Final Project Submission

Please fill out:

- Student name: David Schenck
- Student pace: Flex
- Scheduled project review date/time: June 2, 2023, 12:30 pm Mountain Time
- Instructor name: Morgan Jones
- Blog post URL: https://wordpress.com/home/daviddata24.wordpress.com (https://wordpress.com/home/daviddata24.wordpress.com)

Phase 1 Project

In this hypothetical scenario, Microsoft wants to start a movie studio and has asked me to look at historical movie data to try to determine what works and what does not.

The two metrics I will use to measure success are profit and viewer ratings. Microsoft is going to want their new studio to be profitable, so my recommendations will be based on what has made movies profitable in the past. In addition to making money, I want to make sure that the new studio's movies are well-received, particularly the first few releases, because a positive first impression will make people more likely to want to see future movies. That is why I am also looking at trends for movies with both high and low ratings.

The three main factors I will look at are budget, genre, and star-power. Budget: Microsoft will want to know how much capital to invest into the new studio in order to realistically make a large profit.

Genre: I will look through movies released in the US market to determine which genres tend to make the most money and receive the highest praise.

Star-power: I want to determine if having recognizable actors/directors involved in a movie affects profit or ratings. This is the most challenging factor to tease out of the data because I need to come up with a metric for how much "star-power" a person has as a function of time throughout their career.

In [1]:
```
#Import necessary packages
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
plt.rcParams['figure.figsize'] = (10, 10)
plt.style.use('ggplot')
```

Data Exploration

I am going to start by just reading in the data and seeing what info is included.

In [2]:
```
#Define a path to where all the data is kept
dir_path = 'C:/Users/david/Documents/Flatiron/phase_1/Phase1-Microsoft-Movie-Studio/'
```

Box Office Mojo

The Box Office Mojo data includes title, studio, domestic and foreign gross, and the year. There are 3387 movies included.

In [3]:
```
#Box Office Mojo
df = pd.read_csv(dir_path+'Data/bom.movie_gross.csv')
```

In [4]:
```
#Look at what is in the data
df.info()
```

```
<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
```

In [5]:
```
#Look at first 10 entries
df.head(10)
```

Out[5]:

	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
5	The Twilight Saga: Eclipse	Sum.	300500000.0	398000000	2010
6	Iron Man 2	Par.	312400000.0	311500000	2010
7	Tangled	BV	200800000.0	391000000	2010
8	Despicable Me	Uni.	251500000.0	291600000	2010
9	How to Train Your Dragon	P/DW	217600000.0	277300000	2010

In [6]:
```
#Find earliest movie
df.year.min()
```

Out[6]: 2010

Rotten Tomatoes Information

This dataset includes information about genre, rating, director, writer, release date, box office, runtime, and studio. Strangely, it doesn't actually include the title of the movie. I will look to see if that is in a different dataset.

In [7]:
```
#Rotten Tomatoes Information
df = pd.read_csv(dir_path+'Data/rt.movie_info.tsv',sep='\t')
```

In [8]:
```
#See what is in the dataset
df.info()
```

```
<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 [9]:
```
#Look at first 10 entries
df.head(10)
```

Out[9]:

	id	synopsis	rating	genre	director	writer	theater_date	dvd_date	currency	box_office
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971	Sep 25, 2001	NaN	NaN
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	Jan 1, 2013	$	600,000
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996	Apr 18, 2000	NaN	NaN
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994	Aug 27, 1997	NaN	NaN
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN	NaN	NaN	NaN
5	8	The year is 1942. As the Allies unite overseas...	PG	Drama|Kids and Family	Jay Russell	Gail Gilchriest	Mar 3, 2000	Jul 11, 2000	NaN	NaN
6	10	Some cast and crew from NBC's highly acclaimed...	PG-13	Comedy	Jake Kasdan	Mike White	Jan 11, 2002	Jun 18, 2002	$	41,032,915
7	13	Stewart Kane, an Irishman living in the Austra...	R	Drama	Ray Lawrence	Raymond Carver|Beatrix Christian	Apr 27, 2006	Oct 2, 2007	$	224,114
8	14	"Love Ranch" is a bittersweet love story that ...	R	Drama	Taylor Hackford	Mark Jacobson	Jun 30, 2010	Nov 9, 2010	$	134,904
9	15	When a diamond expedition in the Congo is lost...	PG-13	Action and Adventure|Mystery and Suspense|Scie...	Frank Marshall	John Patrick Shanley	Jun 9, 1995	Jul 27, 1999	NaN	NaN

Rotten Tomatoes Reviews

This dataset includes reviews of each movie. Again, the movie title is not included. The ID allows for this dataset to be joined with the previous Rotten Tomatoes dataset which also does not contain movie titles. Based on my current plans, I do not expect to use this dataset.

In [10]:
```
#Rotten Tomatoes Reviews
df = pd.read_csv(dir_path+'Data/rt.reviews.tsv', sep='\t', encoding = 'latin1')
```

In [11]:

```
#See what is in the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54432 entries, 0 to 54431
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   id          54432 non-null  int64 
 1   review      48869 non-null  object
 2   rating      40915 non-null  object
 3   fresh       54432 non-null  object
 4   critic      51710 non-null  object
 5   top_critic  54432 non-null  int64 
 6   publisher   54123 non-null  object
 7   date        54432 non-null  object
dtypes: int64(2), object(6)
memory usage: 3.3+ MB
```

In [12]:

```
#Look at first 10 entries
df.head(10)
```

Out[12]:

	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
5	3	... Cronenberg's Cosmopolis expresses somethin...	NaN	fresh	Michelle Orange	0	Capital New York	September 11, 2017
6	3	Quickly grows repetitive and tiresome, meander...	C	rotten	Eric D. Snider	0	EricDSnider.com	July 17, 2013
7	3	Cronenberg is not a director to be daunted by ...	2/5	rotten	Matt Kelemen	0	Las Vegas CityLife	April 21, 2013
8	3	Cronenberg's cold, exacting precision and emot...	NaN	fresh	Sean Axmaker	0	Parallax View	March 24, 2013
9	3	Over and above its topical urgency or the bit ...	NaN	fresh	Kong Rithdee	0	Bangkok Post	March 4, 2013

TheMovieDB

This dataset contains genre, title, popularity, release date, vote average, and vote count. I will need to do a little research to figure out how some of these columns are calculated. What is popularity? Does more votes automatically mean the movie was well received?

After looking into the data more closely, some movies show up multiple times with identical information. I am not sure why, but if I use this dataset, I should try to remove the duplicates. There are a little over 1000 duplicates.

Popularity: I looked this up and it seems like popularity is just a measure of how much people are engaging with the movie on the website. It also looks like something that changes day-to-day, so chances are the popularity score in the table is just from the day the data was pulled. I do not actually think popularity is a useful metric for my analysis.

I tried merging this dataset with the dataset from TheNumbers because I wanted to see how hard it would be. What I found is that a little fewer than 100 movies couldn't be crosslisted by title, and when I checked the titles to see why, it actually turns out TheMovieDB data just doesn't have many of them (including big releases). I think this is just because of when the data was pulled. For example, Captain Marvel is not in this dataset, but there is currently a webpage for it on TheMovieDB. This does make me less likely to utilize this dataset. That is okay. The main thing I wanted from it was user ratings, but IMDB has that, too.

In [13]:
```
#TheMovieDB
df = pd.read_csv(dir_path+'Data/tmdb.movies.csv')
```

In [15]:
```
#See what is in the dataset
df.info()
```

```
<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
```

In [16]:
```
#Drop duplicates and see how many remain
df.drop_duplicates(subset='id').info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 25497 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         25497 non-null  int64  
 1   genre_ids          25497 non-null  object 
 2   id                 25497 non-null  int64  
 3   original_language  25497 non-null  object 
 4   original_title     25497 non-null  object 
 5   popularity         25497 non-null  float64
 6   release_date       25497 non-null  object 
 7   title              25497 non-null  object 
 8   vote_average       25497 non-null  float64
 9   vote_count         25497 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.1+ MB
```

TheNumbers

This dataset includes release date, title, budget, and both domestic and worldwide gross. Production budget is one of the things I want to use in my analysis, so I will use this dataset.

In [17]:
```
#TheNumbers
df = pd.read_csv(dir_path+'Data/tn.movie_budgets.csv')
```

In [18]:
```
#See what is in the dataset
df.info()
```

```
<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
```

In [19]:
```
#Look at first 10 entries
df.head(10)
```

Out[19]:

	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
5	6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	$306,000,000	$936,662,225	$2,053,311,220
6	7	Apr 27, 2018	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200
7	8	May 24, 2007	Pirates of the Caribbean: At Worldâ□□s End	$300,000,000	$309,420,425	$963,420,425
8	9	Nov 17, 2017	Justice League	$300,000,000	$229,024,295	$655,945,209
9	10	Nov 6, 2015	Spectre	$300,000,000	$200,074,175	$879,620,923

IMDB

There are 8 tables in this database:

1. persons: This includes 606,648 people, but many are from the distant past or they are not known to the American movie audience. I will want to only keep those who are featured in major movies released in the US.
2. principals: This lists the principal actors and directors of each movie.
3. known_for: I looked up what this means specifically, and it only includes 4 movies per person. This is not particularly helpful for what I want to do.
4. directors: Lists the movies directed by each person.
5. writers: Lists the movies written by each person.
6. movie_basics: For each movie, it lists the title, year, and genres. I will want to filter the 146,144 movies down to just major movies released in the US.

7. movie_ratings: Lists average rating of each movie and the number of votes. I do not know where the ratings are coming from (are they professional critics or just user ratings). My guess is that it has to just be user ratings because some movies have hundreds of thousands of votes.
8. movie_akas: Is the only table that includes the regions in which the movies were released. This will be needed to find movies released in the US.

In [20]:

```
#IMDB
conn = sqlite3.connect(dir_path+'im.db/im.db')
```

In [21]:

```
#persons
pd.read_sql("""
SELECT *
  FROM persons;
""", conn)
```

Out[21]:

	person_id	primary_name	birth_year	death_year	primary_profession
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_department
2	nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decorator
...	...	...	...	...	...
606643	nm9990381	Susan Grobes	NaN	NaN	actress
606644	nm9990690	Joo Yeon So	NaN	NaN	actress
606645	nm9991320	Madeline Smith	NaN	NaN	actress
606646	nm9991786	Michelle Modigliani	NaN	NaN	producer
606647	nm9993380	Pegasus Envoyé	NaN	NaN	director,actor,writer

606648 rows × 5 columns

In [22]:
```
#movie_basics
pd.read_sql("""
SELECT *
  FROM movie_basics;
""", conn)
```

Out[22]:

	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
...	...	...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019	123.0	Drama
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015	NaN	Documentary
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013	NaN	Comedy
146142	tt9916730	6 Gunn	6 Gunn	2017	116.0	None
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013	NaN	Documentary

146144 rows × 6 columns

In [23]:
```
#movie_ratings
pd.read_sql("""
SELECT *
  FROM movie_ratings;
""", conn)
```

Out[23]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21
...	...	...	...
73851	tt9805820	8.1	25
73852	tt9844256	7.5	24
73853	tt9851050	4.7	14
73854	tt9886934	7.0	5
73855	tt9894098	6.3	128

73856 rows × 3 columns

Cleaning

I am going to use two sources of data:

1. TheNumbers: Includes budget and domestic/worldwide gross information
2. IMDb: Includes information about rating, actors, and directors.

The IMDB dataset, which has the most information, has unique IDs for each movie. Unfortunately, those IDs can only be used to merge tables within the dataset, not with other non-IMDB datasets. I can try using movie titles to combine, but that leads to issues because some movies were released under multiple different titles. There are three other issues when trying to merge

1. The merge method is case sensitive. This is the easiest to fix because I can just make all the titles lower case.
2. In the TheNumbers data, the original csv document with the data has some strange artifacts in place of special characters. Characters like apostrophes, dashes, ellipses, and accented letters do not show up correctly. There are a small enough number of these (less than 100) that they can be fixed manually.
3. The hardest problem to fix is just that some titles differ slightly. For example, a movie can have a colon in it in one table, but

In [24]:
```
#Read in IMDb movie_basics
df_basics = pd.read_sql("""
SELECT *
  FROM movie_basics;
""", conn)
```

```
I read in a cleaned version of TheNumbers data. All of the strange artifacts have been removed from
the titles and other edits have been made to make sure it matches up with the titles in IMDb.
```

In [25]:
```
df = pd.read_csv(dir_path+'Data/tn.movie_budgets_clean.csv')
#Change date to datetime format
df.release_date = pd.to_datetime(df.release_date, format = '%b %d, %Y')
#Change money from string to float
df.production_budget = df.production_budget.replace('[\$,]', '', regex=True).astype(float)/1000000
df.domestic_gross = df.domestic_gross.replace('[\$,]', '', regex=True).astype(float)/1000000
df.worldwide_gross = df.worldwide_gross.replace('[\$,]', '', regex=True).astype(float)/1000000
#Create new column for profit
df['profit'] = df.worldwide_gross - df.production_budget
#Only keep data since 2010
df_10_hasgross = df[df.release_date >= '2010-01-01'][df.domestic_gross > 0]
```

```
<ipython-input-25-d2249b7c05f7>:11: UserWarning: Boolean Series key will be reindexed to match DataFr
ame index.
  df_10_hasgross = df[df.release_date >= '2010-01-01'][df.domestic_gross > 0]
```

In [26]:
```
#Change the titles so it is all lower case. This will help when merging.
df_basics['primary_title'] = df_basics['primary_title'].str.lower()
df_10_hasgross['movie'] = df_10_hasgross['movie'].str.lower()
```

In [27]:
```
#Merge the data from TheNumbers with the IMDB basics table
df_merge = df_10_hasgross.merge(df_basics, left_on = 'movie', right_on = 'primary_title', how = 'left'
```

Below is a list of all the movies in TheNumbers that I couldn't match with something in IMDb. It mostly includes foreign movies and low budget movies. I had to manually change quite a few movie titles so they would match, but it was at least reasonable to do by hand.

In [28]:
```
#Unmatched movies
df_merge[df_merge.primary_title.isna()]
```

Out[28]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	profit	movie_id	primary
400	2	2010-04-22	oceans	80.000	19.422319	86.787530	6.787530	NaN	
450	49	2010-05-28	agora	70.000	0.619423	38.992292	-31.007708	NaN	
488	20	2015-09-04	tian jiang xiong shi	65.000	0.074070	122.519874	57.519874	NaN	
497	42	2016-02-19	mei ren yu	60.720	3.229457	554.516671	493.796671	NaN	
538	51	2016-02-05	xi you ji zhi sun wu kong san da bai gu jing	60.000	0.709982	194.058503	134.058503	NaN	
...	...	...	...	...	...	...	...	...	
2383	39	2010-06-25	kynodontas	0.323	0.110248	1.373407	1.050407	NaN	

The total number of rows in the merged table is 2439. However, when I use drop_duplicates on the movie title, there are only 1785 rows. This means there are a lot of duplicates. This happens because multiple different movies in the IMDb dataset have the same title. I need to remove the duplicates and determine which movies should actually be matched.

In [29]:
```
#Drop duplicates
df_merge.drop_duplicates(subset = 'movie')
```

Out[29]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	profit	movie_id	primary_title
0	2	2011-05-20	pirates of the caribbean: on stranger tides	410.600	241.063875	1045.663875	635.063875	tt1298650	pirates of the caribbean: on stranger tides
1	3	2019-06-07	dark phoenix	350.000	42.762350	149.762350	-200.237650	tt6565702	dark phoenix
2	4	2015-05-01	avengers: age of ultron	330.600	459.005868	1403.013963	1072.413963	tt2395427	avengers: age of ultron
3	5	2017-12-15	star wars: the last jedi	317.000	620.181382	1316.721747	999.721747	tt2527336	star wars: the last jedi
4	6	2015-12-18	star wars: episode vii - the force awakens	306.000	936.662225	2053.311220	1747.311220	tt2488496	star wars: episode vii - the force awakens
...	...	...	...	...	...	...	...	...	...
2432	38	2016-03-18	krisha	0.030	0.144822	0.144822	0.114822	tt4266638	krisha
2433	41	2010-10-15	down terrace	0.030	0.009812	0.009812	-0.020188	NaN	NaN
2434	45	2017-01-27	emily	0.027	0.003547	0.003547	-0.023453	tt1863224	emily
2437	61	2010-04-02	breaking upwards	0.015	0.115592	0.115592	0.100592	NaN	NaN
2438	73	2012-01-13	newlyweds	0.009	0.004584	0.004584	-0.004416	tt1880418	newlyweds

1785 rows × 13 columns

In [30]:
```
#Get value counts of the movie titles to see what titles are being duplicated
df_merge.movie.value_counts()
```

Out[30]:
```
home                       24
the gift                   13
brothers                   13
the wall                   10
silence                    10
                           ..
death wish                  1
lovely, still               1
identity thief              1
deliver us from evil        1
the magnificent seven       1
Name: movie, Length: 1785, dtype: int64
```

I can fix the duplicates by just going through the list and seeing which movies actually match. I made the matches by looking at the release date and start year as well as looking at the budget and gross. I used the actual IMDb pages for each movie to double check the matches.

In [31]:
```
#Used for cleaning
df_merge[df_merge.movie == 'robin hood']
```

Out[31]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	profit	movie_id	primary_title
28	39	2010-05-14	robin hood	210.0	105.487148	322.459006	112.459006	tt0955308	robin hood
29	39	2010-05-14	robin hood	210.0	105.487148	322.459006	112.459006	tt2363363	robin hood
30	39	2010-05-14	robin hood	210.0	105.487148	322.459006	112.459006	tt4532826	robin hood
31	39	2010-05-14	robin hood	210.0	105.487148	322.459006	112.459006	tt6858500	robin hood
32	39	2010-05-14	robin hood	210.0	105.487148	322.459006	112.459006	tt8558276	robin hood
309	9	2018-11-21	robin hood	99.0	30.824628	84.747441	-14.252559	tt0955308	robin hood

Below is a new CSV file I made that stores the movie_id that matches each movie that has duplicates

In [33]:
```
#Read in information to remove duplicates
df_dup_remover = pd.read_csv(dir_path + 'Data/duplicate_remover.csv')
```

In [34]:

```
#Look at data
df_dup_remover
```

Out[34]:

	movie	movie_id
0	home	tt2224026
1	brothers	tt3802576
2	the gift	tt4178092
3	the promise	tt4776998
4	silence	tt0490215
...	...	...
289	burlesque	tt1126591
290	flight	tt1907668
291	elysium	tt1535108
292	don't breathe	tt4160708
293	the last stand	tt1549920

I am going to loop over each movie in df_dup_remover an drop rows that don't have the right movie_id

In [35]:

```
#Remove duplicates
for movie, movie_id in zip(df_dup_remover.movie, df_dup_remover.movie_id):
    if movie_id == 'remove': #Some movies did not actually match up with anything, so we remove them
        df_merge.drop(df_merge.index[df_merge.movie == movie], inplace = True)
    if movie != 'robin hood': #There are two robin hood movies, so I will handle them separately
        df_merge.drop(df_merge.index[(df_merge.movie == movie) & (df_merge.movie_id != movie_id)], inp
```

There are two movies called robin hood that match a movie from the IMDb database. There are two movies called the square, but only one matches up with something in IMDb. I need to treat these cases separately because the code above would just get rid of everything and make no match.

In [36]:

```
#Robin Hood
df_merge.drop(df_merge.index[(df_merge.movie == 'robin hood') & (df_merge.movie_id != 'tt0955308') & (
df_merge.drop(df_merge.index[(df_merge.movie == 'robin hood') & (df_merge.movie_id != 'tt4532826') & (
#The Square
df_merge.drop(df_merge.index[(df_merge.movie == 'the square') & (df_merge.id == 10)], inplace = True)
```

In [37]:

```
#Now, the combined data should not include any duplicates
df_merge.movie.value_counts()
```

Out[37]:

```
robin hood          2
the founder         1
sinister 2          1
alien: covenant     1
queen of katwe      1
                   ..
locke               1
remember me         1
barfi               1
standing ovation    1
max                 1
Name: movie, Length: 1783, dtype: int64
```

There are two robin hood movies, but that it because there are actually two different movies with that title in the dataset.

Now there are no repeats in the dataset. Just so I don't have to rerun the cells above every time I restart the kernel, I am going to save this DataFrame as a CSV file.

In [38]:
```
#Select the cells I want to keep (I am excluding redundant cells)
df_merge = df_merge[['release_date', 'movie', 'production_budget', 'domestic_gross', 'worldwide_gross',
                     'movie_id', 'runtime_minutes', 'genres']]
```

In [39]:
```
#Write cleaned data to file
df_merge.to_csv(dir_path + 'Data/TheNumbers_IMDB_Merge.csv')
```

Budget

In this section, I look at how the budget of movies are related to the profit the movies make.

In [43]:
```
#Read in data
df_merge = pd.read_csv(dir_path+'Data/TheNumbers_IMDB_Merge.csv')
```

In [48]:
```
#Change release_date to datetime
df_merge.release_date = pd.to_datetime(df_merge.release_date, format = '%Y-%m-%d')
```

In [49]:
```
df_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1784 entries, 0 to 1783
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         1784 non-null   int64
 1   release_date       1784 non-null   datetime64[ns]
 2   movie              1784 non-null   object
 3   production_budget  1784 non-null   float64
 4   domestic_gross     1784 non-null   float64
 5   worldwide_gross    1784 non-null   float64
 6   profit             1784 non-null   float64
 7   movie_id           1657 non-null   object
 8   runtime_minutes    1656 non-null   float64
 9   genres             1657 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(1), object(3)
memory usage: 139.5+ KB
```

Some movies have a domestic and/or worldwide gross that is equal to 0 dollars. I looked into one example (Bright) and the webpage for Bright on TheNumbers.com does not show a gross of 0, but instead just a dash. I think this means 0 is just a placeholder from an unknown amount.

In [68]:

```
#Remove movies with no recorded worldwide gross
df_merge[df_merge.worldwide_gross > 0].sort_values(by = 'profit', ascending = False)
```

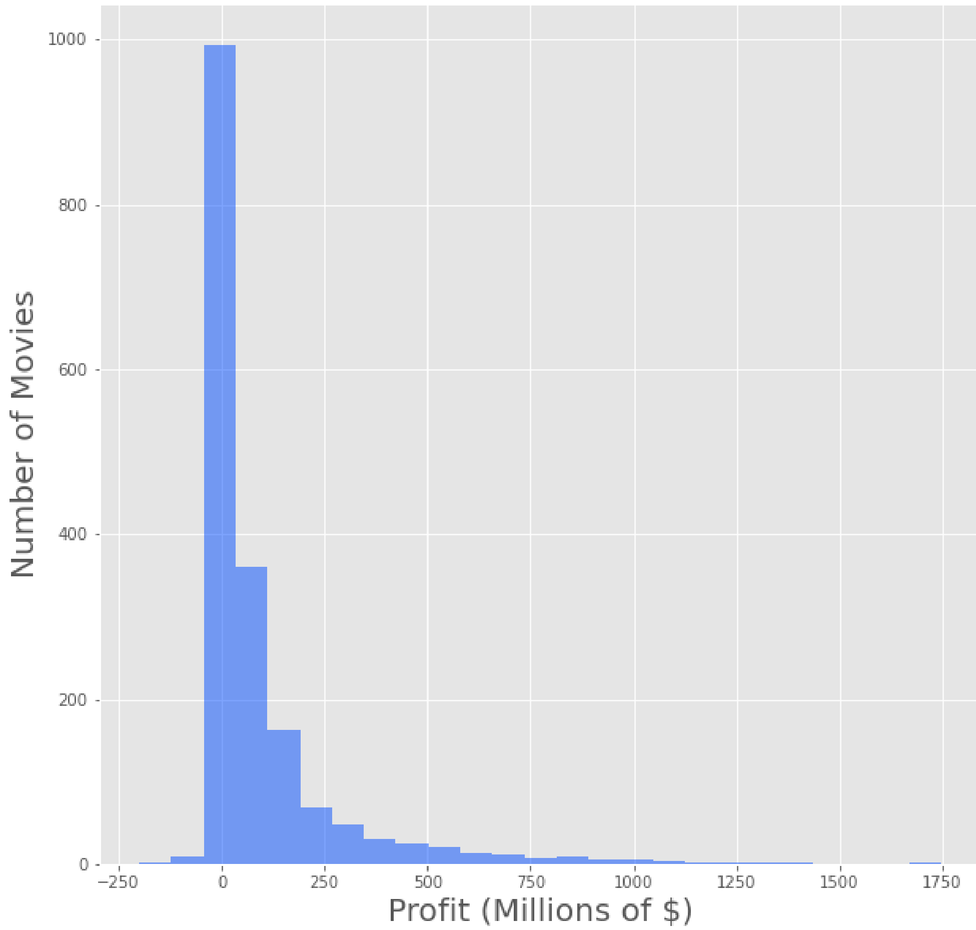
Out[68]:

	Unnamed: 0	release_date	movie	production_budget	domestic_gross	worldwide_gross	profit	movie_id	runtim
5	5	2018-04-27	avengers: infinity war	300.0	678.815482	2048.134200	1748.134200	tt4154756	
4	4	2015-12-18	star wars: episode vii - the force awakens	306.0	936.662225	2053.311220	1747.311220	tt2488496	
25	25	2015-06-12	jurassic world	215.0	652.270625	1648.854864	1433.854864	tt0369610	
50	56	2015-04-03	furious 7	190.0	353.007020	1518.722794	1328.722794	tt2820852	
20	20	2012-05-04	the avengers	225.0	623.279547	1517.935897	1292.935897	tt0848228	
...	...	...	...	...	...	...	...	...	
196	254	2010-12-17	how do you know	120.0	30.212620	49.628177	-70.371823	tt1341188	
273	359	2017-04-21	the promise	90.0	8.224288	10.551417	-79.448583	tt4776998	
218	277	2019-06-14	men in black: international	110.0	3.100000	3.100000	-106.900000	tt2283336	
134	161	2011-03-11	mars needs moms	150.0	21.392758	39.549758	-110.450242	tt1305591	
1	1	2019-06-07	dark phoenix	350.0	42.762350	149.762350	-200.237650	tt6565702	

1784 rows × 10 columns

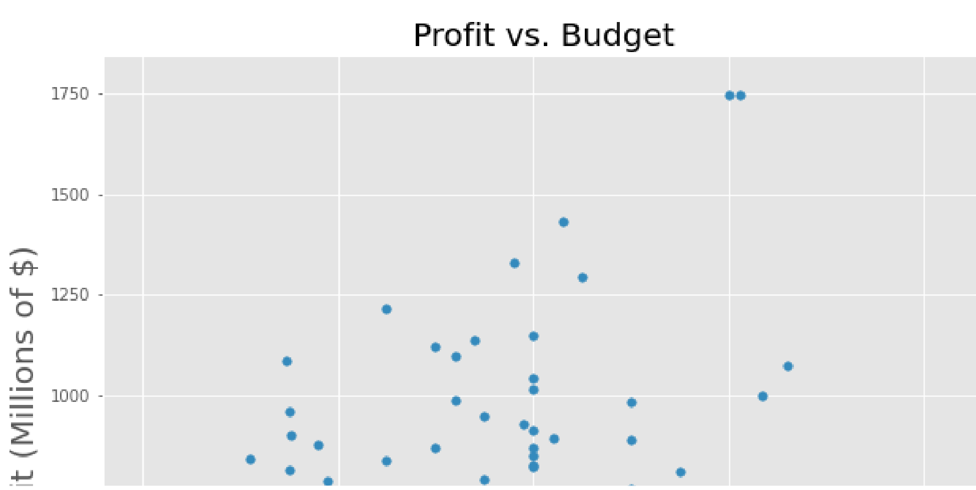
In [67]:
```
#Histogram of profit
ax = df_merge.profit.plot(kind = 'hist', bins=25, color = (0.0 , 0.3 , 1, 0.5))
ax.set_ylabel('Number of Movies', fontsize = 20)
ax.set_xlabel('Profit (Millions of $)', fontsize = 20)
```

Out[67]: Text(0.5, 0, 'Profit (Millions of $)')



In [69]:
```
ax = df_merge.plot('production_budget', 'profit', kind = 'scatter', s=30)
ax.set_xlabel('Production Budget (Millions of $)', fontsize = 20)
ax.set_ylabel('Worldwide Profit (Millions of $)', fontsize = 20)
ax.set_title('Profit vs. Budget', fontsize = 20)
```

Out[69]: Text(0.5, 1.0, 'Profit vs. Budget')



Below, I divide the movies into 10 groups based on their production budget. Each range will have the same number of movies. This is being added as a column called budget_range

In [70]:

```
#I am dividing the movies into 10 groups based on their production budget.
#This is being added to the table as budget_range
groups, edges = pd.qcut(df_merge.production_budget,q=10, retbins=True)

df_merge['budget_range'] = groups
```

In [71]:

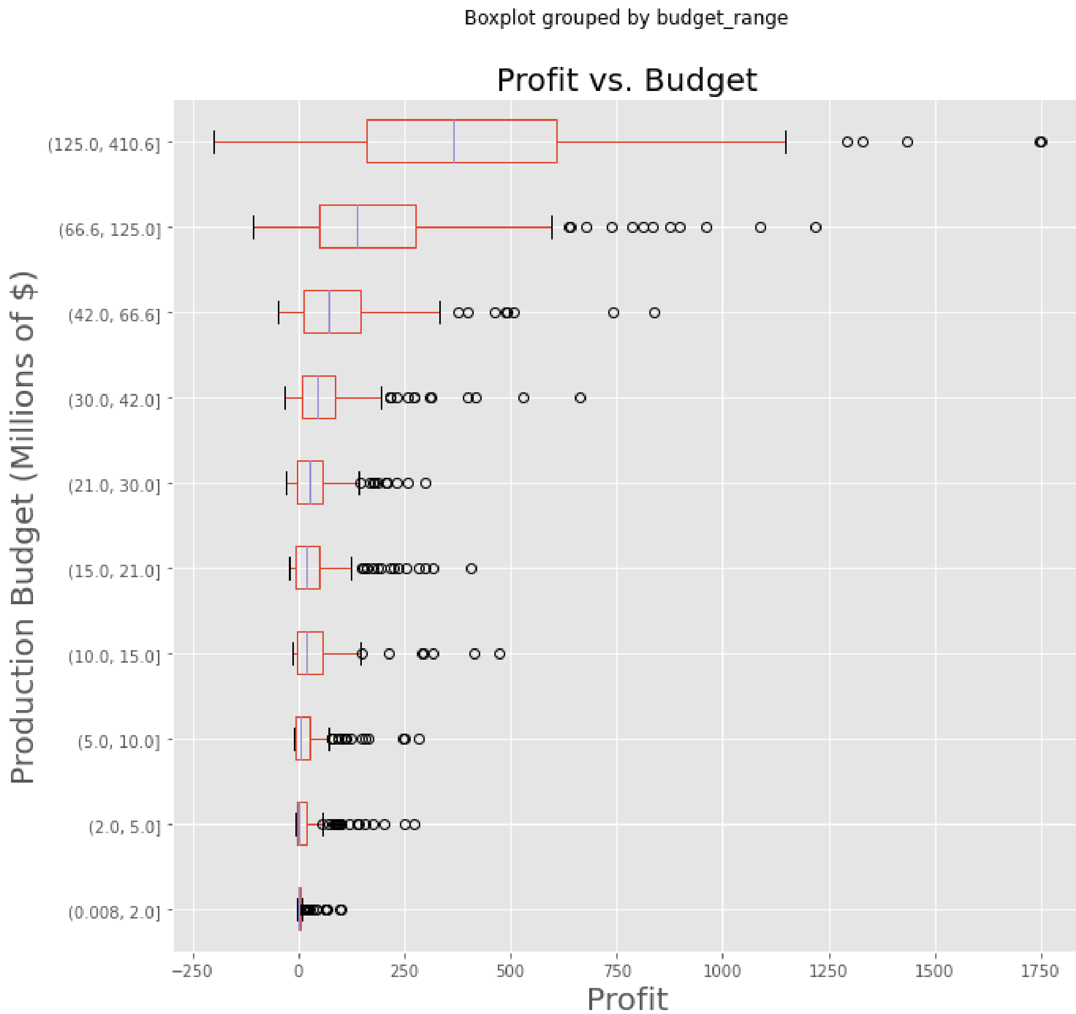
```
#Determine what percentage of movies were profitable in each budget range
#Profitable is True if movie made money, False if it did not
df_merge['Profitable?'] = df_merge.profit > 0
df_group = df_merge.groupby('budget_range').mean()
df_group
```

Out[71]:

budget_range	Unnamed: 0	production_budget	domestic_gross	worldwide_gross	profit	runtime_minutes	Profitable?
(0.008, 2.0]	2317.887755	1.025320	3.534122	6.023209	4.997889	95.272189	0.520408
(2.0, 5.0]	2063.740113	3.996328	13.186484	24.337733	20.341405	100.086093	0.598870
(5.0, 10.0]	1793.079812	8.323099	13.520780	28.064199	19.741100	105.414365	0.586854
(10.0, 15.0]	1550.885135	13.170270	24.796864	54.187265	41.016994	108.477612	0.716216
(15.0, 21.0]	1323.811321	18.913836	32.520400	58.827385	39.913548	107.033784	0.679245
(21.0, 30.0]	1081.638298	26.779787	35.383912	65.671996	38.892208	109.816667	0.739362
(30.0, 42.0]	823.056818	36.971023	49.537701	103.658113	66.687091	110.121387	0.784091
(42.0, 66.6]	579.200000	54.174235	63.775392	154.698468	100.524232	112.157576	0.811765
(66.6, 125.0]	343.260870	92.649457	105.114964	290.249091	197.599635	111.791209	0.885870
(125.0, 410.6]	103.341040	180.047977	210.812805	608.055506	428.007529	121.682081	0.947977

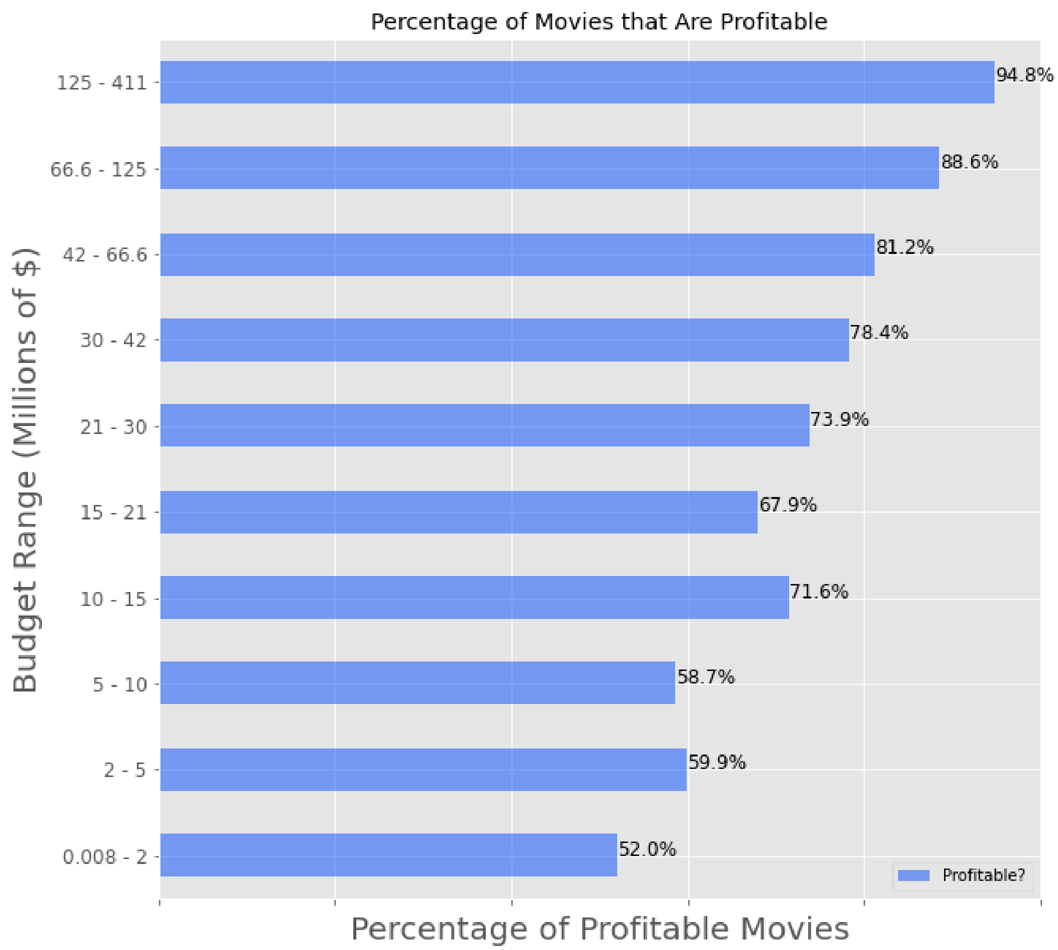
In [72]:
```
# Make a figure with boxplots for each range of budget
ax = df_merge.boxplot(column = 'profit', by = 'budget_range', vert=False)
ax.set_xlabel('Profit', fontsize = 20)
ax.set_ylabel('Production Budget (Millions of $)', fontsize = 20)
ax.set_title('Profit vs. Budget', fontsize = 20)
```

Out[72]: Text(0.5, 1.0, 'Profit vs. Budget')



In [77]:

```
# This plot shows the percentage of movies that are profitable within each budget range
ax = df_group.plot(y = 'Profitable?', kind='barh', color=(0.0,0.3,1,0.5), title = 'Percentage of Movie
ax.set_xlim((0,1))
ax.set_xlabel('Percentage of Profitable Movies', fontsize = 20)
ax.set_ylabel('Budget Range (Millions of $)', fontsize = 20)
ax.set_yticklabels(['0.008 - 2', '2 - 5', '5 - 10', '10 - 15', '15 - 21', '21 - 30', '30 - 42', '42 - 
                    '66.6 - 125', '125 - 411'], fontsize = 12)
ax.set_xticklabels([])
for i,perc in enumerate(df_group['Profitable?']):
    ax.text(perc, i, f"{round(100*perc,1)}%", fontsize = 12)
```



Conclusions about budget

The plot shows that the higher the budget, the higher the profit tends to be. Importantly, as the budget increases, the probability of making a profit increases. Movies with budgets exceeding 125 million made money about 95% of the time while movies with budgets between 42 and 66.6 million only made money about 81% of the time.

Spending lots of money does not guarantee a big payday. Just look at Dark Phoenix. That movie had a budget of 350 million and it lost 200 million. We need to make sure that we spend the money wisely. That is what the other recommendations will help us do.

Genres

Using the movies that are listed in both TheNumbers and the IMDB datasets, I will look at how the genre is related to both the profit and the rating.

In [78]:
```
#Read in movie_ratings data from IMDb
df_ratings = pd.read_sql("""
SELECT *
  FROM movie_ratings;
""", conn)
```

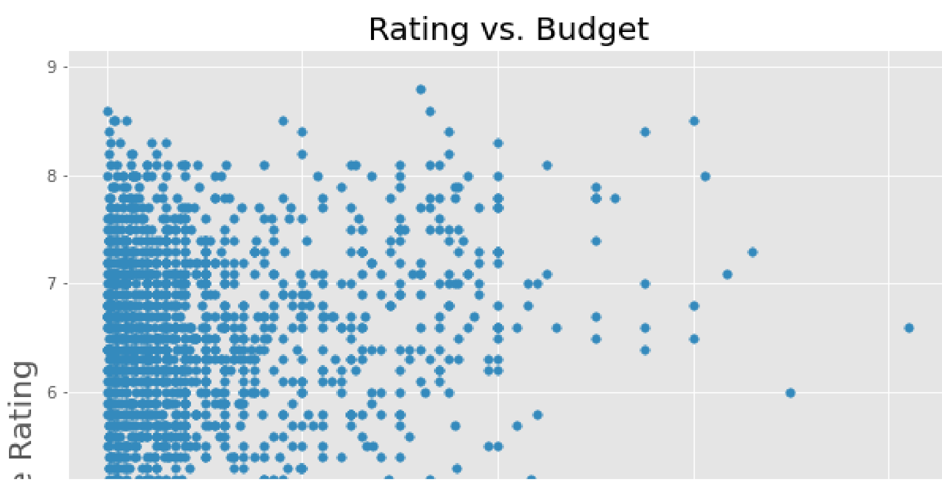
In [79]:
```
#Merge cleaned data with movie_ratings
df_merge = df_merge.merge(df_ratings, left_on = 'movie_id', right_on = 'movie_id', how = 'inner')
```

In [80]:
```
#Make it so the genres are all lower case
df_merge['genres'] = df_merge['genres'].str.lower()
```

Below, I look at rating vs. budget. It seems that very high budget movies have a higher floor than low budget movies in terms of ratings. High budget movies don't get completely panned by critics and viewers.

In [81]:
```
#I also looked at Rating vs. Budget
#It seems that very high budget movies have a higher floor than low budget movies in terms of ratings
ax = df_merge.plot('production_budget', 'averagerating', kind = 'scatter', s=30)
ax.set_xlabel('Production Budget (Millions of $)', fontsize = 20)
ax.set_ylabel('Average Rating', fontsize = 20)
ax.set_title('Rating vs. Budget', fontsize = 20)
```

Out[81]: Text(0.5, 1.0, 'Rating vs. Budget')



The IMDb data lists multiple genres per movie as a single string separated by commas. I want to get a list of unique genres so I can look at each genre separately. I found a nice snippet of code from https://medium.com/analytics-vidhya/exploratory-data-analysis-imdb-dataset-cff0c3991ad5 (https://medium.com/analytics-vidhya/exploratory-data-analysis-imdb-dataset-cff0c3991ad5)

In [82]:
```
from sklearn.feature_extraction.text import CountVectorizer

temp = df_merge.genres.dropna()
vec = CountVectorizer(token_pattern='(?u)\\b[\\w-]+\\b', analyzer='word').fit(temp)
unique_genres =  vec.get_feature_names()
unique_genres
```

Out[82]:
```
['action',
 'adventure',
 'animation',
 'biography',
 'comedy',
 'crime',
 'documentary',
 'drama',
 'family',
 'fantasy',
 'history',
 'horror',
 'music',
 'musical',
 'mystery',
 'romance',
 'sci-fi',
 'sport',
 'thriller',
 'war',
 'western']
```

In [84]:
```
#Find the median rating of each genre and sort the genres by that median
med = []
for genre in unique_genres:
    genre_med = df_merge[df_merge.genres.str.contains(genre)].averagerating.median()
    med.append(genre_med)

med
```

Out[84]:
```
[6.4,
 6.6,
 6.7,
 7.1,
 6.3,
 6.5,
 6.95,
 6.7,
 6.3,
 6.2,
 7.0,
 5.9,
 6.4,
 6.15,
 6.3,
 6.4,
 6.5,
 7.0,
 6.3,
 6.3,
 6.6]
```

In [85]:
```
#Sort genres by median rating
med_and_genres = sorted(list(zip(med,unique_genres)))
sorted_genres = [med_and_genres[i][1] for i in range(len(med_and_genres))]
sorted_genres
```

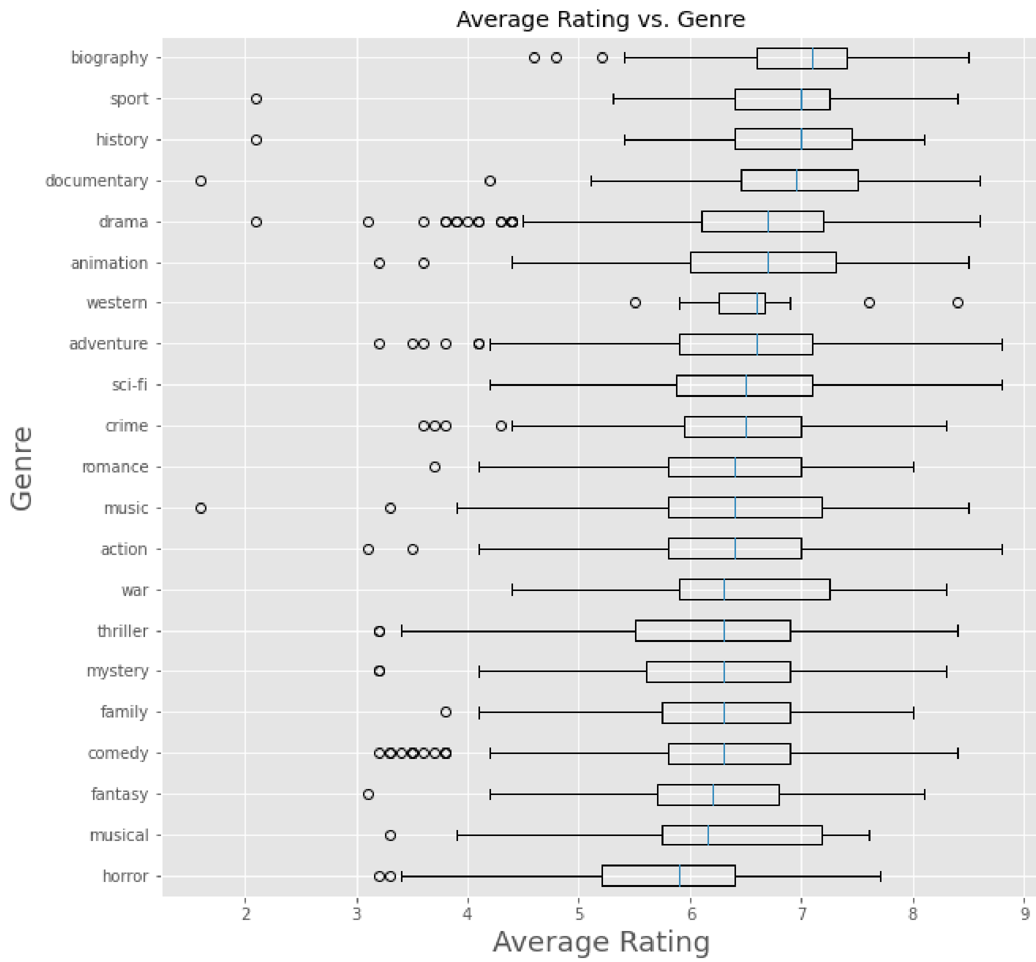
Out[85]:
```
['horror',
 'musical',
 'fantasy',
 'comedy',
 'family',
 'mystery',
 'thriller',
 'war',
 'action',
 'music',
 'romance',
 'crime',
 'sci-fi',
 'adventure',
 'western',
 'animation',
 'drama',
 'documentary',
 'history',
 'sport',
 'biography']
```

In [86]:

```
#Average rating vs. genre
fig, ax = plt.subplots(figsize = (10,10))

n=0
for genre in sorted_genres:
    #df_sub = df_merge[(df_merge.genres.str.contains(genre)) & (df_merge.production_budget > 10)]
    df_sub = df_merge[df_merge.genres.str.contains(genre)]
    ax.boxplot(df_sub.averagerating, vert = False, positions = [n], widths = 0.5)
    n = n + 1
ax.set_yticklabels(sorted_genres)
ax.set_title('Average Rating vs. Genre')
ax.set_xlabel('Average Rating', fontsize = 18)
ax.set_ylabel('Genre', fontsize = 18)
```

Out[86]:

```
Text(0, 0.5, 'Genre')
```



In terms of rating, it seems like people enjoy true stories because the highest rated genres are biography, history, and documentary. Sports movies are also quite high. Some of these movies are also based on true stories. The worst rated are horror, musical, fantasy, and comedy.

The truth is that the movies have a wide range of ratings and you can have a highly rated horror movie or a low-rated biography.

Now to see how the genres compare financially.

In [87]:
```
#Find the median profit of each genre and sort the genres by that median
med = []
for genre in unique_genres:
    genre_med = df_merge[df_merge.genres.str.contains(genre)].profit.median()
    med.append(genre_med)

med
```

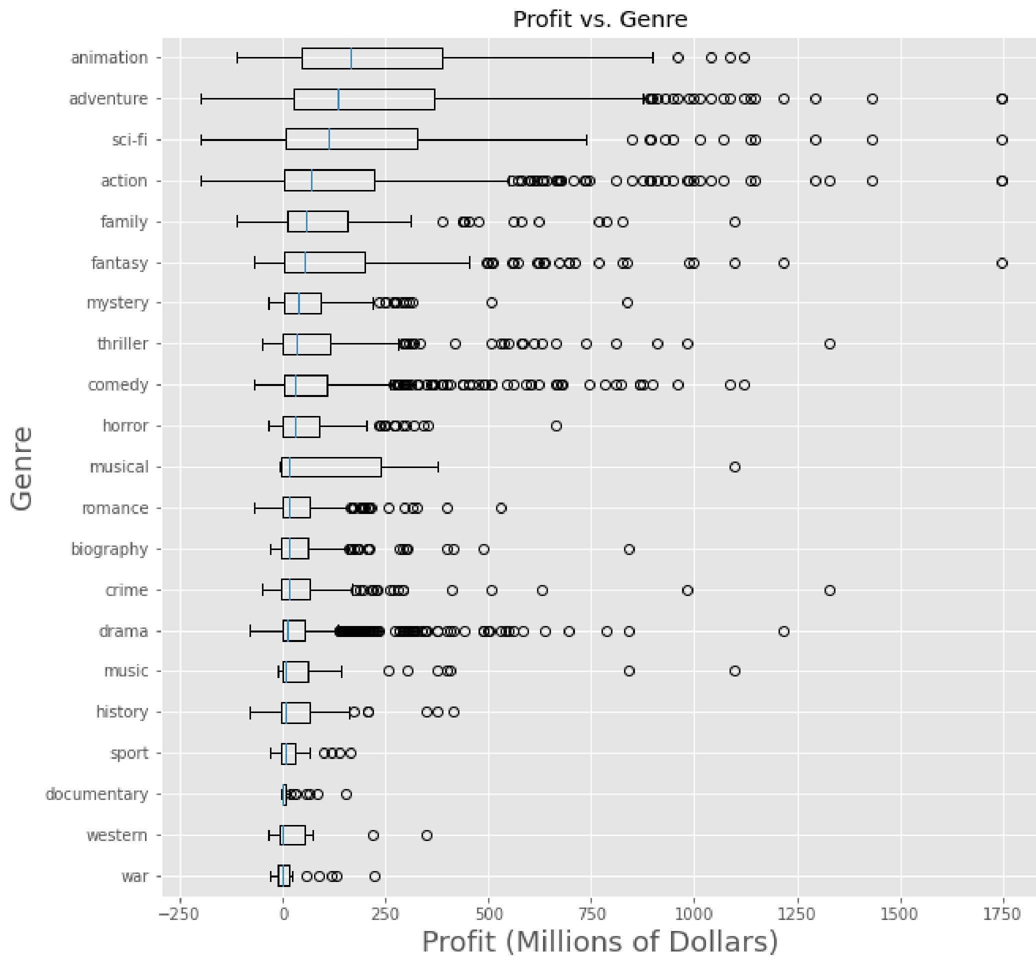
Out[87]:
```
[69.59089,
 132.98126100000002,
 166.56231200000002,
 16.0067175,
 31.887901500000005,
 13.844132,
 0.38636800000000004,
 12.141616999999998,
 55.462444500000004,
 53.461527,
 8.099931,
 30.74923,
 9.1696265,
 16.878986499999996,
 36.824065999999995,
 16.649645000000003,
 110.0982585,
 7.3621764999999995,
 33.866088000000005,
 -1.315295,
 -1.185188]
```

In [88]:
```
#Sort genres by median profit
med_and_genres = sorted(list(zip(med,unique_genres)))
sorted_genres = [med_and_genres[i][1] for i in range(len(med_and_genres))]
sorted_genres
```

Out[88]:
```
['war',
 'western',
 'documentary',
 'sport',
 'history',
 'music',
 'drama',
 'crime',
 'biography',
 'romance',
 'musical',
 'horror',
 'comedy',
 'thriller',
 'mystery',
 'fantasy',
 'family',
 'action',
 'sci-fi',
 'adventure',
 'animation']
```

In [89]:
```
#Profit vs. genre
fig, ax = plt.subplots(figsize = (10,10))

n=0
for genre in sorted_genres:
    #df_sub = df_merge[(df_merge.genres.str.contains(genre)) & (df_merge.production_budget > 10)]
    df_sub = df_merge[df_merge.genres.str.contains(genre)]
    ax.boxplot(df_sub.profit, vert = False, positions = [n], widths = 0.5)
    n = n + 1
ax.set_yticklabels(sorted_genres)
ax.set_title('Profit vs. Genre')
ax.set_xlabel('Profit (Millions of Dollars)', fontsize = 18)
ax.set_ylabel('Genre', fontsize = 18)
```

Out[89]: Text(0, 0.5, 'Genre')



People love their documentaries, but that doesn't mean those movies make money. Documentaries, biographies, and history movies are near the bottom in profits. The most profitable movies are animated movies, adventure movies, and sci-fi. The average is higher due to a small number of very successful movies. Inception, Interstellar, and Avengers: Infinity War all fall in the adventure and sci-fi genres. They are the bigger outliers to the right. However, the adventure and sci-fi genres do still have some of the highest MEDIAN profits, showing that it is not just a matter of the box office hits skewing the data.

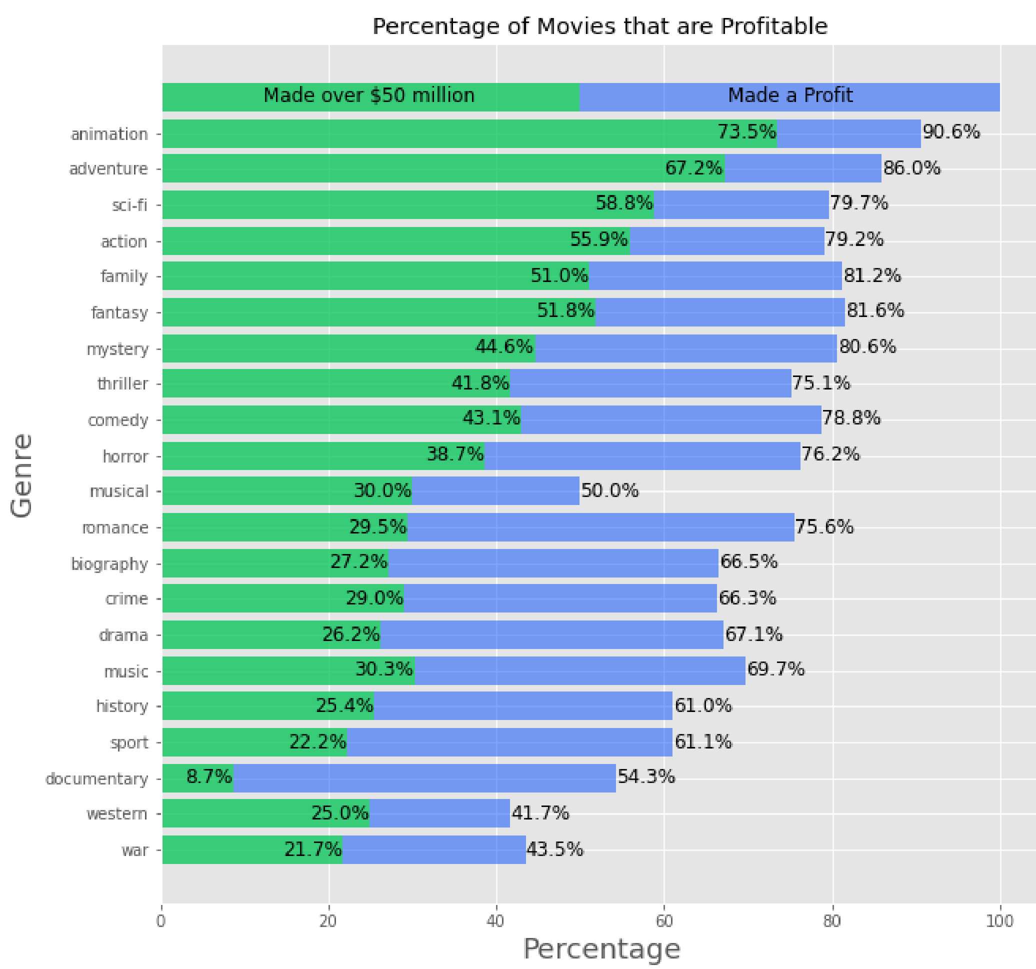
This does not necessarily mean I recommend making only animated sci-fi, adventure movies. Most of the genres include multiple highly profitable movies (the only genres that do not include at least one movie with a profit of 500 million dollars are horror, western, history, romance, sport, war, and documentary). These genres are probably not what we should aim for if we want to make a major profit.

In [90]:

```
#Percentage of movies that are profitable within each genre
fig, ax = plt.subplots(figsize = (10,10))

n=0
for genre in sorted_genres:
    df_sub = df_merge[df_merge.genres.str.contains(genre)]
    perc = 100 * len(df_sub[df_sub.profit > 0.0]) / len(df_sub)
    ax.barh(width = perc, height = 0.8, y = n, color = (0.0, 0.3, 1, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'left', verticalalignme

    perc = 100 * len(df_sub[df_sub.profit > 50.0]) / len(df_sub)
    ax.barh(width = perc, height = 0.8, y = n, color = (0.0, 1, 0, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'right', verticalalignm

    n = n + 1

ax.barh(width = 100, height = 0.8, y = n, color = (0.0 , 0.3 , 1, 0.5))
ax.barh(width = 50, height = 0.8, y = n, color = (0.0, 1, 0.0, 0.5))
ax.text(25, n, "Made over $50 million", fontsize = 12, horizontalalignment = 'center', verticalalignme
ax.text(75, n, "Made a Profit", fontsize = 12, horizontalalignment = 'center', verticalalignment = 'ce

ax.set_yticklabels(sorted_genres)
ax.set_yticks(range(len(sorted_genres)))
ax.set_title('Percentage of Movies that are Profitable')
ax.set_xlabel('Percentage', fontsize = 18)
ax.set_ylabel('Genre', fontsize = 18)
```

```
<ipython-input-90-33097a5fd156>:22: UserWarning: FixedFormatter should only be used together with Fix
edLocator
  ax.set_yticklabels(sorted_genres)
```

Out[90]: Text(0, 0.5, 'Genre')



Genre Conclusion:

While a wide variety of genres are capable of earning large profits and high ratings, the most successful genres are animation, adventure, sci-fi, and action. Movies in the war, western, and sports genres are not as successful. Documentaries are very popular (high ratings), but don't typically make much money.

Star Power

Next, I will look at how the profit and ratings of movies are affected by the people who make those movies. This will be the hardest to code. Plan: For each movie, create a "star power" rating. I will look at the principal people involved in each movie, then I will count how many movies those people have previously done. I can also require that those movies had a certain level of success (profit above 50 million). To do this, I will need to loop through each movie and find the people associated with that movie (using the principals table). Then, I will sum up all of the movies those people have PREVIOUSLY done to get the star power rating.

PROBLEM: If the movies in my main dataset start in 2010, then this limits what I can do with this star power metric. This means movies released prior to 2010 will not contribute to the star power rating.

How to fix it: I can specifically look at movies from the last 6 years and use star power since 2010. This should still work pretty well because more recent starring roles are probably more influential than ones from decades ago.

In [91]:

```
# Read in the principals table from IMDB
# I want to do this separately for actors/actresses and directors
df_actors = pd.read_sql("""
SELECT movie_id, person_id, category
  FROM principals
  WHERE category IN ('actor', 'actress')
;
""", conn)

# Directors
df_directors = pd.read_sql("""
SELECT movie_id, person_id, category
  FROM principals
  WHERE category IN ('director')
;
""", conn)
```

In [92]:

```
#Create a new column for actor/actress star power
df_merge['act_star_power'] = 0.0
df_merge['dir_star_power'] = 0.0
```

Below, I loop over every movie and do the following:

1. Find the principal actors in the movie.
2. Count the previous movies those actors have starred in that made at least $100 million.
3. Sum those movies and add it to the act_star_power column

In [104]:

```
# This is for actors/actresses only
star_power_list = []
for mov_id in df_merge.movie_id:
    release_date = df_merge[df_merge.movie_id == mov_id].release_date
    person_ids = list(df_actors[df_actors.movie_id == mov_id].person_id)
    df_mov_with_same_people = df_actors[df_actors.person_id.isin(person_ids)].merge(df_merge, left_on =
                                                                                    right_on = 'movie_
    star_power_list.append(len(df_mov_with_same_people[(df_mov_with_same_people.release_date < release_
                                                        (df_mov_with_same_people.profit > 50)]))

df_merge['act_star_power'] = star_power_list
```

In [105]:

```
# This is for directors only
star_power_list = []
for mov_id in df_merge.movie_id:
    release_date = df_merge[df_merge.movie_id == mov_id].release_date
    person_ids = list(df_directors[df_directors.movie_id == mov_id].person_id)
    df_mov_with_same_people = df_directors[df_directors.person_id.isin(person_ids)].merge(df_merge, le
                                                                                    right_on = 'movie_
    star_power_list.append(len(df_mov_with_same_people[(df_mov_with_same_people.release_date < release_
                                                        (df_mov_with_same_people.profit > 50)]))

df_merge['dir_star_power'] = star_power_list
```

Below, I make a new DataFrame that includes movies since 2016. The reason for doing this is because movies prior to that won't have very accurate star power ratings

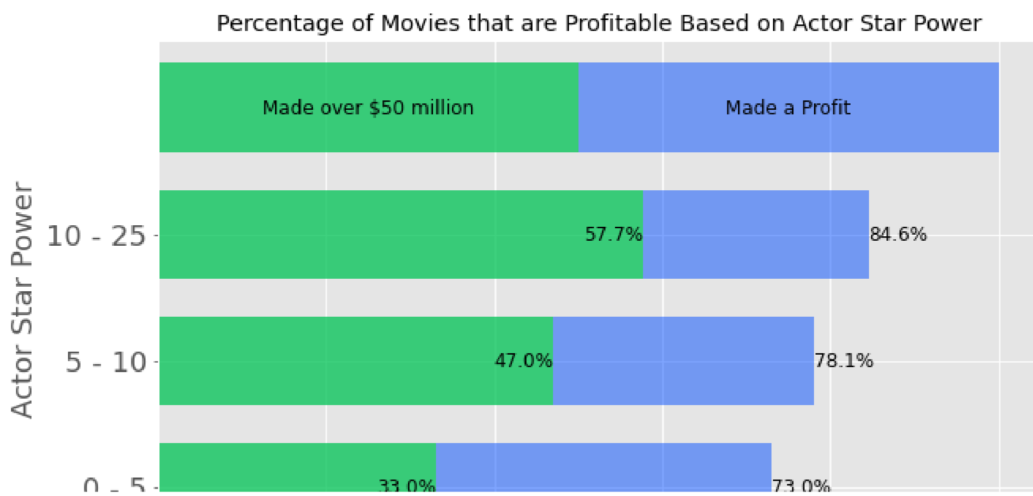
In [106]:

```
#Keep data starting in 2016
df_cut = df_merge[df_merge.release_date > '2016-01-01']
```

In [107]:

```
#Percentage of movies that are profitable based on actor/actress star power
fig, ax = plt.subplots(figsize = (10,6))

star_power_bins = [0,5,10,25]

for n in range(len(star_power_bins)-1):
    perc = 100 * len(df_cut[(df_cut.act_star_power >= star_power_bins[n]) & (df_cut.act_star_power < st
                        & (df_cut.profit > 0.0)]) / \
        len(df_cut[(df_cut.act_star_power >= star_power_bins[n]) & (df_cut.act_star_power < star_power_

    ax.barh(width = perc, height = 0.7, y = n, color = (0.0, 0.3, 1.0, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'left', verticalalignme

    perc = 100 * len(df_cut[(df_cut.act_star_power >= star_power_bins[n]) & (df_cut.act_star_power < st
                        & (df_cut.profit > 50.0)]) / \
        len(df_cut[(df_cut.act_star_power >= star_power_bins[n]) & (df_cut.act_star_power < star_power_

    ax.barh(width = perc, height = 0.7, y = n, color = (0.0, 1, 0.0, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'right', verticalalignm

n = n + 1
ax.barh(width = 100, height = 0.7, y = n, color = (0.0 , 0.3 , 1, 0.5))
ax.barh(width = 50, height = 0.7, y = n, color = (0.0, 1, 0.0, 0.5))
ax.text(25, n, "Made over $50 million", fontsize = 12, horizontalalignment = 'center', verticalalignme
ax.text(75, n, "Made a Profit", fontsize = 12, horizontalalignment = 'center', verticalalignment = 'ce

ax.set_yticklabels([f"{star_power_bins[n]} - {star_power_bins[n+1]}" for n in range(len(star_power_bin
ax.set_xticklabels(np.arange(0,120,20),fontsize = 18)
ax.set_yticks(range(len(star_power_bins)-1))
ax.set_title('Percentage of Movies that are Profitable Based on Actor Star Power')
ax.set_xlabel('Percentage', fontsize = 18)
ax.set_ylabel('Actor Star Power', fontsize = 18)
```

Out[107]: Text(0, 0.5, 'Actor Star Power')



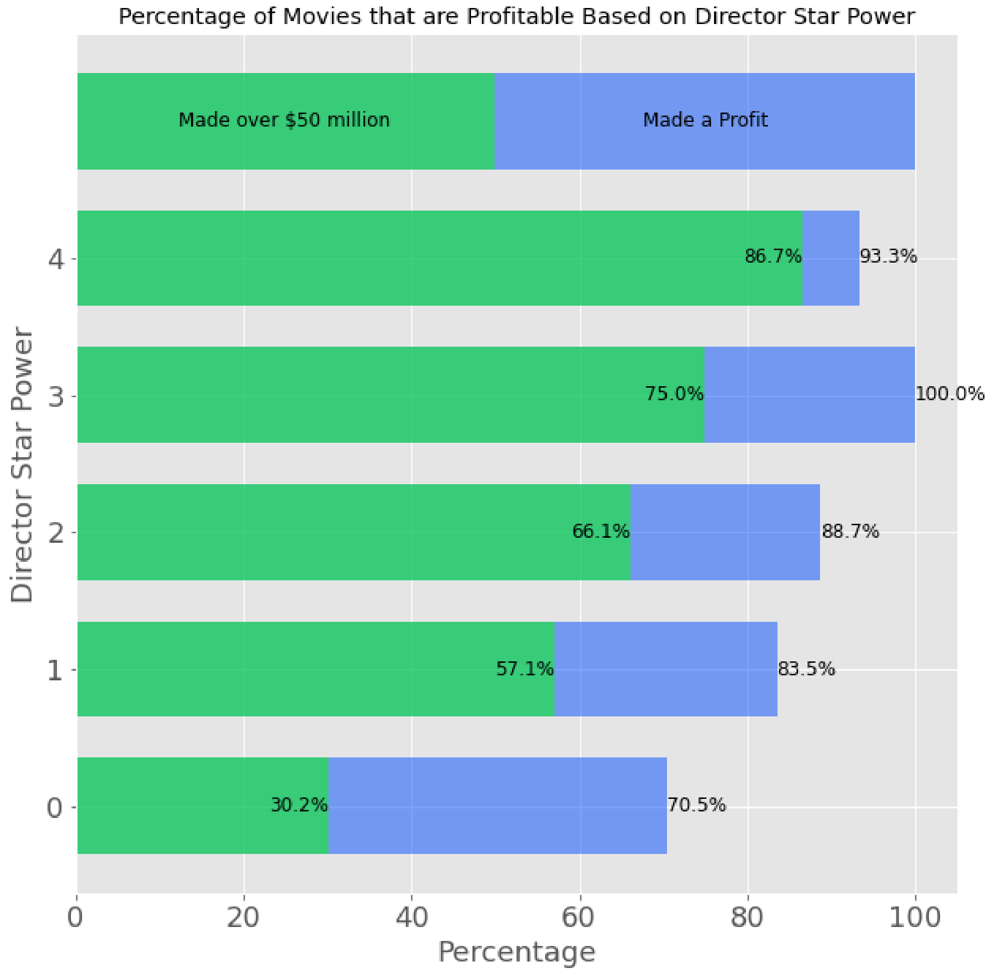
In [108]:

```
#Percentage of movies that are profitable based on director star power
fig, ax = plt.subplots(figsize = (10,10))

star_power_bins = [0,1,2,3,4,5,6]

for n in range(5):
    perc = 100 * len(df_cut[(df_cut.dir_star_power >= star_power_bins[n]) & (df_cut.dir_star_power < st
                        & (df_cut.profit > 0.0)]) / \
        len(df_cut[(df_cut.dir_star_power >= star_power_bins[n]) & (df_cut.dir_star_power < star_power_

    ax.barh(width = perc, height = 0.7, y = n, color = (0.0, 0.3, 1.0, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'left', verticalalignme

    perc = 100 * len(df_cut[(df_cut.dir_star_power >= star_power_bins[n]) & (df_cut.dir_star_power < st
                        & (df_cut.profit > 50.0)]) / \
        len(df_cut[(df_cut.dir_star_power >= star_power_bins[n]) & (df_cut.dir_star_power < star_power_

    ax.barh(width = perc, height = 0.7, y = n, color = (0.0, 1, 0.0, 0.5))
    ax.text(perc, n, f"{round(perc,1)}%", fontsize = 12, horizontalalignment = 'right', verticalalignm

n = n + 1
ax.barh(width = 100, height = 0.7, y = n, color = (0.0 , 0.3 , 1, 0.5))
ax.barh(width = 50, height = 0.7, y = n, color = (0.0, 1, 0.0, 0.5))
ax.text(25, n, "Made over $50 million", fontsize = 12, horizontalalignment = 'center', verticalalignme
ax.text(75, n, "Made a Profit", fontsize = 12, horizontalalignment = 'center', verticalalignment = 'ce

ax.set_yticklabels(range(5), fontsize = 18)
ax.set_xticklabels(np.arange(0,120,20),fontsize = 18)
ax.set_yticks(range(5))
ax.set_title('Percentage of Movies that are Profitable Based on Director Star Power')
ax.set_xlabel('Percentage', fontsize = 18)
ax.set_ylabel('Director Star Power', fontsize = 18)
```

```
<ipython-input-108-b9d7c6393af7>:27: UserWarning: FixedFormatter should only be used together with Fi
xedLocator
  ax.set_yticklabels(range(5), fontsize = 18)
<ipython-input-108-b9d7c6393af7>:28: UserWarning: FixedFormatter should only be used together with Fi
xedLocator
  ax.set_xticklabels(np.arange(0,120,20),fontsize = 18)
```

Out[108]:

```
Text(0, 0.5, 'Director Star Power')
```



Casting/Directing Conclusions

Hiring actors/actresses and directors who have previously been involved in successful, profitable movies does seem to be correlated with future success. This could be because those people are talented and are therefore more likely to help create a good movie. It could also be because having big, recognizable names attached to a movie helps get more attention on that movie so that people will go see it, regardless of its actual quality. It is easier to sell a movie starring Tom Hanks then a movie starring John Whoever, even if Mr. Whoever is very talented.