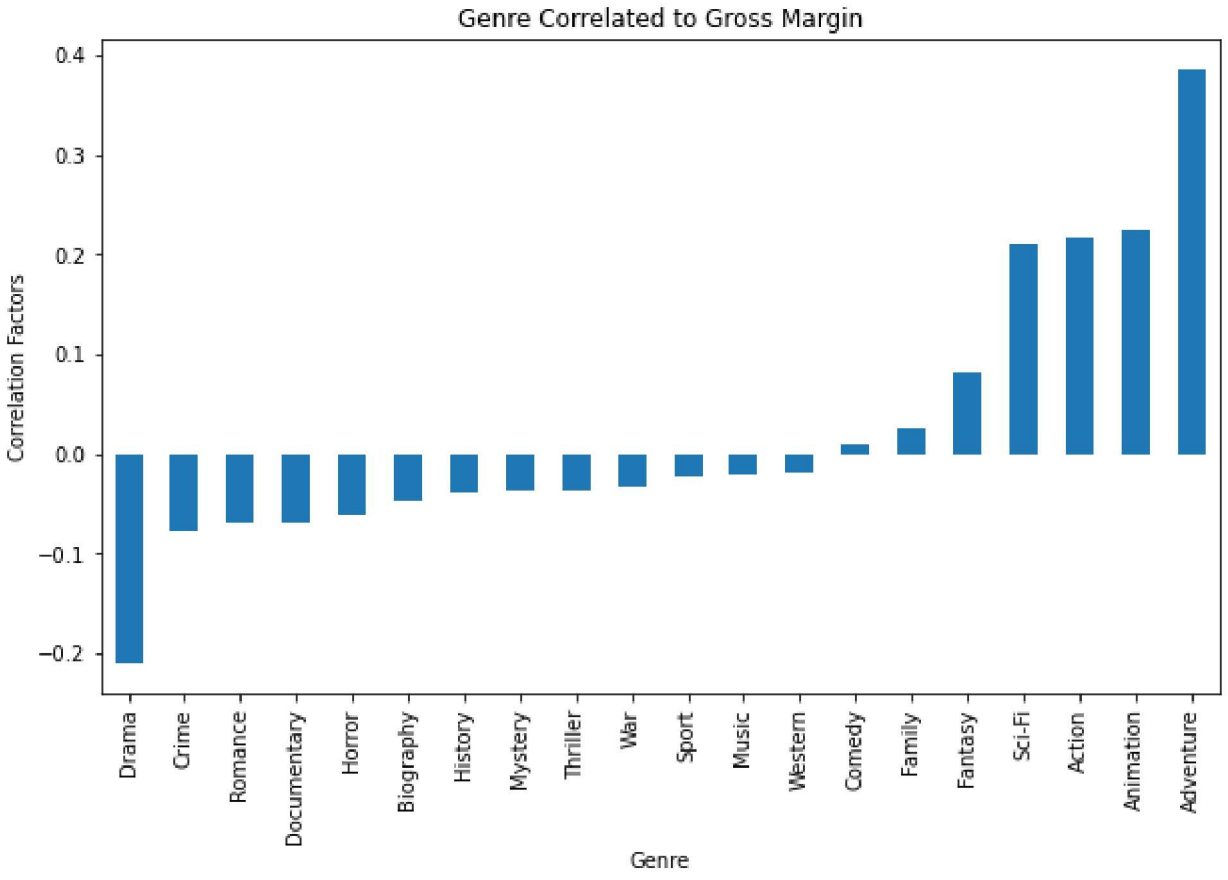
Out[33]:
```
Drama          -0.210502
Crime          -0.077882
Romance        -0.069531
Documentary    -0.068286
Horror         -0.061838
Biography      -0.047755
History        -0.039188
Mystery        -0.037551
Thriller       -0.036598
War            -0.033543
```

```
Sport         -0.023850
Music         -0.021382
Western       -0.018811
Comedy         0.008325
Family         0.024850
Fantasy        0.081739
Sci-Fi         0.209867
Action         0.216303
Animation      0.223826
Adventure      0.386086
Name: Gross_Margin, dtype: float64
```

This is interesting, there appears to be a correlation trend here. With the "adventure" genre showing the strongest correlation. Let's look at this graphed below.

In [34]:
```
#plot a vertical bar graph

fig, ax = plt.subplots(figsize=(10,6))

ax = question_1a_corr.plot.bar()

ax.set_title('Genre Correlated to Gross Margin')
ax.set_xlabel('Genre')
ax.set_ylabel('Correlation Factors')
```

Out[34]:
```
Text(0, 0.5, 'Correlation Factors')
```



Visualization 1 - Vertical Bar Graph of Genre Correlation with ROI.

Recommendation 1 - The Adventure Genre shows the highest correlation to Gross Margin. Perhaps we should make adventure

movies.

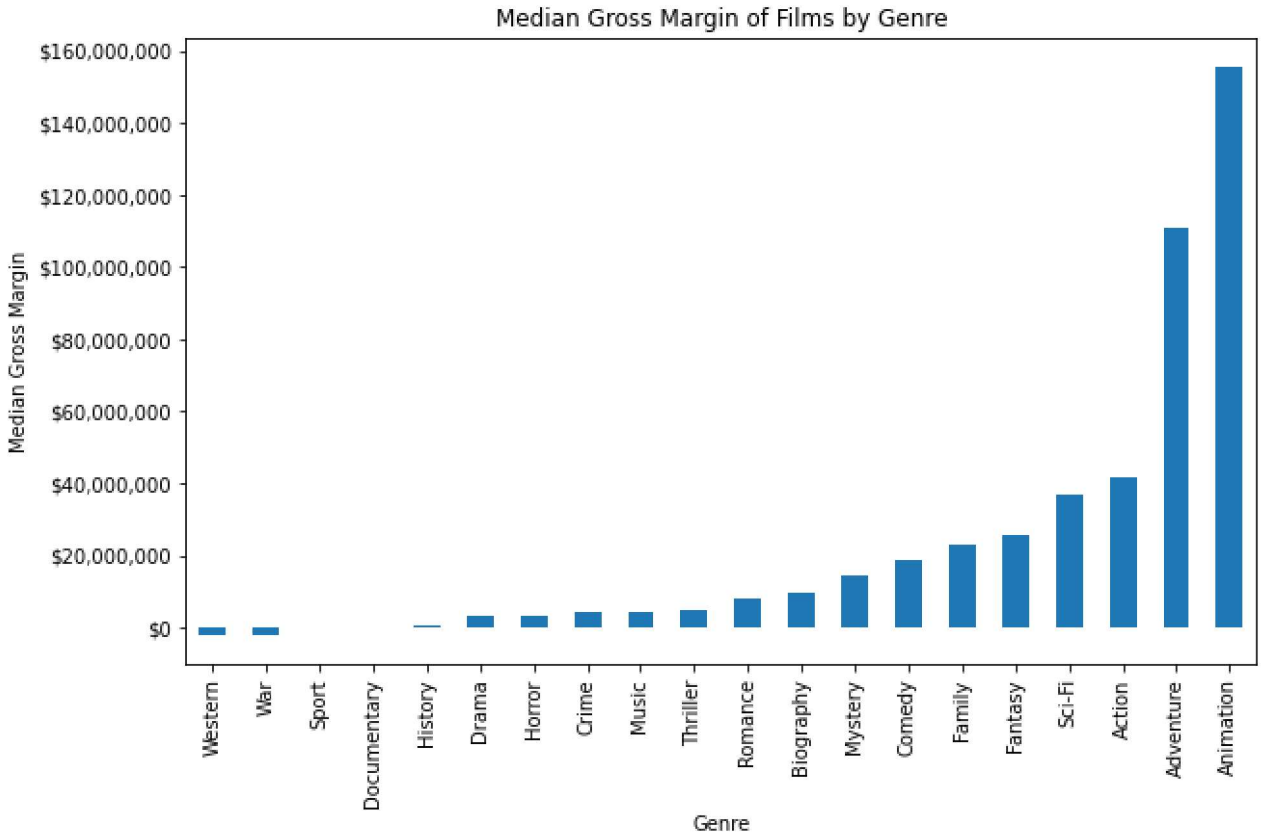
Let's look at the median Gross_Margin values are for each of these genres to get a sense of maybe the "typical movie". We look at median instead of mean because it's less influenced by outliers (meaning super hits and super flops).

In [35]:
```
#slice a table to get Gross_Margin and all of the genres
question_1a_me = genre_and_budget.loc[:,'Gross_Margin':'Western']

#replace the 1s in the list with the actual Gross_Margin values.
for genre in genre_list:
    question_1a_me[genre] = question_1a_me[genre] * question_1a_me['Gross_Margin']

#replace the 0s with Nan.
question_1a_me.replace(0, np.nan, inplace = True)

#now create find the median values by each genre
question_1a_me = question_1a_me.drop('Gross_Margin', axis=1)
question_1a_m = question_1a_me.median().sort_values()
question_1a_m
```

Out[35]:
```
Western         -2141346.5
War             -1973745.0
Sport              -37395.0
Documentary        255538.0
History            721474.5
Drama             3239723.0
Horror            3327198.0
Crime             4106975.0
Music             4629950.5
Thriller          5139730.0
Romance           7955193.0
Biography         9629774.0
Mystery          14610760.0
Comedy           18831384.0
Family           22938391.0
Fantasy          25474658.0
Sci-Fi           37079671.0
Action           41647574.0
Adventure       110859554.0
Animation       155707267.0
dtype: float64
```

Wow, we've seen some trends here. Let's see how this looks visually.

In [36]:
```
#plot a vertical bar graph

fig, ax = plt.subplots(figsize=(10,6))

ax = question_1a_m.plot.bar()

ax.set_title('Median Gross Margin of Films by Genre')
ax.set_xlabel('Genre')
ax.set_ylabel('Median Gross Margin')

ax.yaxis.set_major_formatter('${x:,.0f}')
```



So we see that both Adventure and Animation have the highest median values for Worldwide Gross Margin, but again, we don't factor in ROI here. what about return on investment.

Let's look at ROI.

In [37]:
```
#Let's create a table with just ROI and the genres we need.
gross_ROI = genre_and_budget.drop('Gross_Margin', axis = 1)
gross_ROI = gross_ROI.loc[:,'ROI':'Western']

#Let's sort that table by ROI.
gross_ROI = gross_ROI.sort_values('ROI', ascending = False)
gross_ROI
```

Out[37]:

	ROI	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family
2826	416.56	0	0	0	0	0	0	0	0	0
2758	101.76	0	0	0	0	0	0	0	0	0
1430	66.58	0	0	0	0	0	0	0	0	0
3501	64.36	0	0	0	0	0	0	0	0	0
2014	59.17	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
2823	0.00	0	0	0	0	0	0	0	1	0
2822	0.00	0	0	0	1	1	0	1	0	0
822	0.00	1	0	0	0	1	1	0	0	0

	ROI	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family
827	0.00	0	0	0	0	1	0	0	0	0
0	0.00	0	0	0	0	0	1	0	1	0

2083 rows × 21 columns

Okay, now let's see if there's any correlation between genre and ROIs.

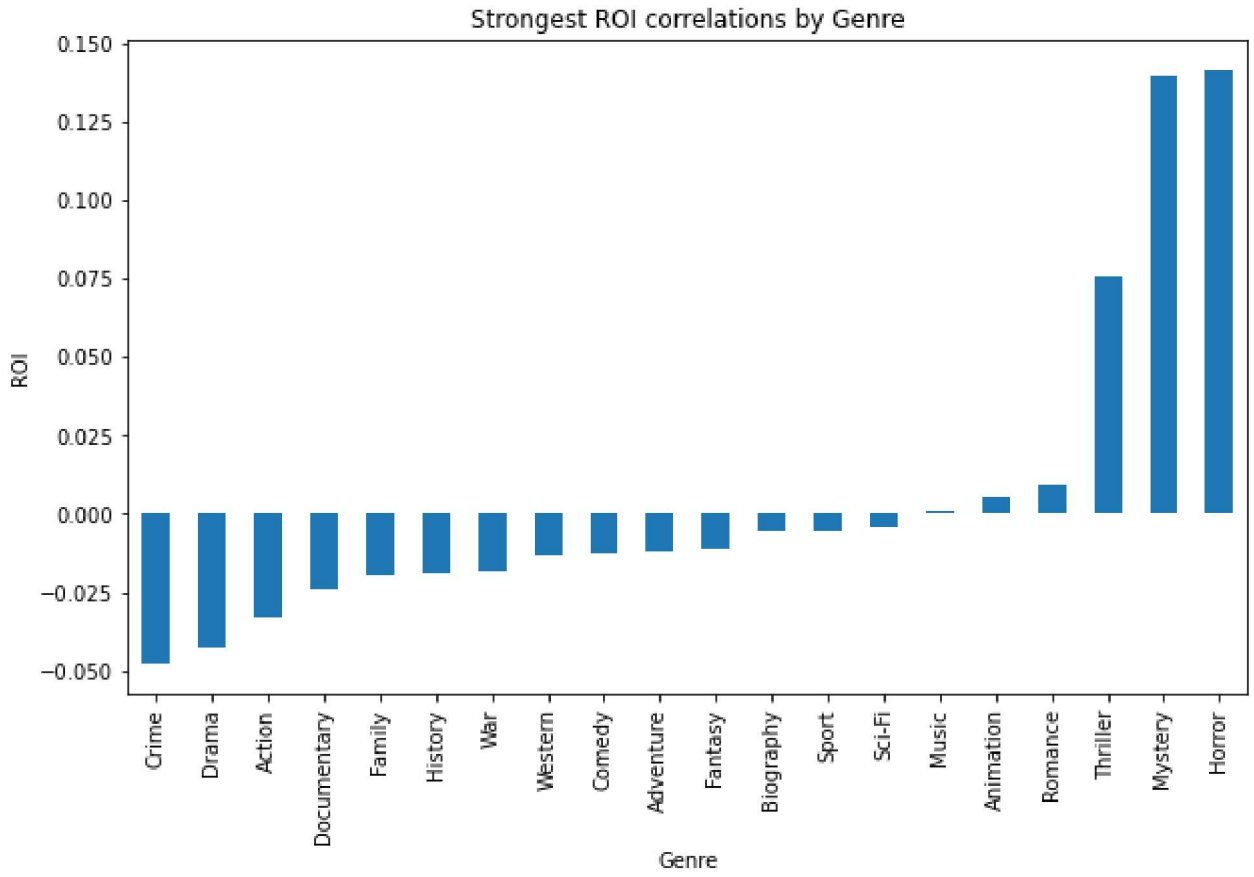
In [38]:
```
#Let's find the correlation between different genres and the median ROI
positivecorr = gross_ROI.corr().drop('ROI')
question_1b_corr = positivecorr['ROI']
question_1b_corr=question_1b_corr.sort_values()
question_1b_corr
```

Out[38]:
```
Crime         -0.047778
Drama         -0.042426
Action        -0.032856
Documentary   -0.023879
Family        -0.019821
History       -0.019144
War           -0.018381
Western       -0.013374
Comedy        -0.012668
Adventure     -0.011977
Fantasy       -0.011309
Biography     -0.005761
Sport         -0.005192
Sci-Fi        -0.003994
Music          0.000690
Animation      0.005306
Romance        0.008940
Thriller       0.075420
Mystery        0.139505
Horror         0.141594
Name: ROI, dtype: float64
```

Interesting, let's look at this visually.

In [39]:
```
#plot a vertical bar graph
fig, ax = plt.subplots(figsize=(10,6))

ax = question_1b_corr.plot.bar()

ax.set_title('Strongest ROI correlations by Genre')
ax.set_xlabel('Genre')
ax.set_ylabel('ROI')
```

Out[39]:
```
Text(0, 0.5, 'ROI')
```



So we see some trends here. While Mystery and Horror do have the highest correlation, no genre seems to have a STRONG correlation. Mystery and Horror could correlate based with ROI because they have smaller budgets, but we would have to investigate further.

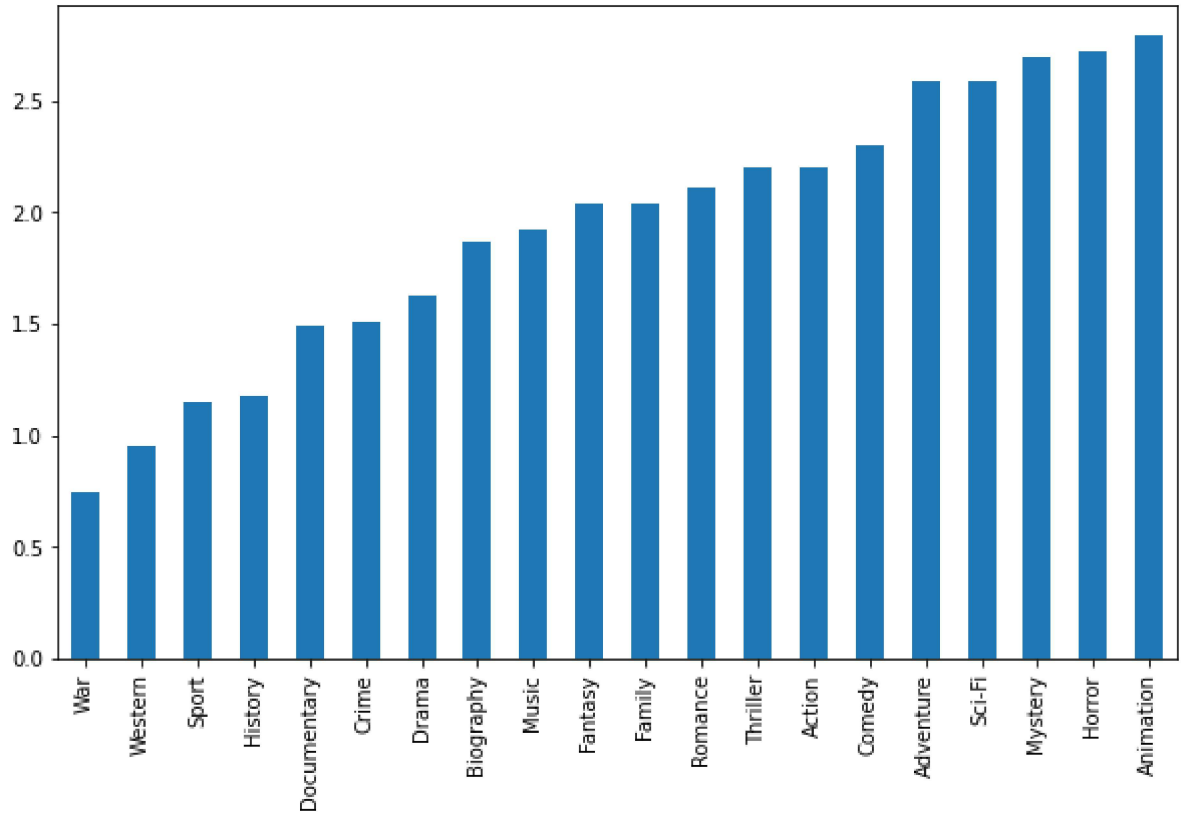
Now let's look at the median ROI from each genre. To do this, I'm going to replace the 1s in each of the genre categories with the corresponding ROI for that movie. So for instance, if Movie A had an ROI of 2.3, and was an Action, Adventure. The columns for Action and Adventure would now have a 2.3 in them instead of a 1. But, we might end up with skewed data with all of the 0s. To solve this, will convert the 0s to Nans. The median calculation will ignore those and we'll find the median ROI for each genre.

In [40]:
```
#slice a table of only ROI and genres
question_1b_me = genre_and_budget.drop('Gross_Margin', axis = 1)
question_1b_me = question_1b_me.loc[:,'ROI':'Western']

#replace the 1s in the list with the ROIs. We'll do this by mathematical operation for
for genre in genre_list:
    question_1b_me[genre] = question_1b_me[genre] * question_1b_me['ROI']

#replace the 0s with Nan to find the median
question_1b_me.replace(0, np.nan, inplace = True)

#now find the median
question_1b_me = question_1b_me.drop('ROI', axis=1)
question_1b_m = question_1b_me.median().sort_values()
question_1b_m
```

Out[40]:
```
War            0.750
Western        0.950
```

```
Sport          1.150
History        1.180
Documentary    1.490
Crime          1.510
Drama          1.630
Biography      1.870
Music          1.925
Fantasy        2.035
Family         2.035
Romance        2.115
Thriller       2.200
Action         2.205
Comedy         2.300
Adventure      2.585
Sci-Fi         2.590
Mystery        2.695
Horror         2.720
Animation      2.790
dtype: float64
```

In [41]:
```
#plot a vertical bar graph
plt.figure(figsize=(10,6))
question_1b_m.plot.bar()
plt.show()
```



So we see that Animation has the top ROI, with Horre, Mystery, Sci-FI, and Adventure close behind. What's going on here? Well, let's talk budget. Maybe we can find some answers there.

Question 2: Budget - How much money should we spend?

Budget is always a BIG QUESTION. But how do we answer this? In the last section we looked at the affects of genre on ROI and Gross_Margin. We found the strongest correlation between the Adventure Genre and Gross_Margin.

Well, maybe we can see if there's any relationship between budgets and ROI.

Let's see if we can group budget into different categories, and then make some judgements on ROI. So first, let's assemble the budgets with ROI and Gross_Margin

In [42]:
```
#slice a table of relevant financial data
moneydf = genre_and_budget.loc[:,'production_budget':'Gross_Margin']
moneydf['budget'] = moneydf['production_budget']
moneydf = moneydf.loc[:,'ROI':'budget']

#sort values in ascending order
moneydf.sort_values('ROI', ascending = False).head(30)
```

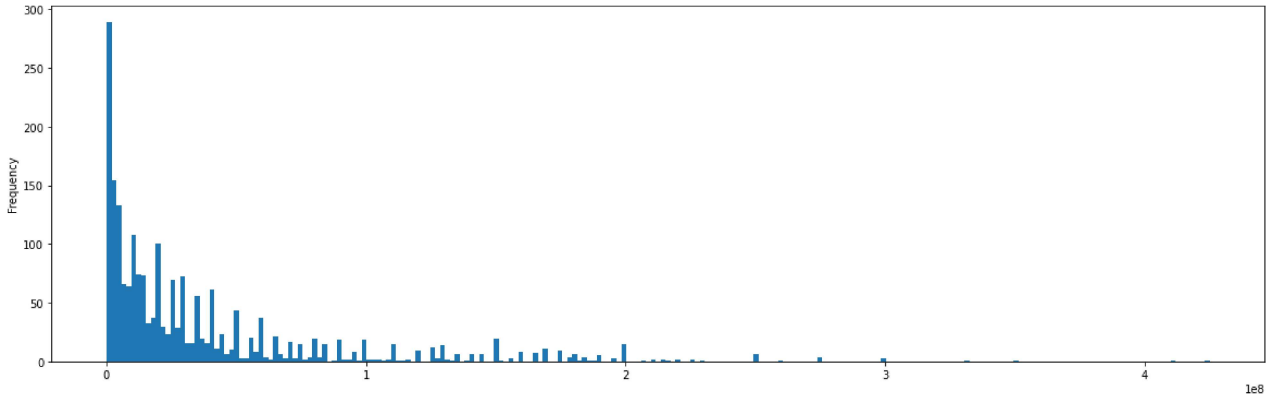
Out[42]:

	ROI	Gross_Margin	budget
2826	416.56	41556474	100000
2758	101.76	100759490	1000000
1430	66.58	98370886	1500000
3501	64.36	63364198	1000000
2014	59.17	174512032	3000000
2465	55.79	273964806	5000000
1075	51.07	250367951	5000000
1833	43.50	63745512	1500000
546	42.41	41411721	1000000
2015	41.41	202039844	5000000
210	39.52	250362920	6500000
3016	38.98	68365900	1800000
699	31.02	285154618	9500000
3172	30.42	88266581	3000000
1642	29.76	143806510	5000000
2392	29.24	84727807	3000000
2016	28.56	137817992	5000000
43	27.70	2669782	100000
3446	27.18	91627344	3500000
3704	26.89	25887177	1000000
3707	25.75	2970593	120000
2797	25.60	295166834	12000000
369	25.48	318266710	13000000
1186	25.00	120010260	5000000
3583	24.66	70975239	3000000
1726	24.36	163549753	7000000

	ROI	Gross_Margin	budget
45	23.97	57422558	2500000
356	21.85	104253745	5000000
1	21.66	103286422	5000000
1579	21.32	406351163	20000000

Wow! We see some high ROIs. Let's see that distribution of budget!

In [43]:
```
#let's plot a distribution (histogram) of all the movies made by budget
plt.figure(figsize=(20,6))
moneydf['budget'].plot.hist(bins=225)
plt.show()
```



Whoa. A lot of small budget movies here. Let's see what the median budget is.

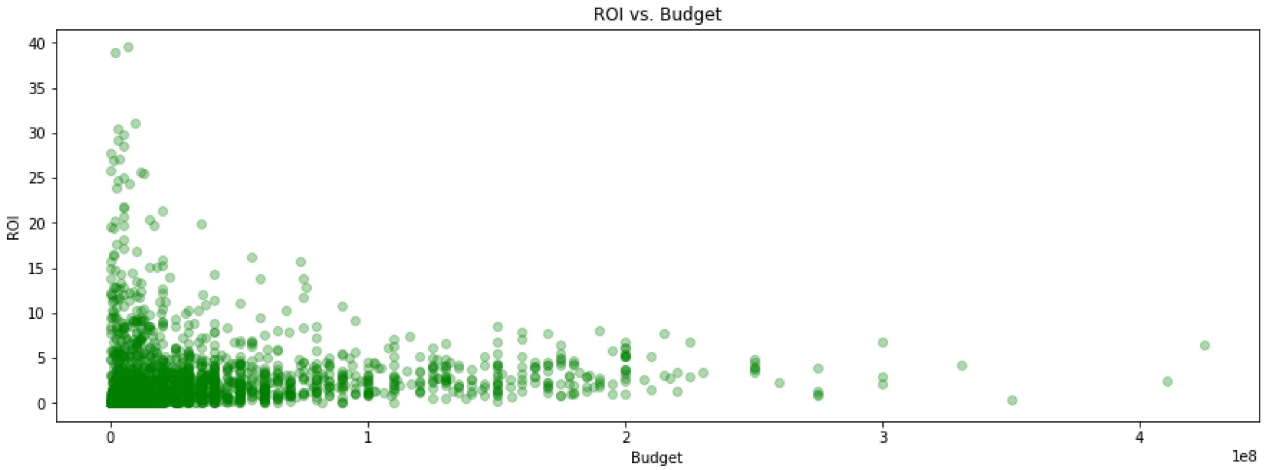
In [44]:
```
#let's find the the median and mean
print(moneydf['budget'].median())
print(moneydf['budget'].mean())
print(moneydf['budget'].std())
```

```
19500000.0
37652578.94383101
51572465.35430997
```

So the median budget is 19.5M, while the mean is ~38M. This shows a skew to the left, which makes sense considering the number of movies made for the cheap. Also, we can look at the graph above and see a sharp skew to the left.

Let's do a quick scatter plot of all movies of ROI vs budget. This may give us a general sense of which movies do the best.

In [45]:
```
#let's limit the movies with ROIs under 40 for the sake of the visuals
moneyplot = moneydf.loc[moneydf['ROI'] < 40]

#let's do a scatter plot with x-axis is budget and y value is ROI
fig, ax = plt.subplots(figsize=(15,5))
ax.scatter(moneyplot['budget'],moneyplot['ROI'],alpha=0.3, color="green")

#let's set some of the titles.
ax.set_title("ROI vs. Budget")
ax.set_xlabel("Budget")
ax.set_ylabel("ROI");
```



Wow, okay, there are many movies made on a small budget. It also looks like ROI doesn't increase with a higher budget. This makes sense intuitively, smaller budget movies need to sell less tickets in order to make their money back and show a return.

Let's try to look more specifically at the budgets and see if we can find where we see the most success. Let's put the budgets into 6 different categories, because that seems like a manageable number. So for 6 numbers, we need the 16th, 33rd, 50th, 66th, and 83rd percentile to segment our groups.

I'm going to utilize the np.percentile tool in my budgets to find these numbers.

In [46]:
```
#initialize variables for list
numlist=[]
num = 6

#find 16th, 33rd, 50th, 66th, and 83rd percentile and add to the list
numlist.append(np.percentile(moneydf['budget'],(100/num)*1))
numlist.append(np.percentile(moneydf['budget'],(100/num)*2))
numlist.append(np.percentile(moneydf['budget'],(100/num)*3))
numlist.append(np.percentile(moneydf['budget'],(100/num)*4))
numlist.append(np.percentile(moneydf['budget'],(100/num)*5))

numlist
```

Out[46]:
```
[2100000.0000000116, 9000000.0, 19500000.0, 35000000.0, 68000000.00000022]
```

Okay, so we have 2.1M, 9 M, 19.5M, 35M, and 68M as our numbers. For simplicity, I'm going to use 2, 10, 20, 35, and 70M. This will help for readibility. I'm going to put thse into categories like, micro, small, etc...

Additionally, perhaps we should look at the movies which MADE money, with an ROI greater than 1. We might glean insight into what successful movies are.

Now, let's make those categories.

In [47]:
```
#slice only the data without outliers
mademoneydf = moneydf.loc[(moneydf['ROI'] >= 1) & (moneydf['ROI'] <= 35)]

#categorize our budgets into nice categories
Micro = mademoneydf.loc[mademoneydf['budget'] < 2000000]
```

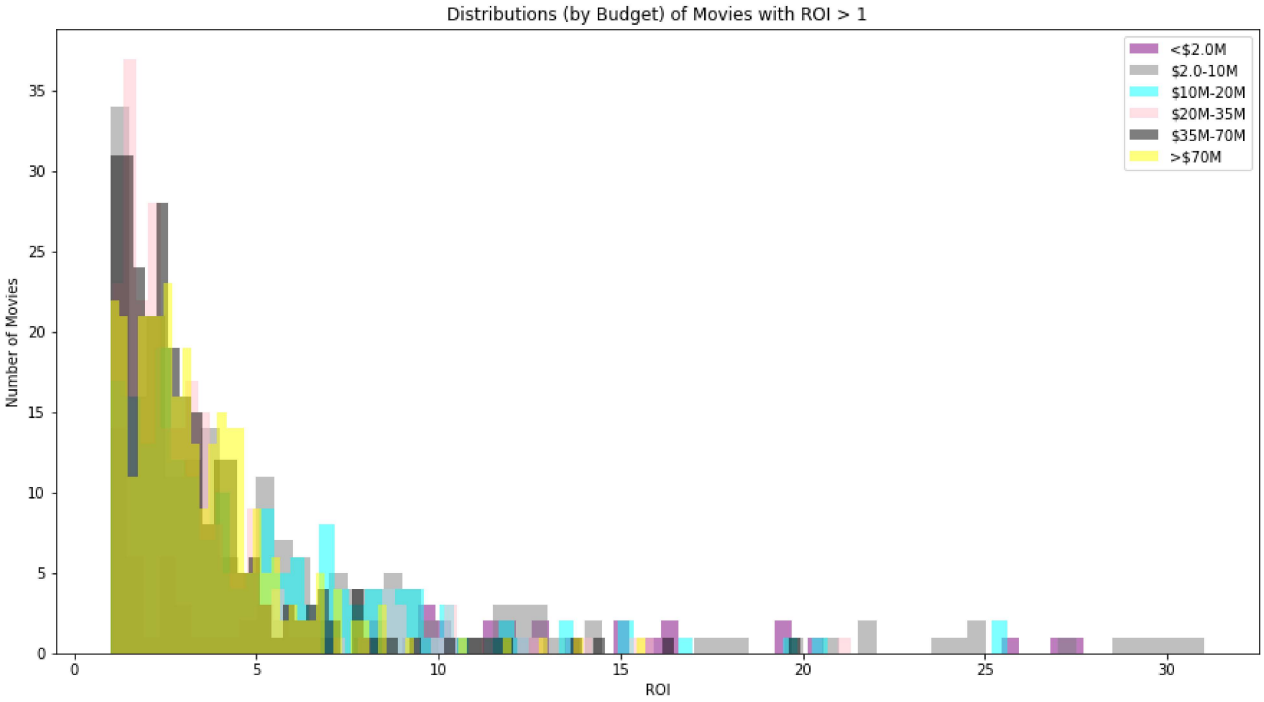
```
Small = mademoneydf.loc[(mademoneydf['budget'] >= 2000000) & (mademoneydf['budget'] < 1
Mid = mademoneydf.loc[(mademoneydf['budget'] >= 10000000) & (mademoneydf['budget'] < 20
Large = mademoneydf.loc[(mademoneydf['budget'] >= 20000000) & (mademoneydf['budget'] <
Macro = mademoneydf.loc[(mademoneydf['budget'] >= 35000000) & (mademoneydf['budget'] <
Enormous = mademoneydf.loc[mademoneydf['budget'] >= 70000000]
```

Let's try and make a histrogram of all movies in the budget categories and what their ROI is.

In [48]:

```
#create a histrogram plot
fig, ax = plt.subplots(figsize=(15,8))

# Create custom bins so all are on the same scale
bins = 60

# Plot six histograms, with reduced opacity (alpha) so we can see them overlapping
ax.hist(
    x=Micro["ROI"],
    label="<$2.0M",
    bins=bins,
    color="purple",
    alpha=0.5
)
ax.hist(
    x=Small["ROI"],
    label="$2.0-10M",
    bins=bins,
    color="gray",
    alpha=0.5
)
ax.hist(
    x=Mid["ROI"],
    label="$10M-20M",
    bins=bins,
    color="cyan",
    alpha=0.5
)

ax.hist(
    x=Large["ROI"],
    label="$20M-35M",
    bins=bins,
    color="pink",
    alpha=0.5
)

ax.hist(
    x=Macro["ROI"],
    label="$35M-70M",
    bins=bins,
    color="black",
    alpha=0.5
)

ax.hist(
    x=Enormous["ROI"],
    label=">$70M",
    bins=bins,
    color="yellow",
    alpha=0.5
)
```

```
# Customize labels
ax.set_title("Distributions (by Budget) of Movies with ROI > 1")
ax.set_xlabel("ROI")
ax.set_ylabel("Number of Movies")
ax.legend();

#ax.yaxis.set_major_formatter('${x:,.0f}')
#ax.xaxis.set_major_locator(ticker.MultipleLocator(500))
#ax.xaxis.set_minor_locator(ticker.MultipleLocator(100))
```



Okay, no clear answer here. It seems that there are a lot of movies that made money, at least from what we can see here. Maybe there's a different way to look at this.

Let's see how many movies make money (ROI>=1) vs how many lose money, for each category. To get more even distributions, I'm going to alter the numbers so they resembled the percentiles we saw previously, (ie 2M, 9M, 20M, 35M, 70M)

In [49]:
```
#categorize our budgets into nice categories, and count the numbers that are greater or
Count = pd.DataFrame()

Micro = moneydf.loc[moneydf['budget'] < 2000000]
Count['< $2M'] = Micro.groupby(Micro['ROI']>1).count()['ROI']

Small = moneydf.loc[(moneydf['budget'] >= 2000000) & (moneydf['budget'] < 9000000)]
Count['$2M-9M'] = Small.groupby(Small['ROI']>1).count()['ROI']

Mid = moneydf.loc[(moneydf['budget'] >= 9000000) & (moneydf['budget'] < 20000000)]
Count['$9M-20M'] = Mid.groupby(Mid['ROI']>1).count()['ROI']

Large = moneydf.loc[(moneydf['budget'] >= 20000000) & (moneydf['budget'] < 35000000)]
Count['$20M-35M'] = Large.groupby(Large['ROI']>1).count()['ROI']

Macro = moneydf.loc[(moneydf['budget'] >= 35000000) & (moneydf['budget'] < 70000000)]
Count['$35M-70M'] = Macro.groupby(Macro['ROI']>1).count()['ROI']
```

```
Enormous = moneydf.loc[moneydf['budget'] >= 70000000]
Count['>$70M'] = Enormous.groupby(Enormous['ROI']>1).count()['ROI']

Count
```

Out[49]:

ROI	< $2M	$2M-9M	$9M-20M	$20M-35M	$35M-70M	>$70M
False	218	200	159	111	90	36
True	78	187	201	233	262	308

WOW! We've got a trend. We see that as the budget grows, the number of movies making money does as well! Let's chart this to see how this looks!

First, let's clean up this table so it's ready to plot.

In [50]:
```
Count['ROI>1'] = ['Red', 'Black']
Count = Count.set_index('ROI>1')
PlotCount=Count.transpose()
PlotCount
```

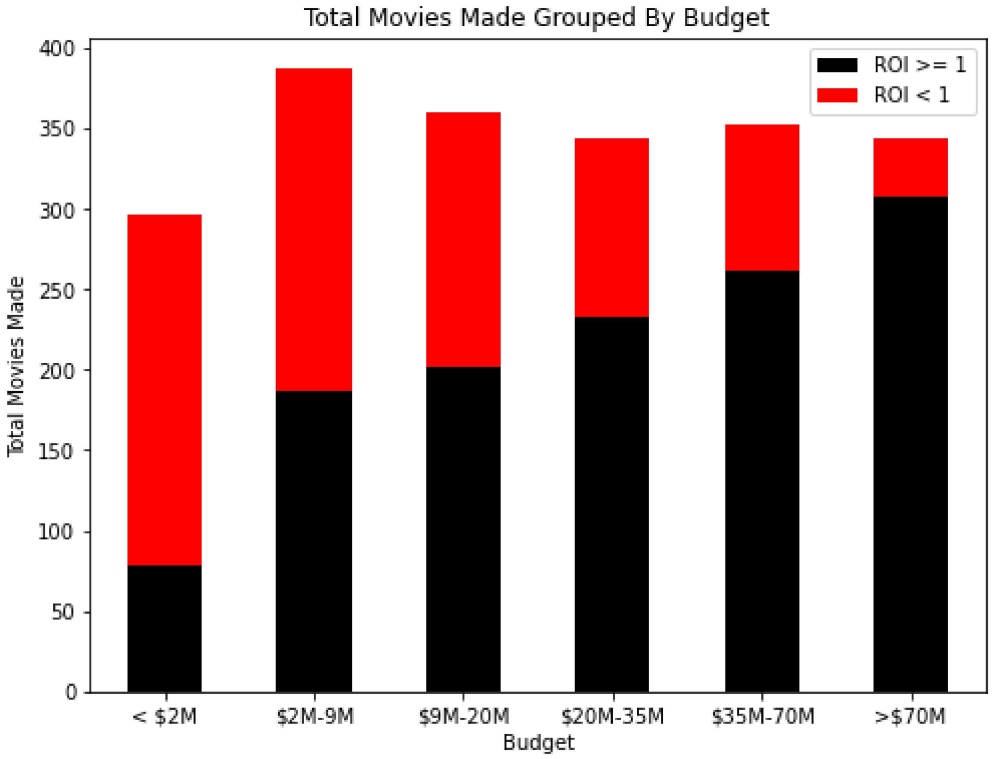
Out[50]:

ROI>1	Red	Black
< $2M	218	78
$2M-9M	200	187
$9M-20M	159	201
$20M-35M	111	233
$35M-70M	90	262
>$70M	36	308

Now, we have a very clean table with which to plot

In [51]:
```
#plot a vertical bar graph
fig, ax = plt.subplots(figsize=(8,6))

ax = PlotCount['Black'].plot.bar(color='black', label = 'ROI >= 1')
ax = PlotCount['Red'].plot.bar(bottom = PlotCount['Black'], color ='red', label = 'ROI

ax.set_title('Total Movies Made Grouped By Budget')
ax.set_xlabel('Budget')
ax.set_ylabel('Total Movies Made')
ax.legend()
plt.xticks(rotation=None)
```

Out[51]:
```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, '< $2M'),
  Text(1, 0, '$2M-9M'),
  Text(2, 0, '$9M-20M'),
  Text(3, 0, '$20M-35M'),
  Text(4, 0, '$35M-70M'),
  Text(5, 0, '>$70M')])
```



WOW! That's a great trend. We can see that as the budgets for the movies grow, the number of money making movies did too!

Visualization 2: The number of Movies Making Money Increases with Budget

Recommendation 2: Make Movies with budgets over 70M. They more likely to have an ROI > 1

Okay, so we know we want a big budget action movie, but who should we hire? Big Budget Action Movies are not easy to do. We need someone who's experienced. So now let's figure out our director

Question 3: The Director

So... we did a nice job in the early phases of getting some of that director information into our dataframe. And, we already know what our budget is, and the genre. So why don't we find a list of directors who make the big budget action blockbuster

To start, let's make a table of the director information

In [52]:
```
#slice a new dataframe based on director info
directordf = genre_and_budget.loc[:,'production_budget':'Western']
directordf['budget'] = directordf['production_budget']
directorcountdf = directordf.loc[:,'primary_name':'budget']

#do a count of all of the directors here
directorcountdf['primary_name'].value_counts()
```

Out[52]:
```
Steven Spielberg      7
David Gordon Green    6
Ridley Scott          6
```

```
Steven Soderbergh     6
Clint Eastwood        6
                     ..
Ian Sharp             1
Max Joseph            1
Chan-wook Park        1
Coke Daniels          1
David Wain            1
Name: primary_name, Length: 1581, dtype: int64
```

Okay. We've got some familiary names here. Now let's find the best directors for our budget range ($30-45M). Maybe... some directors who have made more than 1 movie in this budget range.

In [53]:
```
#slice the movies from our director dataframe with $70M budget
bigbudgetaction = genre_and_budget.loc[(genre_and_budget['production_budget'] >= 700000
bigbudgetaction

#Now, let's slice a new dataframe and count the totals by name
bigbudgetdirectorROI = bigbudgetaction.loc[:,'primary_name':'ROI']
bigbudgetdirectorROI = bigbudgetdirectorROI.drop('director_2', axis = 1)
directorlist = bigbudgetdirectorROI['primary_name'].value_counts()

#Now, let's slice this dataframe and keep only the directors where the count is greater
directorlist = directorlist.loc[directorlist > 1]
dlist = directorlist.reset_index()
director_list = list(dlist['index'].sort_values())
director_list
#Now, let's make a list of these directors and print it out.
```

Out[53]:
```
['Andrew Stanton',
 'Anthony Russo',
 'Brad Bird',
 'Brad Peyton',
 'Bryan Singer',
 'Byron Howard',
 'Carlos Saldanha',
 'Chris Renaud',
 'Chris Wedge',
 'Christopher Nolan',
 'David Yates',
 'Dean DeBlois',
 'Eric Darnell',
 'Francis Lawrence',
 'Gareth Edwards',
 'George Miller',
 'Gore Verbinski',
 'Guy Ritchie',
 'James Mangold',
 'Jennifer Yuh Nelson',
 'Jon M. Chu',
 'Jonathan Liebesman',
 'Joss Whedon',
 'Kyle Balda',
 'Lee Unkrich',
 'M. Night Shyamalan',
 'Marc Forster',
 'Marc Webb',
 'Matt Reeves',
 'Matthew Vaughn',
 'Michael Bay',
 'Mike Mitchell',
 'Paul W.S. Anderson',
 'Peter Jackson',
```

```
'Peyton Reed',
'Raja Gosnell',
'Ridley Scott',
'Ron Howard',
'Sam Mendes',
'Shane Black',
'Steve Martino',
'Steven Spielberg',
'Tim Burton',
'Zack Snyder']
```

Now that we have this list, let's make a list of the average (mean) `ROI` for these directors. I'm picking `ROI` because we don't know EXACTLY what the budget is for the movies these directors made. The difference of making 50M profit on a 70M movie budget or an 400M budget should be factored. So let's do it.

To do this, we're going to slice a new datafram containing the `ROI` of only the directors from our list AND only the movies that they directed from our budget range. Once we have this list we can find the mean of the `ROI`.

In [54]:
```
#make a function to determine if the director is on our list
def bdirector (director):
    for directors in director_list:
        if (directors == director):
            return True

#slice new dataframe that checks our director's name and budget of movie
ourdirectors = directordf.loc[directordf['primary_name'].map(bdirector) & (directordf['

#create list of bigbudget directors grouped by 'primary_name' and sorted by the mean of
prime_directors = ourdirectors.groupby('primary_name')['ROI'].mean().sort_values(ascend
prime_directors
```

Out[54]:
```
primary_name
Ron Howard              1.113333
Gore Verbinski          1.385000
Paul W.S. Anderson      1.455000
Chris Wedge             1.560000
George Miller           1.820000
M. Night Shyamalan      2.030000
Zack Snyder             2.246000
Bryan Singer            2.600000
Steven Spielberg        2.602500
Marc Forster            2.715000
Guy Ritchie             2.830000
James Mangold           2.915000
Tim Burton              2.916667
Jonathan Liebesman      2.960000
Ridley Scott            3.042000
Andrew Stanton          3.070000
Jon M. Chu              3.165000
David Yates             3.220000
Brad Peyton             3.270000
Matthew Vaughn          3.483333
Marc Webb               3.490000
Matt Reeves             3.700000
Brad Bird               3.715000
Dean DeBlois            3.756667
Peter Jackson           3.896667
Carlos Saldanha         3.940000
Shane Black             3.945000
```

```
Eric Darnell            3.965000
Jennifer Yuh Nelson     4.065000
Christopher Nolan       4.132500
Raja Gosnell            4.145000
Sam Mendes              4.240000
Gareth Edwards          4.280000
Mike Mitchell           4.380000
Peyton Reed             4.390000
Byron Howard            4.530000
Michael Bay             4.600000
Lee Unkrich             4.950000
Anthony Russo           5.196667
Joss Whedon             5.495000
Francis Lawrence        5.613333
Steve Martino           5.895000
Chris Renaud            7.125000
Kyle Balda             14.740000
Name: ROI, dtype: float64
```

Perfect! Except. Who is Kyle Balda and what movies did he direct?

In [56]:
```
#slicing a nwe dataframe to find Kyle Balda
balda = genre_and_budget.loc[(genre_and_budget['primary_name'] == 'Kyle Balda')]
balda
```

Out[56]:

	index	id	release_date	production_budget	domestic_gross	worldwide_gross	movie_id	star
754	Despicable Me 3	30	2017-06-30	75000000	264624300	1034727750	tt3469046	
1794	Minions	73	2015-07-10	74000000	336045770	1160336173	tt2293640	

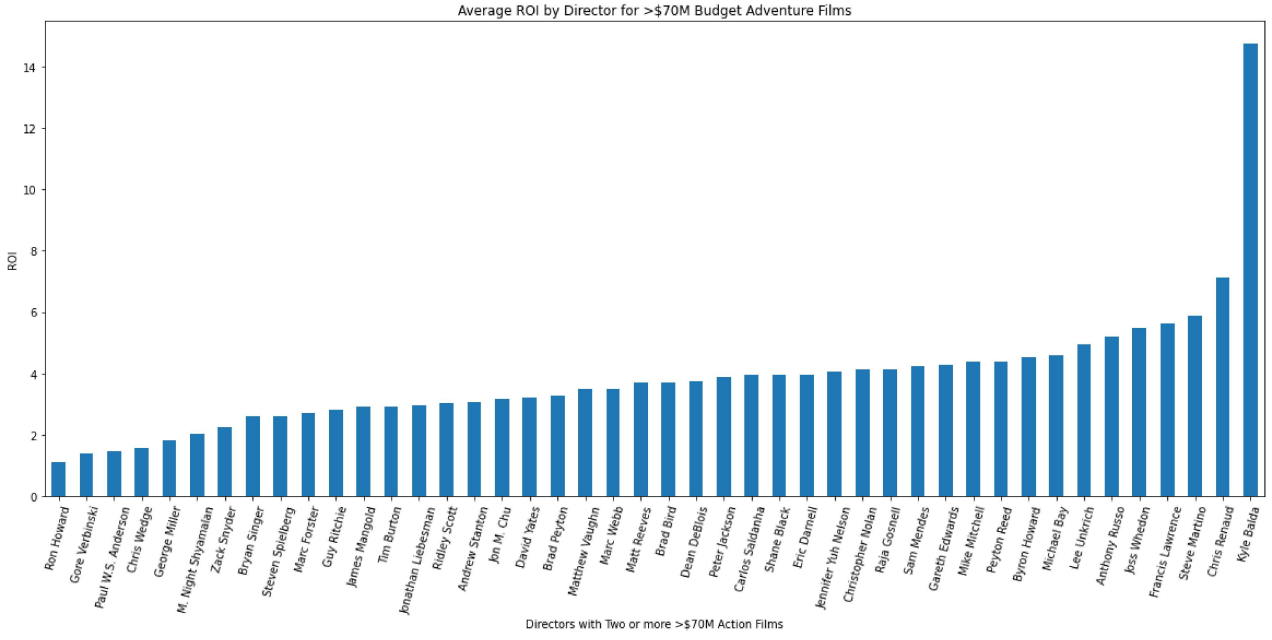
2 rows × 34 columns

Okay, yeah, I've head of those movies. Kyle Balda is real! SO... let's see these movies below.

In [68]:
```
fig, ax = plt.subplots(figsize=(20,8))

ax = prime_directors.plot.bar()

ax.set_title('Average ROI by Director for >$70M Budget Adventure Films')
ax.set_ylabel('ROI')
ax.set_xlabel('Directors with Two or more >$70M Action Films')
plt.xticks(rotation=75)

#ax.xaxis.set_major_formatter('${x:,.0f}')
```

Out[68]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43]),
 [Text(0, 0, 'Ron Howard'),
  Text(1, 0, 'Gore Verbinski'),
  Text(2, 0, 'Paul W.S. Anderson'),
  Text(3, 0, 'Chris Wedge'),
  Text(4, 0, 'George Miller'),
  Text(5, 0, 'M. Night Shyamalan'),
  Text(6, 0, 'Zack Snyder'),
  Text(7, 0, 'Bryan Singer'),
  Text(8, 0, 'Steven Spielberg'),
  Text(9, 0, 'Marc Forster'),
```

```
Text(10, 0, 'Guy Ritchie'),
Text(11, 0, 'James Mangold'),
Text(12, 0, 'Tim Burton'),
Text(13, 0, 'Jonathan Liebesman'),
Text(14, 0, 'Ridley Scott'),
Text(15, 0, 'Andrew Stanton'),
Text(16, 0, 'Jon M. Chu'),
Text(17, 0, 'David Yates'),
Text(18, 0, 'Brad Peyton'),
Text(19, 0, 'Matthew Vaughn'),
Text(20, 0, 'Marc Webb'),
Text(21, 0, 'Matt Reeves'),
Text(22, 0, 'Brad Bird'),
Text(23, 0, 'Dean DeBlois'),
Text(24, 0, 'Peter Jackson'),
Text(25, 0, 'Carlos Saldanha'),
Text(26, 0, 'Shane Black'),
Text(27, 0, 'Eric Darnell'),
Text(28, 0, 'Jennifer Yuh Nelson'),
Text(29, 0, 'Christopher Nolan'),
Text(30, 0, 'Raja Gosnell'),
Text(31, 0, 'Sam Mendes'),
Text(32, 0, 'Gareth Edwards'),
Text(33, 0, 'Mike Mitchell'),
Text(34, 0, 'Peyton Reed'),
Text(35, 0, 'Byron Howard'),
Text(36, 0, 'Michael Bay'),
Text(37, 0, 'Lee Unkrich'),
Text(38, 0, 'Anthony Russo'),
Text(39, 0, 'Joss Whedon'),
Text(40, 0, 'Francis Lawrence'),
Text(41, 0, 'Steve Martino'),
Text(42, 0, 'Chris Renaud'),
Text(43, 0, 'Kyle Balda')])
```



Perfect, we can see the best directors in the Adventure genre here.

Visualization 3: Directors with Largest Average ROIs with Big Budget Adventure Films

Recommendation 3: Hire the Directors at the top of this chart - Kyle Balda, Chris Renaud, Steve Martino, Francis Lawrence, Joss Whedon, or Anthony Russo. Their big budget movies have an average ROI of 5.

The visual above shows many directors who have a proven track record in this budget range. Let's give some of the names at the top of the list a call and see what they're working on!

SUMMARY

While there's much more work to done to launch the movie studio, we were able to glean some important insights from our analysis. The data from the past 20 years shows that are many big $(<70M) budget Action movies that take in more money at the box office than they cost to make. Microsoft should target these directors (provided in the visual) on this list that have a track record of making profitable movies.

In [ ]: