LandonTatro / Flatiron_Capstone Public

Code Issues Pull requests Actions Projects Wiki Security Insig

main

Flatiron_Capstone / notebooks / Film_Factors_And_Association_With_Profit.ipynb

DimaTaher updated final notebook and reran all the notebooks History

2 contributors

5477 lines (5477 sloc) 471 KB

Film Factors and Association with Profitability

Authors: Dima Taher, Leo Muntaner, Landon Tatro, Morgan Pelletier (Deloitte Strategy and Analytics - Orlando)

Business Overview

The problem

We were tasked with providing Computing Vision a series of suggestions for their transition into the film industry. Specifically, we aimed to determine suggestions that could target higher levels of revenue and/or profit.

The goal

The film industry is a creative and diverse market with several avenues to success. With the understanding that there is no one path to success, we aimed to generate insight into a variety of actions Computing Vision may want to take in order to carve their own unique path to success. To accomplish this goal, we analyzed several different facets of films and their relation to generating revenue and profit. These areas included genres, day of release, and experience level of directors.

The Datasets

The range of our analyses required utilization of several datasets. For each area of analysis, we used:

Directors:

- Movie Info dataset from Rotten Tomatoes which included the Director column required for this analysis with 1,560 rows.

- Movie Budgets dataset from The-Numbers.com which included the movie titles, production budget, and worlwide gross revenue which we used to calculate the profit which is our main measure of success in the project, the columns had 5,782 rows.

Release Day:

- Used the rt.movie_info.tsv dataset which included the day of release and box office revenue column required for this analysis with 1,560 entries.

Genres:

- Movie budgets dataset from the-numbers.com including movie titles, production budget, and worldwide gross revenue with 5,782 rows.
- TheMovieDB dataset including movie titles and genres with 26,517 rows.

The Methods and Results

Imports

In [1]:

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
import datetime

import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.ticker import StrMethodFormatter
%matplotlib inline

import scipy.stats
import scipy.optimize
import scipy.spatial
from sklearn.preprocessing import OneHotEncoder

from IPython import display
from ipywidgets import interact, widgets

import sqlite3
import re
import mailbox
import csv
import seaborn as sns
```

```
import statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf

import warnings
warnings.filterwarnings(action = 'ignore', categ
```

Experience Level of Directors in relation to Profit

The Business Question

Is there an association between the director's expertise and a movie's profitability?

The Datasets

In this section, we used the following datasets:

- Movie Info dataset from Rotten Tomatoes which included the Director column required for this analysis with 1,560 rows.
- Movie Budgets dataset from The-Numbers which included the movie titles, production budget, and worlwide gross which we used to calculate the profit which is our main measure of success in the project, the columns had 5,782 rows.

Movie Info

Starting with the movie info dataframe, the first step is to read into the tsv file

In [2]:

```
# Read in the data as movie_info
# Use parse_dates and pass column name to read i
movie_info = pd.read_csv('../Data/rt.movie_info.
                         parse_dates=['theater_da
movie_info.head()
```

Out[2]:

	id	synopsis	rating	genre	dir
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	W Fri
1	3	New York City, not-...	R	Drama|Science Fiction	

1	3	too-distant-future: Eric Pa...	R	and Fantasy	Cronei
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	A A
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Lev
4	7	NaN	NR	Drama|Romance	Rc Be

Data Cleaning

In this section, we will start by cleaning the movie_info dataframe before we can draw any conclusions. It will help us inspect the data better and get a more accurate general understanding of the data at hand. We will check for null values (missing values within the data set) and we will replace those values so that it's all consistent across the columns. We will then check for duplicates, there were no duplicates within our data set so we were good to proceed from there.

Step 1: Check if we have any null values in each column

In [3]:
```
movie_info.isnull().sum()
```

Out[3]:
```
id                 0
synopsis          62
rating             3
genre              8
director         199
writer           449
theater_date     359
dvd_date         359
currency        1220
box_office      1220
runtime           30
studio          1066
dtype: int64
```

Step 2: Dealing with null values

To clean the columns from null values, we will be replacing the null values in the column with generic terms relevant to each column so that it's all consistent across the columns.

In [4]:

```
#Fill the missing values in synposis, genre, dire
movie_info['synopsis'].fillna('Missing', inplace=
movie_info['rating'].fillna('Missing', inplace=Tı
movie_info['genre'].fillna('Missing', inplace=Tru
movie_info['director'].fillna('Missing', inplace=
movie_info['writer'].fillna('Missing', inplace=Tı
movie_info['currency'].fillna('Missing', inplace=
movie_info['studio'].fillna('Missing', inplace=Tı
```

In [5]:

```
#Fill theater_date and dvd_date missing values w
movie_info['theater_date'].fillna('1800-01-01', i
movie_info['dvd_date'].fillna('1800-01-01', inpla
```

In [6]:

```
#Fill box_office missing valus with 0
movie_info['box_office'].fillna(0, inplace=True)
```

In [7]:

```
#Fill runtime missing valus with 0
movie_info['runtime'].fillna('0 minutes', inplac

#Change the type of data so that we are able to u
movie_info['runtime'] = movie_info['runtime'].st
movie_info['runtime'] = pd.to_numeric(movie_info
```

Step 3: check for any duplicates

In [8]:

```
movie_info.duplicated().value_counts()
```

Out[8]:

```
False    1560
dtype: int64
```

It doesn't look like we have any duplicates. In this case, we are good to proceed forward.

Analysis methods

In this section, We will look into the Director column within this data frame to see if there is an association between the director's expertise and the movie's profitability.

We will look at the count of movies directed per director, as directors with more experience could potentially yield higher profit due to their expertise potentially yield higher profit due to their expertise.

We will also be looking at the Budgets dataframe to extract the profit from it and relate it to the director's experience.

- First, we will look at the budgets table.
- Second, we want to look at the trend between the count of movies per director and the profit
  - To do that, we will merge the budget dataframe and movie_info dataframe to check the profit generated by each director.
- Third, we will sort the top directors with the highest average profit.

We will start by reading into the budgets table and cleaning it

In [9]:

```
#reading into the csv data file
budgets = pd.read_csv("../data/tn.movie_budgets.
budgets.head()
```

Out[9]:

	id	release_date	movie	production_budget	domes
0	1	Dec 18, 2009	Avatar	$425,000,000	$76
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$24
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$4
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$45
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$62

In this table we will assume that **Revenue** = 'worldwide_gross' & **Cost** = 'production_budget'

In [10]:

```
#will follow Morgan's data cleaning for this tab
# removing $ and , from gross revenue
budgets['worldwide_gross'] = budgets['worldwide_
budgets['worldwide_gross'] = budgets['worldwide_
```

```
# casting the values as integers
budgets['worldwide_gross'] = pd.to_numeric(budge

# removing $ and , from production budget
budgets['production_budget'] = budgets['producti
budgets['production_budget'] = budgets['producti

# casting the values as integers
budgets['production_budget'] = pd.to_numeric(bud
```

In [11]:

```
# calculating total profit = revenue - cost
budgets['total_profit'] =  budgets['worldwide_gr
```

From here on we will be comparing the total profit from the budgets table to the Directors in the movie_info table.

Since we don't need all the columns in the dataframe, we will create a new one with only the columns necessary to the analysis.

In [12]:

```
movie_budgets = budgets[['id','movie','worldwide_
```

Merging movie_info & budget dataframes

We will join the dataframes using an **inner** join because it returns only the records with matching keys in both tables, we will make a separate dataframe for the joined dataframes. Originally, the movie info dataframe had 1560 entries and the budgets dataframe had 5782 entries, after our merge we were left with 1560 entries in total since we did an inner join.

In [13]:

```
movie_info_budget = movie_budgets.join(movie_inf
#checking the resulting dataframe
movie_info_budget.head()
```

Out[13]:

	id_movie_budgets	movie	worldwide_gross	total_pı
0	1	Avatar	2776345279	2351345
		Pirates of the Caribbean:		

1	2	Caribbean: On Stranger Tides	1045663875	635063
2	3	Dark Phoenix	149762350	-200237
3	4	Avengers: Age of Ultron	1403013963	1072413
4	5	Star Wars Ep. VIII: The Last Jedi	1316721747	999721

In [14]:

```
#We want to check how many 0 we have for worldwi
movie_info_budget['worldwide_gross'].describe()
```

Out[14]:

```
count    1.560000e+03
mean     2.374879e+08
std      2.686596e+08
min      0.000000e+00
25%      6.806081e+07
50%      1.523167e+08
75%      3.029080e+08
max      2.776345e+09
Name: worldwide_gross, dtype: float64
```

To deal with these values, we decided to replace it with the median profit because the median is more resilient against extreme outliers.

In [15]:

```
median_gross = movie_info_budget['worldwide_gros
movie_info_budget['worldwide_gross'] = movie_inf
```

We will now start looking if there are any trends between the expertise of the director vs. the profit. To do that, we created a new dataframe 'top_directors" that consisted of the count of movies per director, the total profit, and the average profit.

In [16]:

```
#Create top directors df so that we can visualize
#The count will show us the count of movies each
director_counts = pd.DataFrame(movie_info_budget
```

```
#We are summing the profit of all the movies per
director_total_profit = pd.DataFrame(movie_info_
```

In [17]:

```
# I will now join the director counts and total p
top_directors = director_counts.join(director_to
top_directors = top_directors.sort_values(by='to
```

In [18]:

```
#Dropping the missing values
top_directors = top_directors.drop(labels="Missi
```

In [19]:

```
#We will add the average profit per director sin
top_directors['avg_profit'] =  top_directors['to
top_directors['avg_profit'] = top_directors['avg_
top_directors
```

Out[19]:

director	movie	total_profit	avg_profit
William Friedkin	4	2705957834	6.764895e+08
Henning Schellerup	1	2008208395	2.008208e+09
Steven Spielberg	10	1777836004	1.777836e+08
Jake Kasdan	1	1748134200	1.748134e+09
Jay Russell	1	1747311220	1.747311e+09
...	...	...	...
Robert Hartford-Davis	1	-94635231	-9.463523e+07
Renny Harlin	2	-111069937	-5.553497e+07
Richard Thorpe	2	-117780537	-5.889027e+07
Jack Bender	1	-150000000	-1.500000e+08
Allison Anders	1	-200237650	-2.002376e+08

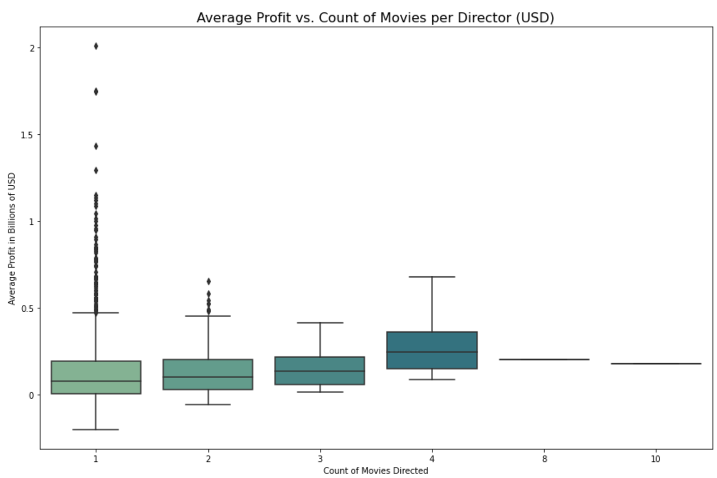
1125 rows × 3 columns

Now to visualize the results we will display it using a boxplot to display the spread of the data.

In [20]:

```
# We will visualize the results
# Plot average profit vs. count of movies directe
from matplotlib.ticker import FuncFormatter

fig, ax = plt.subplots(figsize = (12,8))
sns.boxplot(x = top_directors['movie'], y = top_
plt.title('Average Profit vs. Count of Movies per
plt.xlabel('Count of Movies Directed')
plt.ylabel('Average Profit in Billions of USD')
```

```
plt.tight_layout()


# scale y axis to millions
scale_y = 1e9
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
# ax.set_ylim(-250000000, 1000000000)

plt.show()
```



From this boxplot, we found a trend between increased experience and increased average profit. Specifically, after three movies, the distribution of profits was entirely positive. Directors with 3 or more movies have always had an average profit that is positive, we can see that their minimum is always positive.

There are many outliers for Directors with just one movie, and a few in those with 2 movies, but as we move to directing 3 or more we don't see outliers. These outliers could be due to many factors, one of them might be luck, but the trend we see is that as directors continued to work on three or more movies, they’ve continued to be profitable.

Therefore, we can conclude that as these Directors became more experienced by working on more movies, they've continued to be profitable and the factors that may have previously contributed to the outliers such as the factor of luck are eliminated.

Drawing from this trend, going forward we will look at the top 20 directors in terms of average profit that directed 3 or more movies.

In [21]:

```
#Getting the top 20 directors that directed 3 or
three_plus_movies = top_directors[(top_directors
three_plus_movies = three_plus_movies.reset_inde
three_plus_movies
```

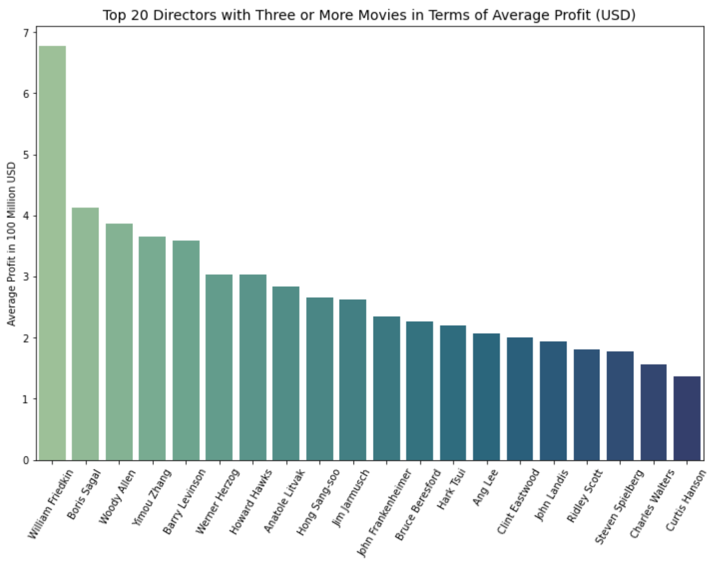
Out[21]:

	director	movie	total_profit	avg_profit
0	William Friedkin	4	2705957834	6.764895e+08
1	Steven Spielberg	10	1777836004	1.777836e+08
2	Clint Eastwood	8	1607570579	2.009463e+08
3	Woody Allen	4	1546517681	3.866294e+08
4	Yimou Zhang	4	1458132256	3.645331e+08
5	Barry Levinson	4	1435779099	3.589448e+08
6	Boris Sagal	3	1237332495	4.124442e+08
7	Jim Jarmusch	4	1050825592	2.627064e+08
8	Werner Herzog	3	911024954	3.036750e+08
9	Howard Hawks	3	909512843	3.031709e+08
10	Bruce Beresford	4	905154964	2.262887e+08
11	Anatole Litvak	3	851817812	2.839393e+08
12	Hong Sang-soo	3	798405830	2.661353e+08
13	Ridley Scott	4	719963861	1.799910e+08
14	John Frankenheimer	3	704291959	2.347640e+08
15	Hark Tsui	3	659235525	2.197452e+08
16	Ang Lee	3	622916667	2.076389e+08
17	John Landis	3	581449675	1.938166e+08
18	Curtis Hanson	4	549082973	1.372707e+08
19	Charles Walters	3	467879183	1.559597e+08

In [22]:

```
#now let's visualize the results
fig, ax = plt.subplots(figsize=(12, 8))
sns.barplot(x = three_plus_movies['director'], y
plt.xticks(rotation=60)
plt.xlabel(None)
plt.ylabel('Average Profit in 100 Million USD')
plt.title('Top 20 Directors with Three or More Mo


# scale y axis to millions
scale_y = 1e8
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
```

```
# ax.set_ylim(-250000000, 1000000000)

plt.show()
```



This barplot displays the top 20 directors in terms of average profit that have directed 3 or more movies. Drawing from this, if Computing Vision has the budget they could potentially hire one of these directors, or they can look into their work and get inspired by it.

Conclusion / Suggestion

In the context of this project an experienced director is defined as a director that has directed three or more movies.

Since Computing Vision is a new studio that is just starting in the movie industry, we ***recommend*** that they take a risk averse route and hire an experienced director that has experience directing three or more movies since our analysis showed that it is likely for these directors to continue being profitable as they work on the third movie and on.

What limitations are there?

One limitation that could be pointed out could be that there is a survival bias in the research, in which survival bias is defined as a type of sampling error or selection bias that occurs when the selection process of a trial favours certain individuals who made it past a certain obstacle or point in time and ignores the individuals who did not. In our case it would be selecting Directors with three or movies, and ignoring the ones with less experience. However, we concluded that for a new studio it is preferable that on their first projects that they take a route that is proven to be successful and taking less risks and that is by hiring an experienced director to direct their movies.

Another limitation is in the case of trying to recommend to hire one of the top 20 Directors in terms of their average profit and their expertise, a limitation was that some of the top 20 directors are in fact deceased. Deceased Directors: Boris Karloff, Howard Hawks, Anatole Litvak, John Frankenheimer, Charles Walters, Curtis Hanson.

However, in light of this limitation, a business suggestion here would be to look into these directors' work and potentially acquiring the rights to their work if possible and generate profit off of that.

We also imputed some values in our dataset with the median to preserve a decent sample size. While this might cause some inaccuracy but we chose the median because it is more resilient against extreme outliers.

Day of Release as a Predictor of Revenue

Project Goals, Data, Methods, and Results:

In this notebook you will find my data cleaning, organiztion, and results for the Capstone Project.

In essence we like to keep things as straight forward as possiable. Our goal here is to demostrate that we understand the data and that we are confident in making relevant connections.

We decided that a simple and very relevant business recommendation is what day of the week a movie should be released. Why is this important? Money, more specifically when it comes to box office sales/revenue.

When it comes to products, be it a pair of shoes, watch, or in this case a Movie, We want to release these products on the day that yields the most money.

Therefore, based on this goal we will go through the data and search for the desired day of the week that shows most box office sales. Very simple and straight forward.

Datasets and cleaning

In [23]:

```
# Load data
movie_info = pd.read_csv('../data/rt.movie_info.t
movie_info.head(10)
```

Out[23]:

	id	synopsis	rating	genre	di
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	W Fr
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	Crone
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	A A
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Le
4	7	NaN	NR	Drama|Romance	R Be
5	8	The year is 1942. As the Allies unite overseas...	PG	Drama|Kids and Family	Jay F
		Some cast and crew			

6	10	from NBC's highly acclaimed...	PG-13	Comedy	K
7	13	Stewart Kane, an Irishman living in the Austra...	R	Drama	Law
8	14	"Love Ranch" is a bittersweet love story that ...	R	Drama	Ha
9	15	When a diamond expedition in the Congo is lost...	PG-13	Action and Adventure|Mystery and Suspense|Scie...	Ma

In [24]:

```
# Basic info in the data
movie_info.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 [25]:

```
# making the box office data type a float and ge
movie_info['box_office'] = pd.to_numeric(movie_i
```

In [26]:

```
# turning the theater_date column into a datetim
movie_info['theater_date'] = pd.to_datetime(movi
```

In [27]:

```
# making sure that the datetime changed
movie_info.head()
```

Out[27]:

	id	synopsis	rating	genre	dir
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	W Fri
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	Crone
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	A A
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Lev
4	7	NaN	NR	Drama|Romance	Ro Be

In [28]:

```
# now we can create another column so that we can
# we can simply make a column with these days
movie_info['day_of_week'] = movie_info['theater_
```

In [29]:

```
# here we want to see the value count for each da
# this will give us a picture of what days have
movie_info['day_of_week'].value_counts()
```

Out[29]:

```
Friday       702
Wednesday    169
Thursday      95
Saturday      76
Monday        60
Tuesday       53
Sunday        46
Name: day_of_week, dtype: int64
```

In [30]:

```
# dropping null values in te box office column
movie_info = movie_info.dropna(subset=['box_offi
```

In [31]:

```
# making sure the nulls are gone
movie_info['box_office'].isnull().sum()
```

Out[31]:

```
0
```

In [32]:
```
# starting a new dataframe with the columns we wo
new_subset = movie_info[['theater_date', 'box_of
```

In [33]:
```
# resetting the index
new_subset.reset_index(drop=True, inplace=True)
```

In [34]:
```
# inspecting the null values that i have.
new_subset.isnull().sum()
```

Out[34]:
```
theater_date    6
box_office      0
day_of_week     6
dtype: int64
```

In [35]:
```
# dropping nulls
new_subset = new_subset.dropna(subset=['theater_d
```

In [36]:
```
new_subset = new_subset.dropna(subset=['day_of_we
```

In [37]:
```
# making a new dataframe with the desired columns
# also sorted them.
final_subset = new_subset[['theater_date', 'box_o
final_subset.sort_values(['box_office', 'day_of_w
```

In [38]:
```
# a way to remove this redundancy is to create a
# an index. this is called data normalization.
days = final_subset[['day_of_week', 'box_office'
days.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 331 entries, 205 to 169
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day_of_week  331 non-null    object 
 1   box_office   331 non-null    float64
dtypes: float64(1), object(1)
memory usage: 7.8+ KB
```

In [39]:
```
# since the index is unique we need to carry it o
# we will reset the index and rename it id
days.index.name = 'id'
my_id = days.reset_index()
my_id.head()
```

Out[39]:

	id	day_of_week	box_office
0	205	Friday	363.0
1	76	Friday	2367.0

2	145	Friday	3328.0
3	143	Wednesday	8300.0
4	278	Friday	8856.0

In [40]:

```
# merging together
pd.merge(my_id, days, on=['day_of_week', 'box_of
```

Out[40]:

	id	day_of_week	box_office
0	205	Friday	363.0
1	76	Friday	2367.0
2	145	Friday	3328.0
3	143	Wednesday	8300.0
4	278	Friday	8856.0

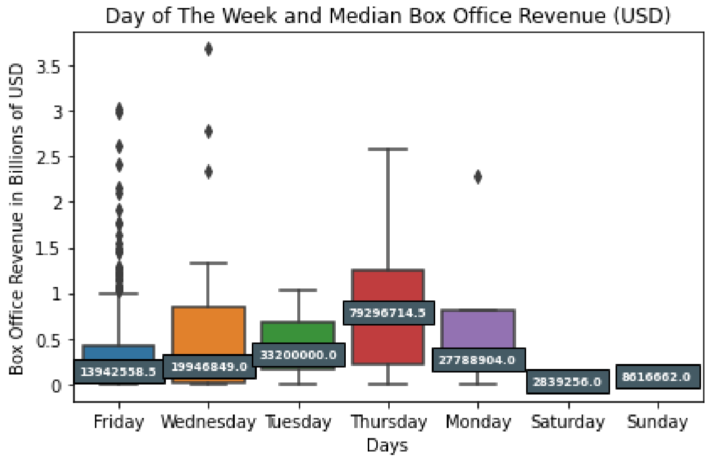
In [41]:

```
# heres a simplified table
tidy = pd.merge(my_id, days, on=['day_of_week',
tidy.sort_values(['box_office', 'day_of_week'], 
```

In [42]:

```
# boxplot visualizing days of the week with box 
box_plot = sns.boxplot(x="day_of_week", y="box_o
box_plot.set(title='Day of The Week and Median Bo

ax = box_plot.axes
lines = ax.get_lines()
categories = ax.get_xticks()
plt.xlabel('Days')
plt.ylabel('Box Office Revenue in Billions of USD

for cat in categories:
    # every 4th line at the interval of 6 is med
    # 0 -> p25 1 -> p75 2 -> lower whisker 3 -> 
    y = round(lines[4+cat*6].get_ydata()[0],1)

    ax.text(
        cat,
        y,
        f'{y}',
        ha='center',
        va='center',
        fontweight='bold',
        size=7,
        color='white',
        bbox=dict(facecolor='#445A64'))

# scale y axis to millions
scale_y = 1e8
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
# ax.set_ylim(-250000000, 1000000000)
```

```
box_plot.figure.tight_layout()
```



Analysis methods

Most of the analysis methods used were filters, and pandas functions. We also used datetime manipulation which made things easier.

We also used the box plot above to describe our findings. Overall, pretty straight forward.

Conclusion / Suggestion

Based on the findings on box office revenue and day of the week that the movie was released, Thursday is the best day to release a movie based on the median for box office revenue being the highest. Since the majority of the movies come out on Friday, this also has to be taken into account for maximizing revenue.

Therefore, we can recommend that Computing Vision should relese their movies on Thursday to maximize box office revenue. But should also take Friday into account as another great day to relaese a movie.

These findings are important because a company like Computing Vision, that is starting off in the movie industry should maximize those box office sales.

What limitations are there?

The only limitations we can think of is that the sample size is not big enough. This does not allow for more accurate testing.

The outliers for Friday were pretty significant as well. Compared to the other days, Friday's box is tight meaning that there is not a lot of varience compared with the other outliers.

Genre

Datasets and cleaning

The datasets used for this analysis were:

```
- im.db (from imdb.com - accessed
through sqlite3)
- tn.budgets.csv (from The-
Numbers.com)
```

Within the IMDB dataset, the tables movie_basics, and movie_reviews was called and merged on the primary key, "movie_id". The values in the genre column needed to be split on every comma, as many movies were classified under multiple genres. The entries were split within the column, and then for every genre in the column, a new row was created with the respective genre.

For example, if a *Lord of the Rings* movie was classified as an action *and* adventure film, *Lord of the Rings* would appear in two separate rows. The genre value would be "action" in one row, and "adventure" on the other. The resulting dataframe was trimmed so that it only contained the primary_title and genre column.

Next, the tn.budgets.csv file was loaded in using pandas. The financial columns were in a common monetary format as strings; this was rectified by using the "str.replace()" method, and then casting the columns as numeric. A new "total_profit" column was created by subtracting the production budget from the worldwide gross. This inevitably creates an *estimate* of the total_profit, as it is unknown if the entire budget was utilized. As a result, the total_profit is expected to be *at least* the current value.

To use information that otherwise is not very telling, the two tables were merged with the keys being "primary_title" and "movie". The merged table would pair the "total_profit" column to each respective movie. After the merge, the resulting dataframe had 7,417 entries (which, as a reminder, is *not* equal to the number of movies).

Bringing back the example from above, *Lord of the Rings* would appear in two separate rows as follows:

```
| primary_title      | genre     |
total_profit |
|*Lord of the Rings* | Action    |
340000000    |
|*Lord of the Rings* | Adventure |
340000000    |
```

Using this data structure, it was possible to create box plots for each genre and compare them to one another.

Analysis methods

In [43]:
```
# establish connection to the database
conn = sqlite3.connect('../data/im.db')

# read in the various tables from the database
tables = pd.read_sql("""SELECT name FROM sqlite_r
tables
```

Out[43]:

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

In [44]:
```
# Look into movie basics
mb_mr = pd.read_sql("""
    SELECT *
```

```
    FROM movie_basics""", conn)
mb_mr
```

Out[44]:

	movie_id	primary_title	original_title	start_year
0	tt0063540	Sunghursh	Sunghursh	2013
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017
...	...	...	...	...
146139	tt9916538	Kuambil Lagi Hatiku	Kuambil Lagi Hatiku	2019
146140	tt9916622	Rodolpho Teóphilo - O Legado de um Pioneiro	Rodolpho Teóphilo - O Legado de um Pioneiro	2015
146141	tt9916706	Dankyavar Danka	Dankyavar Danka	2013
146142	tt9916730	6 Gunn	6 Gunn	2017
146143	tt9916754	Chico Albuquerque - Revelações	Chico Albuquerque - Revelações	2013

146144 rows × 6 columns

In [45]:

```
# Checking to see if any movie id appears more th
pd.read_sql("""
    SELECT *
    FROM movie_basics
    GROUP BY movie_id
    HAVING COUNT(movie_id) > 1;
""", conn)
```

Out[45]:

movie_id	primary_title	original_title	start_year	runtim

In [46]:

```
# Split genres and create a new entry for each of
### FROM Leo's EDA notebook
s = mb_mr['genres'].str.split(',').apply(Series,
s.index = s.index.droplevel(-1)
```

```
s.name = 'genre'
del mb_mr['genres']
mb_mr_genres = mb_mr.join(s)
```

In [47]:

```
mb_mr_genres = mb_mr_genres[['primary_title', 'g
mb_mr_genres.head(20)
```

Out[47]:

	primary_title	genre
0	Sunghursh	Action
0	Sunghursh	Crime
0	Sunghursh	Drama
1	One Day Before the Rainy Season	Biography
1	One Day Before the Rainy Season	Drama
2	The Other Side of the Wind	Drama
3	Sabse Bada Sukh	Comedy
3	Sabse Bada Sukh	Drama
4	The Wandering Soap Opera	Comedy
4	The Wandering Soap Opera	Drama
4	The Wandering Soap Opera	Fantasy
5	A Thin Life	Comedy
6	Bigfoot	Horror
6	Bigfoot	Thriller
7	Joe Finds Grace	Adventure
7	Joe Finds Grace	Animation
7	Joe Finds Grace	Comedy
8	O Silêncio	Documentary
8	O Silêncio	History
9	Nema aviona za Zagreb	Biography

Load in tn.movie.budgets.csv

Cleaning

In [48]:

```
budgets = pd.read_csv("../data/tn.movie_budgets.
budgets.head()
```

Out[48]:

	id	release_date	movie	production_budget	domes
0	1	Dec 18, 2009	Avatar	$425,000,000	$76

1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$24
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$4
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$45
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$62

In [49]:

```
# removing $ and , from gross revenue
budgets['worldwide_gross'] = budgets['worldwide_g
budgets['worldwide_gross'] = budgets['worldwide_g
budgets['production_budget'] = budgets['producti
budgets['production_budget'] = budgets['producti

# casting the values as integers
budgets['production_budget'] = pd.to_numeric(budg
budgets['worldwide_gross'] = pd.to_numeric(budget

# calculating total profit
budgets['total_profit'] =  budgets['worldwide_gro

# Keep only movie and total_profit columns
budgets = budgets[['movie', 'total_profit']]

# confirmation
budgets.head()
```

Out[49]:

	movie	total_profit
0	Avatar	2351345279
1	Pirates of the Caribbean: On Stranger Tides	635063875
2	Dark Phoenix	-200237650
3	Avengers: Age of Ultron	1072413963
4	Star Wars Ep. VIII: The Last Jedi	999721747

In [50]:

```
# merge mb_mr_genres with budgets table on movie
combined = pd.merge(mb_mr_genres, budgets, left_o
combined.drop(columns="movie", inplace=True)
combined
```

Out[50]:

	primary_title	genre	total_profit
0	Foodfight!	Action	-44926294

1	Foodfight!	Animation	-44926294
2	Foodfight!	Comedy	-44926294
3	Mortal Kombat	Action	102133227
4	Mortal Kombat	Adventure	102133227
...	...	...	...
7863	Traitor	Action	5882226
7864	Traitor	Drama	5882226
7865	Traitor	Romance	5882226
7866	Ray	Crime	84823094
7867	Sublime	Documentary	-1800000

7868 rows × 3 columns

The duplicate values in the rows are okay to have for what I am going to accomplish with them. I will create a boxplot where the x is the genre categories, and the y is the total_profit column associated with those genres. Before proceeding to that, I need to clean up the new table and deal with outliers.

Keeping populated genres

In [51]:

```
# Getting 25th percentile of genre counts... the
combined['genre'].value_counts().describe()
```

Out[51]:

```
count      23.000000
mean      338.956522
std       398.313887
min         1.000000
25%        81.000000
50%       229.000000
75%       452.500000
max      1817.000000
Name: genre, dtype: float64
```

In [52]:

```
# remove any genre where the count is lower than
mask = combined['genre'].value_counts() > 100

vals_to_keep = []
for x in mask.items():
    if x[1] == True:
        vals_to_keep.append(x[0])

vals_to_keep
```

Out[52]:

```
['Drama',
```

```
'Comedy',
'Action',
'Thriller',
'Documentary',
'Adventure',
'Horror',
'Crime',
'Romance',
'Mystery',
'Biography',
'Sci-Fi',
'Family',
'Fantasy',
'Animation']
```

In [53]:
```
# Create table where we've kept rows where the va
# matched the vals_to_keep
combined = combined.loc[combined['genre'].isin(va
combined
```

Out[53]:

	primary_title	genre	total_profit
0	Foodfight!	Action	-44926294
1	Foodfight!	Animation	-44926294
2	Foodfight!	Comedy	-44926294
3	Mortal Kombat	Action	102133227
4	Mortal Kombat	Adventure	102133227
...	...	...	...
7863	Traitor	Action	5882226
7864	Traitor	Drama	5882226
7865	Traitor	Romance	5882226
7866	Ray	Crime	84823094
7867	Sublime	Documentary	-1800000

7417 rows × 3 columns

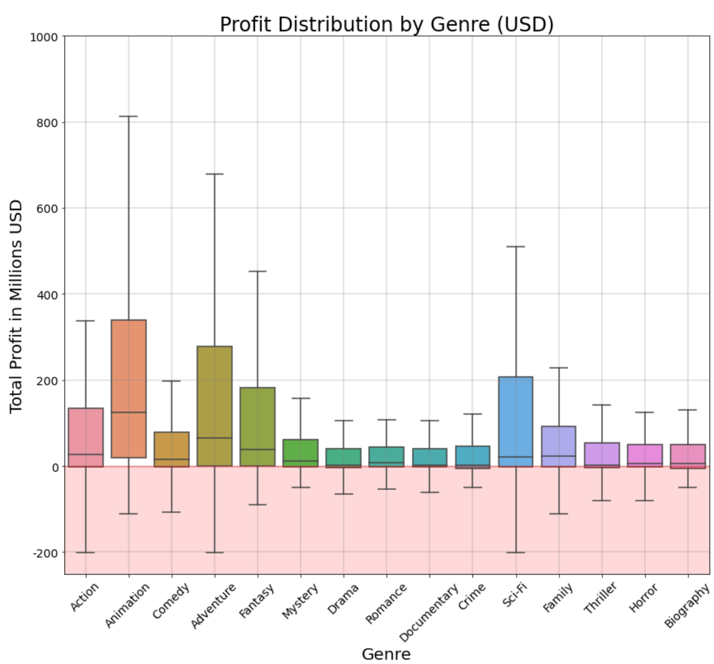
Charting

Dealing with outliers using showfliers=False

In [54]:
```
from matplotlib.ticker import FuncFormatter

fig, ax = plt.subplots(figsize = (14,12))
sns.boxplot(ax=ax, x=combined['genre'],
            y=combined['total_profit'],
            showfliers=False)
```

```
plt.style.use('seaborn-muted')

# scale y axis to millions
scale_y = 1e6
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
ax.set_ylim(-250000000, 1000000000)

# add title/labels/ticks/grid
ax.set_title('Profit Distribution by Genre (USD)
plt.xlabel("Genre", size = 20)
plt.ylabel("Total Profit in Millions USD", size=
plt.yticks(fontsize=14)
plt.xticks(rotation=45, fontsize=14)
plt.grid(color='gray', linestyle='-', linewidth=

# create red area in negative y
plt.axhline(y=[-1000000], alpha=0.3, color='red'
plt.axhspan(-2500000000, 0, alpha=0.15, color='re

# saves fig
#plt.savefig('../resources/charts/prof_genre_box
;
```

Out[54]: ''



Narrow down genres

In [55]:

```
# look at what genres have the top median profits
med_combined = combined.groupby('genre', sort='to
med_combined = med_combined.sort_values(by='total
med_combined_8 = med_combined.head(8)
med_combined_8
```

Out[55]:

	total_profit
genre	

Animation	124790560.0
Adventure	65979147.5
Fantasy	38189217.5
Action	27548508.5
Family	22741345.0
Sci-Fi	21517819.0
Comedy	15223306.5
Mystery	12417298.0

In [56]:

```
# Preparing to keep only the top 8 median genres
vals_to_keep = []
for x in med_combined_8.iterrows():
    vals_to_keep.append(x[0])

vals_to_keep
```

Out[56]:

```
['Animation',
 'Adventure',
 'Fantasy',
 'Action',
 'Family',
 'Sci-Fi',
 'Comedy',
 'Mystery']
```

In [57]:

```
# Create table where we've kept rows where the va
# matched the vals_to_keep
top_8_combined = combined.loc[combined['genre'].i
top_8_combined
```

Out[57]:

	primary_title	genre	total_profit
0	Foodfight!	Action	-44926294
1	Foodfight!	Animation	-44926294
2	Foodfight!	Comedy	-44926294
3	Mortal Kombat	Action	102133227
4	Mortal Kombat	Adventure	102133227
...	...	...	...
7847	What Lies Beneath	Mystery	198693989
7851	Sugar Town	Sci-Fi	-71905
7852	Invincible	Action	18501127
7853	What Just Happened	Comedy	-24587877
7863	Traitor	Action	5882226

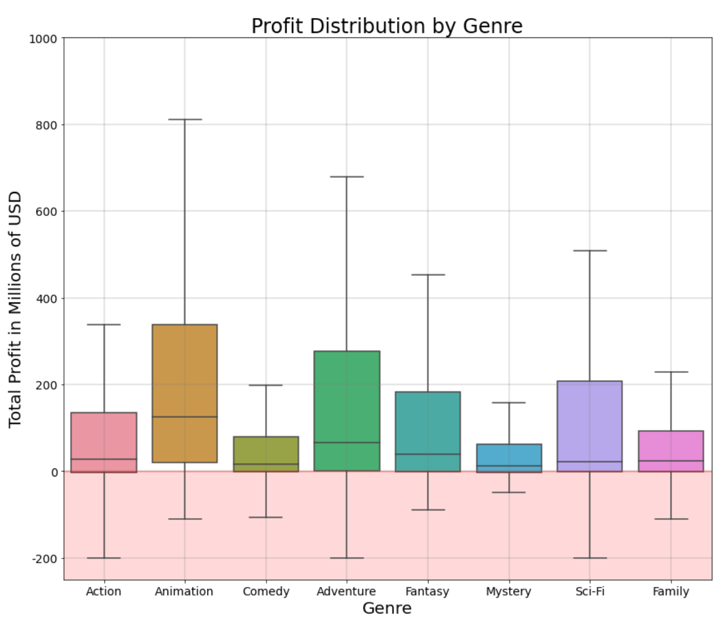
3074 rows × 3 columns

Charting the 8 genres with the highest median profit

In [58]:

```
fig, ax = plt.subplots(figsize = (14,12))
sns.boxplot(ax=ax, x=top_8_combined['genre'],
            y=top_8_combined['total_profit'],
            showfliers=False)

plt.style.use('seaborn-muted')

# scale y axis to millions
scale_y = 1e6
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
ax.set_ylim(-250000000, 1000000000)

# title/labels/ticks
ax.set_title('Profit Distribution by Genre', size
plt.xlabel("Genre", size = 20)
plt.ylabel("Total Profit in Millions of USD", si
plt.xticks(rotation=0, fontsize=14)
plt.yticks(fontsize=14)
plt.grid(color='gray', linestyle='-', linewidth=

# adding negative y color
plt.axhline(y=[-1000000], alpha=0.3, color='red',
plt.axhspan(-250000000, 0, alpha=0.15, color='red

# save fig
#plt.savefig('../resources/charts/top8_prof_genre
;
```

Out[58]: ''



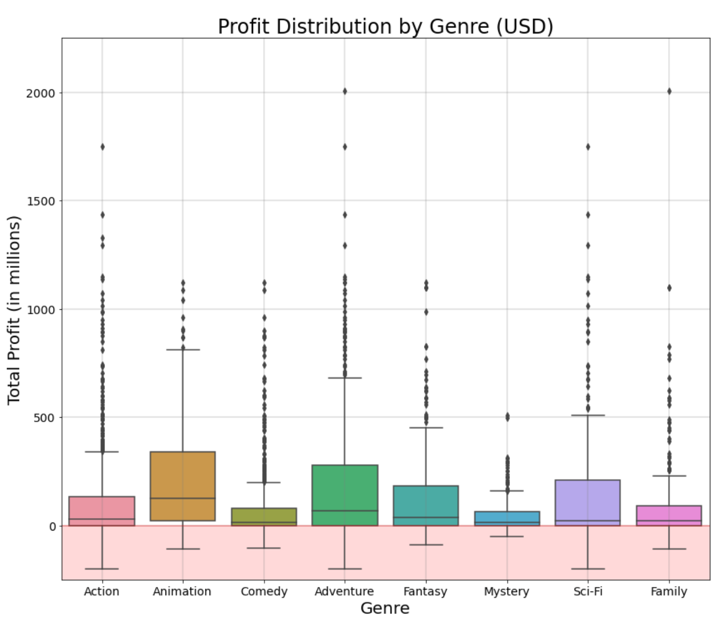
Charting the top 8 WITH

Charting the top 8 with outliers

In [59]:

```
fig, ax = plt.subplots(figsize = (14,12))
sns.boxplot(ax=ax, x=top_8_combined['genre'],
            y=top_8_combined['total_profit'],
            showfliers=True)

plt.style.use('seaborn-muted')

# scale y axis to millions
scale_y = 1e6
ticks_y = FuncFormatter(lambda x, pos: '{0:g}'.fo
ax.yaxis.set_major_formatter(ticks_y)
ax.set_ylim(-250000000, 2250000000)

# title/labels/ticks
ax.set_title('Profit Distribution by Genre (USD)
plt.xlabel("Genre", size = 20)
plt.ylabel("Total Profit (in millions)", size=20
plt.xticks(rotation=0, fontsize=14)
plt.yticks(fontsize=14)
plt.grid(color='gray', linestyle='-', linewidth=

# adding negative y color
plt.axhline(y=[-1000000], alpha=0.3, color='red'
plt.axhspan(-2500000000, 0, alpha=0.15, color='r

# saves fig
#plt.savefig('../resources/charts/top8_outlier_pr
;
```

Out[59]: ''



Creating a small table with top 8 performing genres that displays their median income

In [60]:

```
# rename total profit column to median profit
```

```
med_combined_8 = med_combined_8.rename({'total_pr
                                        axis=1)
med_combined_8
```

Out[60]:

genre	median_profit
Animation	124790560.0
Adventure	65979147.5
Fantasy	38189217.5
Action	27548508.5
Family	22741345.0
Sci-Fi	21517819.0
Comedy	15223306.5
Mystery	12417298.0

Conclusion / Suggestion

Based off of the boxplots, two genres stand out based on their median profit: Animation and Adventure. These genres also appear to have the highest variance as they are stretched (or tall, if you will).

It also appears there is a positive skew, as the distance from the median to the upper quartile is much greater than the distance to the lower quartile. This positive skew means that most movies, within their respective genre, had a positive net income, but some generate extraordinarily higher amounts. It is not to say that genre is an indicator of profit, but just an observation of the current data set.

These findings are merely a surface level analysis, and therefore not conclusive. Going forward, we will utilize statistical methods to dig deeper into our observations.

Side Note

Aside from statistical analysis, a next step could be creating a profit to budget ratio. Lower budget films with a consistently higher profit to budget ratio could be an worthwhile investment as a lower-cost entry point as new studio.

Genre's Association with Profit (Chi Square Analysis)

The Business Question

Does the genre of a movie have any significant association with the movie's profitability?

The Datasets

- Movie budgets dataset from the-numbers.com including movie titles, production budget, and worldwide gross revenue with 5,782 rows.
- TheMovieDB dataset including movie titles and genres with 26,517 rows.

The Methods

Import and Clean Data

The relevant datasets for our analysis were the tn.movie_budgets.csv and tmdb.movies.csv files.

In [61]:
```
budgets = pd.read_csv("../data/tn.movie_budgets.c
tmdb = pd.read_csv("../data/tmdb.movies.csv", in
```

Before running our analysis, we needed to review the contents of the datasets, isolate relevant columns, and clean data as necessary.

First, we looked at the first few rows of the movie budgets dataframe in order to get an idea of the columns, potential datatypes, and areas which may require pre-processing and cleaning.

In [62]:
```
budgets.head()
```

Out[62]:

	id	release_date	movie	production_budget	domes
0	1	Dec 18, 2009	Avatar	$425,000,000	$76
1	2	May 20,	Pirates of the Caribbean:	$410,600,000	$24

1	2	2011	On Stranger Tides	$410,600,000	$24
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$4
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$45
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$62

From this dataframe, we decided that we needed to retain the following columns:

- Movie (for joining with other dataframes)
- Production Budget and Worldwide Gross (for calculating profit)

Thus, we formed a subset of the dataset including only the relevant columns.

In [63]:

```
cols_to_keep = ['movie','production_budget','wor
budgets_relevant = budgets[cols_to_keep]
```

We also noted that the production budget and worldwide gross columns were populated with strings (as evident by the symbolic characters used alongside the numeric characters, such as "$"). These values needed to be cleaned and cast as integers before they could be used to calculate profit.

However, before doing any further cleaning we looked for null values and duplicates so that we could avoid making any unnecessary calculations. There were no obvious nulls in the dataframe. However, looking at the values stored within the worldwide_gross column brought to light some null-esque values. Namely, movies with a worldwide gross revenue of $0. There were some zeroes in this column, presumably because there was no available data on its gross revenue. We originally removed these observations from the dataframe. However, we discovered later on that this resulted in a very small sample size when coupled with the unavoidable loss of other observations. So rather than removing these observations, we decided to replace them with the median once the column had been properly cleaned. We decided to use the median rather than the mean for imputation because this was a highly skewed dataset with outliers that would significantly impact the mean, but that the median would be more resilient against.

From here, we moved on to locating duplicate values. There were no obvious duplicate rows. However, we realized that multiple movies could have the same title. This would pose an issue when we needed to join dataframes using movie titles as the mutual column.

In [64]:

```
def get_title_counts(data, col):

    '''
    Inputs: Dataframe and movie title column
    Outputs: Dataframe of movie titles which appe

    '''

    # set the value counts as a dataframe
    title_counts = pd.DataFrame(data[col].value_

    # reset index so that we can easily access th
    title_counts = title_counts.reset_index()

    # take a subset of the title counts datafram
    title_counts = title_counts.loc[ title_count

    # return this dataframe of title duplicates
    return title_counts

title_counts = get_title_counts(budgets_relevant
print(f"Number of repeated titles: {len(title_co
```

```
Number of repeated titles: 81
```

There were 81 movies with repeated titles in the dataframe. 81 rows out of a 5,000+ row dataset didn't seem substantial enough to justify an attempted mutli-column merge. So, we decided to simply drop the duplicates.

In [65]:

```
import warnings
warnings.filterwarnings(action = 'ignore', categ

def remove_duplicate_titles(data, col):
```

```
    '''
    Inputs: Dataframe and movie title column
    Outputs: The same dataframe without movies tl

    '''

    # get the dataframe for titles which appear i
    title_counts = get_title_counts(data,col)

    # create a dichotomous column for which there
    data['duplicate'] = data[col].map(lambda x: 1

    # take a subset of the dataframe of only non-
    data = data.loc[ data['duplicate'] == 0]

    # initialize a list of columns to maintain
    keepers = []

    # for each column in the dataframe
    for col in data.columns:
        # if it isn't the duplicate column
        if col != "duplicate":
            # add it to the list of columns to be
            keepers.append(col)

    # keep only the columns intended
    data = data[keepers]

    return data

budgets_relevant = remove_duplicate_titles(budge
```

```
<ipython-input-65-aca4b65cdd55>:16: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice
from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value i
nstead

See the caveats in the documentation: https://pan
das.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  data['duplicate'] = data[col].map(lambda x: 1 i
f any([movie in x for movie in list(title_counts
['index'])]) else 0)
```

After checking for nulls and duplicates, we cleaned the budget and gross revenue columns so that we could eventually use them to calculate profit.

In [66]:

```
def dollar_to_numeric(column):

    '''
    Inputs: Column containing USD strings
    Outputs: The contents of the column as intege

    '''
```

```
    # removing $ and , from string
    column = column.str.replace(",","")
    column = column.str.replace("$","")

    # casting the values as integers
    column = pd.to_numeric(column)

    return column

budgets_relevant['worldwide_gross'] = dollar_to_i
budgets_relevant['production_budget'] = dollar_to
```

After successfully casting the data as integers, we replaced all zeroes in the worldwide_gross column with the median worldwide gross revenue.

In [67]:

```
median_gross = budgets_relevant['worldwide_gross
budgets_relevant['worldwide_gross'] = budgets_re
```

Now that we had two clean revenue and cost columns to work with, we used this information to create a new column in the dataframe for the calculated profit.

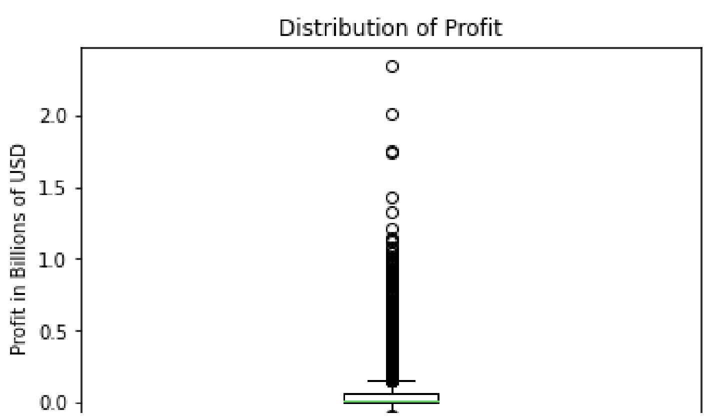
In [68]:

```
# calculating total profit
budgets_relevant['total_profit'] =  budgets_rele

budgets_relevant = budgets_relevant[['movie','to
```

Next, we decided to visualize the distribution of profit and examine it's summary statistics since this is our dependent variable.

In [69]:

```
fig, ax = plt.subplots()
ax.boxplot(budgets_relevant['total_profit'])
ax.set_title("Distribution of Profit")
ax.set_ylabel("Profit in Billions of USD")
plt.xticks([])
ax.yaxis.get_offset_text().set_visible(False);
```



In [70]:

```
budgets_relevant['total_profit'].describe()
```

Out[70]:

```
count    5.459000e+03
mean     6.004190e+07
std      1.431806e+08
min     -2.002376e+08
25%     -1.394899e+06
50%      1.415149e+07
75%      5.964086e+07
max      2.351345e+09
Name: total_profit, dtype: float64
```

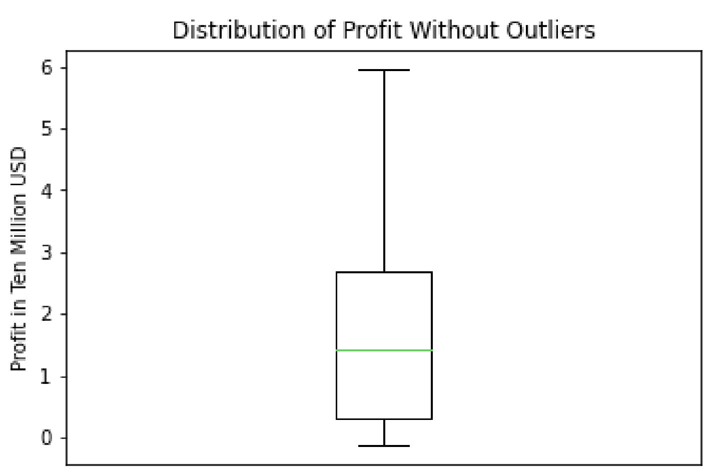
It was evident that there were some extremely profitable (and extremely unprofitable) movies that may influence the results of our analysis. Because of the presense of extremes, we decided to remove any movies with profits outside of the interquartile range. Then, we revisualized the distribution and reexamined the summary statistics.

In [71]:

```
import scipy.stats as stats
```

In [72]:

```
# find Q1, Q3, and interquartile range for each (

Q1 = budgets_relevant['total_profit'].quantile(q
Q3 = budgets_relevant['total_profit'].quantile(q
IQR = budgets_relevant['total_profit'].apply(sta

budgets_relevant = budgets_relevant.loc[~((budget

fig, ax = plt.subplots()
ax.boxplot(budgets_relevant['total_profit'])
ax.set_title("Distribution of Profit Without Outl
ax.set_ylabel("Profit in Ten Million USD")
plt.xticks([])
ax.yaxis.get_offset_text().set_visible(False);
```



In [73]:
```
budgets_relevant['total_profit'].describe()
```

Out[73]:
```
count     2.729000e+03
mean      1.738068e+07
std       1.598187e+07
min      -1.391430e+06
25%       2.951247e+06
50%       1.415149e+07
75%       2.684288e+07
max       5.963504e+07
Name: total_profit, dtype: float64
```

Knowing that we would eventually have to merge this dataframe with the TMDB dataframe, we also set the index to the column on which we wanted to merge (the movie title).

In [74]:
```
budgets_relevant.set_index('movie', inplace = Tru
```

With this dataframe cleaned, we moved on to the TMDB dataframe.

Just like the first dataframe, we began by looking at the first few rows to get an idea of the columns, datatypes, and areas which may require preprocessing/cleaning.

In [75]:
```
tmdb.head()
```

Out[75]:

	genre_ids	id	original_language	original_title	popu
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	
2	[12, 28, 878]	10138	en	Iron Man 2	
3	[16, 35, 10751]	862	en	Toy Story	
4	[28, 878, 12]	27205	en	Inception	

From this dataframe, we noted that we only needed the following columns:

- Title (for merging)
- Genre_ids

We started with dropping the irrelevant columns.

In [76]:
```
cols_to_keep = ['title','genre_ids']
tmdb_relevant = tmdb[cols_to_keep]
```

We noted that the genre_ids column appeared to contain lists of multiple ids associated with specific genres. We needed to clean this column and replace these numbers with their associated genre. However, we decided to wait to replace these values until after the dummy columns were created because it would be easier to rename a small number of columns than replace multiple numbers in every cell with its associated genre.

So for now, we moved on to locating null values and duplicates. There didn't appear to be any null values in the dataset. However, there were 1,088 duplicates which we dropped. There were also duplicate titles in this dataframe which we handled the same as those in the budgets dataframe.

In [77]:
```
# Drop duplicates
tmdb_relevant = tmdb_relevant.drop_duplicates()
tmdb_relevant = remove_duplicate_titles(tmdb_rel
```

After dropping these duplicate values, we set the movie titles as in the index in preparation for merging these two dataframes. The final dataframe contained 17,714 rows.

In [78]:
```
tmdb_relevant = tmdb_relevant.set_index('title')
```

Now that we had cleaned the data, it was ready to be merged.

In [79]:
```
budgets_and_tmdb = budgets_relevant.join(tmdb_re
```

After merging the dataframes, we were left with a much smaller dataframe than either of the parent datasets. This, however, was expected given there was no gaurantee that the datasets would overlap in their contents significantly nor was there a gaurantee that there would not be any spelling errors in the titles that would prevent a successful join for at least some rows. We decided to proceed with the knowledge that 656 movies retained the potential to provide some useful insights.

Next, we cleaned the genre id column and isolated each genre id, using the results to create dummy columns.

In [80]:
```
def create_dummy_cols(data, col):

    '''
    Inputs: Dataframe and column where the column
    Outputs: The same dataframe with dummy column
    '''

    # remove [, ], and whitespace
    data[col] = data[col].str.strip("]")
    data[col] = data[col].str.strip("[")
    data[col] = data[col].str.replace(" ", "")

    # split genre ids by commas
    genre_ids = data[col].str.split(",")

    # create the binary dummy columns
    bin_genre_df = pd.get_dummies(genre_ids.apply
    budgets_and_genre_dummys = data.join(bin_genr

    # rename columns for genres
    budgets_and_genre_dummys.rename(columns = {'
                                              '12'
                                             '16' :
                                             '35' :
                                             '80' :
                                             '99' :
                                             '18' :
                                             '10751
                                             '14' :
                                             '36' :
                                             '27' :
                                             '10402
                                             '9648'
                                             '10749
                                             '878'
                                             '10770
                                             '53' :
                                             '10752
                                             '37' :
    return budgets_and_genre_dummys
```

```
budgets_and_genre_dummys = create_dummy_cols(bud
budgets_and_genre_dummys.head()
```

Out[80]:

	total_profit	genre_ids	Music	Romance
John Carter	7778100	28,12,878	0	0	0
Green Lantern	19535492	12,28,53,878	0	0	0
Jack the Giant Slayer	2687603	28,12,10751,14	0	0	0
Hugo	47784	12,18,10751	0	0	0
Valerian and the City of a Thousand Planets	35098356	12,878,28	0	0	0

5 rows × 22 columns

There was a dummy column seemingly associated with no genre. This appeared to be the result of 11 titles which did not have any associated genre ids. So, we dropped them from analysis.

In [81]:

```
budgets_and_genre_dummys = budgets_and_genre_dumm
```

Chi-Square Analysis

We decided to turn profit into a categorical variable denoting high vs. medium vs. low profit so that we could perform our chi-square analysis (which required that both variables of interest be categorical). The threshold values for these categories were decided as:

- High profit = Profit at the 75th percentile and greater
- Medium = Profit greater than 25th percentile and lower than 75th percentile.
- Low = Profit at the 25th percentile and lower

In [82]:

```
# define thresholds
iqr_Q1 = budgets_and_genre_dummys['total_profit'
iqr_Q3 = budgets_and_genre_dummys['total_profit'
```

```
# assign categories based on thresholds
```