# MICROSOFT MOVIE STUDIO:

Justin Lapidus

Task: Microsoft has been noticing competition branching out into the movie world. They are seeing the loss of potential earnings the longer they sit out of the competition. They have been assigned myself with analyzing data and coming up with recommendations to produce movies that will be popular, make money, and create a successful new division of Microsoft

## Data

Data was, mainly, collected using three different, but reputable sources. They were collected from "The Numbers", Rotten Tomatoes, and IMDB.

IMDB:an online database containing information and statistics about movies, TV shows and video games as well as actors, directors and other film industry professionals

Rotten Tomatoes: Rotten Tomatoes and the Tomatometer score are the world's most trusted recommendation resources for quality entertainment. As the leading online aggregator of movie and TV show reviews from critics, we provide fans with a comprehensive guide to what's Fresh – and what's Rotten – in theaters and at home.

The Numbers: The Numbers provides detailed movie financial analysis, including box office, DVD, and Blu-ray sales reports, and release schedules.

# IMPORTING DATA FROM FILES:

In [231]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [187]:
```
import sqlite3
import pandas as pd
conn = sqlite3.connect("im.db")
cur = conn.cursor()
```

In [229]:
```
import gzip
with gzip.open('rt.Reviews.tsv.gz', 'rb', ) as f:
    RT_reviews = f.read
```

In [153]:
```
%%bash
sqlite3 im1.db
.tables
```

In [188]:
```
pd.read_sql('''SELECT
    name
FROM
    sqlite_schema
WHERE
    type ='table' AND
    name NOT LIKE 'sqlite_%'; '''
           , conn)
```

Out[188]:

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

In [189]:
```
movie_basics = pd.read_sql(
'''
SELECT *
FROM movie_basics
''', conn)
movie_basics
```

Out[189]:

| | movie_id | primary_title | original_title | start_year | runtime_minutes | genres |
|---|---|---|---|---|---|---|
| **0** | tt0063540 | Sunghursh | Sunghursh | 2013 | 175.000 | Action,Crime,Drama |
| **1** | tt0066787 | One Day Before the Rainy Season | Ashad Ka Ek Din | 2019 | 114.000 | Biography,Drama |
| **2** | tt0069049 | The Other Side of the Wind | The Other Side of the Wind | 2018 | 122.000 | Drama |
| **3** | tt0069204 | Sabse Bada Sukh | Sabse Bada Sukh | 2018 | NaN | Comedy,Drama |
| **4** | tt0100275 | The Wandering Soap Opera | La Telenovela Errante | 2017 | 80.000 | Comedy,Drama,Fantasy |
| **...** | ... | ... | ... | ... | ... | ... |
| **146139** | tt9916538 | Kuambil Lagi Hatiku | Kuambil Lagi Hatiku | 2019 | 123.000 | 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 | 116.000 | None |
| **146143** | tt9916754 | Chico Albuquerque - Revelações | Chico Albuquerque - Revelações | 2013 | NaN | Documentary |

146144 rows × 6 columns

In [190]:
```
directors = pd.read_sql(
'''
SELECT *
FROM directors
''', conn)
directors
```

Out[190]:

| | movie_id | person_id |
|---|---|---|
| **0** | tt0285252 | nm0899854 |
| **1** | tt0462036 | nm1940585 |
| **2** | tt0835418 | nm0151540 |
| **3** | tt0835418 | nm0151540 |
| **4** | tt0878654 | nm0089502 |
| **...** | ... | ... |
| **291169** | tt8999974 | nm10122357 |
| **291170** | tt9001390 | nm6711477 |
| **291171** | tt9001494 | nm10123242 |
| **291172** | tt9001494 | nm10123248 |
| **291173** | tt9004986 | nm4993825 |

291174 rows × 2 columns

In [191]:
```
known_for = pd.read_sql(
'''
SELECT *
FROM known_for
''', conn)
known_for
```

Out[191]:

| | person_id | movie_id |
|---|---|---|
| **0** | nm0061671 | tt0837562 |
| **1** | nm0061671 | tt2398241 |
| **2** | nm0061671 | tt0844471 |
| **3** | nm0061671 | tt0118553 |
| **4** | nm0061865 | tt0896534 |
| **...** | ... | ... |
| **1638255** | nm9990690 | tt9090932 |
| **1638256** | nm9990690 | tt8737130 |
| **1638257** | nm9991320 | tt8734436 |
| **1638258** | nm9991320 | tt9615610 |
| **1638259** | nm9993380 | tt8743182 |

1638260 rows × 2 columns

In [158]:
```
movie_akas = pd.read_sql(
'''
SELECT *
FROM movie_akas
''', conn)
movie_akas
```

Out[158]:

| | movie_id | ordering | title | region | language | types | attributes | is_original_title |
|---|---|---|---|---|---|---|---|---|
| **0** | tt0369610 | 10 | Джурасик свят | BG | bg | None | None | 0.000 |
| **1** | tt0369610 | 11 | Jurashikku warudo | JP | None | imdbDisplay | None | 0.000 |
| **2** | tt0369610 | 12 | Jurassic World: O Mundo dos Dinossauros | BR | None | imdbDisplay | None | 0.000 |
| **3** | tt0369610 | 13 | O Mundo dos Dinossauros | BR | None | None | short title | 0.000 |
| **4** | tt0369610 | 14 | Jurassic World | FR | None | imdbDisplay | None | 0.000 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **331698** | tt9827784 | 2 | Sayonara kuchibiru | None | None | original | None | 1.000 |
| **331699** | tt9827784 | 3 | Farewell Song | XWW | en | imdbDisplay | None | 0.000 |
| **331700** | tt9880178 | 1 | La atención | None | None | original | None | 1.000 |
| **331701** | tt9880178 | 2 | La atención | ES | None | None | None | 0.000 |
| **331702** | tt9880178 | 3 | The Attention | XWW | en | imdbDisplay | None | 0.000 |

331703 rows × 8 columns

In [192]:
```
persons = pd.read_sql(
'''
SELECT *
FROM persons
''', conn)
persons
```

Out[192]:

| | person_id | primary_name | birth_year | death_year | primary_prof |
|---|---|---|---|---|---|
| **0** | nm0061671 | Mary Ellen Bauder | NaN | NaN | miscellaneous,production_manager,pr |
| **1** | nm0061865 | Joseph Bauer | NaN | NaN | composer,music_department,sound_depa |
| **2** | nm0062070 | Bruce Baum | NaN | NaN | miscellaneous,acto |
| **3** | nm0062195 | Axel Baumann | NaN | NaN | camera_department,cinematographer,art_depa |
| **4** | nm0062798 | Pete Baxter | NaN | NaN | production_designer,art_department,set_de |
| **...** | ... | ... | ... | ... | |
| **606643** | nm9990381 | Susan Grobes | NaN | NaN | a |
| **606644** | nm9990690 | Joo Yeon So | NaN | NaN | a |
| **606645** | nm9991320 | Madeline Smith | NaN | NaN | a |
| **606646** | nm9991786 | Michelle Modigliani | NaN | NaN | pr |
| **606647** | nm9993380 | Pegasus Envoyé | NaN | NaN | director,acto |

606648 rows × 5 columns

In [193]:
```
principals = pd.read_sql(
'''
SELECT *
FROM principals
''', conn)
principals
```

Out[193]:

| | movie_id | ordering | person_id | category | job | characters |
|---|---|---|---|---|---|---|
| **0** | tt0111414 | 1 | nm0246005 | actor | None | ["The Man"] |
| **1** | tt0111414 | 2 | nm0398271 | director | None | None |
| **2** | tt0111414 | 3 | nm3739909 | producer | producer | None |
| **3** | tt0323808 | 10 | nm0059247 | editor | None | None |
| **4** | tt0323808 | 1 | nm3579312 | actress | None | ["Beth Boothby"] |
| **...** | ... | ... | ... | ... | ... | ... |
| **1028181** | tt9692684 | 1 | nm0186469 | actor | None | ["Ebenezer Scrooge"] |
| **1028182** | tt9692684 | 2 | nm4929530 | self | None | ["Herself","Regan"] |
| **1028183** | tt9692684 | 3 | nm10441594 | director | None | None |
| **1028184** | tt9692684 | 4 | nm6009913 | writer | writer | None |
| **1028185** | tt9692684 | 5 | nm10441595 | producer | producer | None |

1028186 rows × 6 columns

In [194]:
```
movie_ratings = pd.read_sql(
'''
SELECT *
FROM movie_ratings
''', conn)
movie_ratings
```

Out[194]:

| | movie_id | averagerating | numvotes |
|---|---|---|---|
| **0** | tt10356526 | 8.300 | 31 |
| **1** | tt10384606 | 8.900 | 559 |
| **2** | tt1042974 | 6.400 | 20 |
| **3** | tt1043726 | 4.200 | 50352 |
| **4** | tt1060240 | 6.500 | 21 |
| **...** | ... | ... | ... |
| **73851** | tt9805820 | 8.100 | 25 |
| **73852** | tt9844256 | 7.500 | 24 |
| **73853** | tt9851050 | 4.700 | 14 |
| **73854** | tt9886934 | 7.000 | 5 |
| **73855** | tt9894098 | 6.300 | 128 |

73856 rows × 3 columns

In [226]:
```
tmdb_movies= pd.read_csv('tmdb.movies.csv.gz')
```

In [196]:
```
Movie_Gross= pd.read_csv('bom.movie_gross.csv.gz')
Movie_Gross


Movie_budgets= pd.read_csv('tn.movie_budgets.csv.gz')

Movie_budgets
```

Out[196]:

| | 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 |
| **...** | ... | ... | ... | ... | ... | ... |
| **5777** | 78 | Dec 31, 2018 | Red 11 | $7,000 | $0 | $0 |
| **5778** | 79 | Apr 2, 1999 | Following | $6,000 | $48,482 | $240,495 |
| **5779** | 80 | Jul 13, 2005 | Return to the Land of Wonders | $5,000 | $1,338 | $1,338 |
| **5780** | 81 | Sep 29, 2015 | A Plague So Pleasant | $1,400 | $0 | $0 |
| **5781** | 82 | Aug 5, 2005 | My Date With Drew | $1,100 | $181,041 | $181,041 |

5782 rows × 6 columns

# Merging of Data

Merging of data to determine which movie studios generating the most money after taking the production_budget from the worldwide_gross data

In [323]:
```
grossbudget= pd.merge(Movie_Gross, Movie_budgets, left_on="title", right_on
```

In [342]:
```
grossbudget
```

Out[342]:

| | title | studio | domestic_gross_x | foreign_gross | year | id | release_date | movie | product |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Toy Story 3 | BV | 415000000.000 | 652000000 | 2010 | 47 | Jun 18, 2010 | Toy Story 3 | $2 |
| **1** | Inception | WB | 292600000.000 | 535700000 | 2010 | 38 | Jul 16, 2010 | Inception | $1 |
| **2** | Shrek Forever After | P/DW | 238700000.000 | 513900000 | 2010 | 27 | May 21, 2010 | Shrek Forever After | $1 |
| **3** | The Twilight Saga: Eclipse | Sum. | 300500000.000 | 398000000 | 2010 | 53 | Jun 30, 2010 | The Twilight Saga: Eclipse | $ |
| **4** | Iron Man 2 | Par. | 312400000.000 | 311500000 | 2010 | 15 | May 7, 2010 | Iron Man 2 | $1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1242** | Gotti | VE | 4300000.000 | NaN | 2018 | 64 | Jun 15, 2018 | Gotti | $ |
| **1243** | Ben is Back | RAtt. | 3700000.000 | NaN | 2018 | 95 | Dec 7, 2018 | Ben is Back | $ |
| **1244** | Bilal: A New Breed of Hero | VE | 491000.000 | 1700000 | 2018 | 100 | Feb 2, 2018 | Bilal: A New Breed of Hero | $ |
| **1245** | Mandy | RLJ | 1200000.000 | NaN | 2018 | 71 | Sep 14, 2018 | Mandy | |
| **1246** | Lean on Pete | A24 | 1200000.000 | NaN | 2018 | 13 | Apr 6, 2018 | Lean on Pete | |

1247 rows × 11 columns

In [358]:
```
grossbudget['production_budget']= grossbudget['production_budget'].str.stri
```

In [359]:
```
grossbudget['worldwide_gross']= grossbudget['worldwide_gross'].replace(',',
grossbudget['production_budget']= grossbudget['production_budget'].replace(
```

In [360]:
```
grossbudget['worldwide_gross']= grossbudget['worldwide_gross'].astype(int)
grossbudget['production_budget']= grossbudget['production_budget'].astype(i


Money= grossbudget['worldwide_gross']-grossbudget['production_budget']
Moneydivided= Money/(10**9)
```

In [397]:
```
grossbudgets = pd.merge(Movie_Gross, Movie_budgets, left_on="title", right_
```

In [259]:
```
RottenTomatoes= pd.merge(df4, df5, left_on="id", right_on = "id", how="inne
```

In [268]:
```
Movie_Gross
newmoviegross2= Movie_Gross.merge(newmoviedata, how='inner', left_on='title
```

In [310]:
```
Newdataframe= pd.merge(MovieGrossTMDB,newmoviedata, on='original_title', ho
```

In [272]:
```
newmoviegross= newmoviedata.merge(Movie_Gross, how='inner', left_on='origin
```

In [260]:
```
Newdataframe=pd.merge(MovieGrossTMDB,newmoviedata, left_on='original_title'
```

# Reccomendation one: Analyzing the top ten grossing movie studio.

In [394]:
```
studio= grossbudget['studio'].head(10)
```

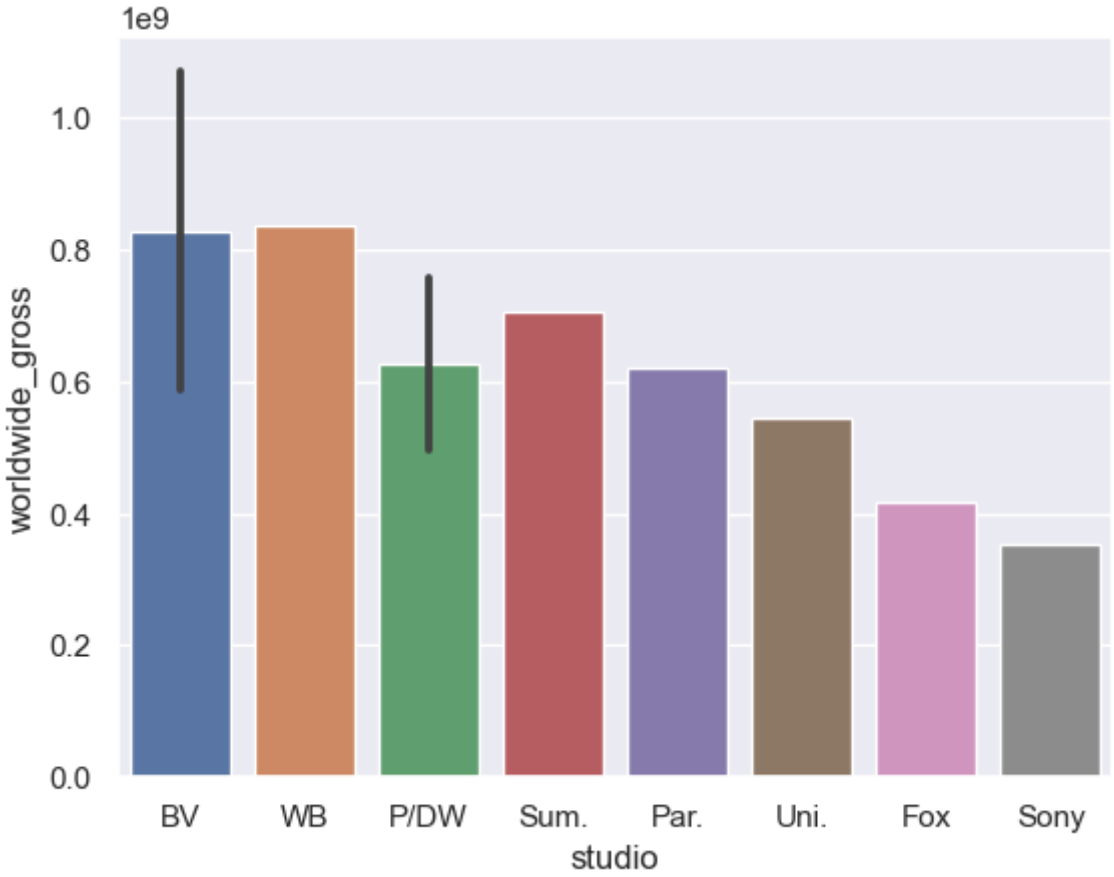
In [237]:
```
Newmovie=pd.merge(MovieGrossTMDB, movie_basics, right_on = "original_title"
```

In [262]:
```
MovieGrossTMDB=tmdb_movies.merge(Movie_Gross,how='left',left_on=["original_

#New Data frame combinbing two dataframes to idenify gross total of movies
```

In [238]:
```
Newmoviedata= pd.merge(Newmovie, Movie_budgets, left_on='original_title', r
```

In [414]:
```
sns.barplot(data=grossbudget, x=studio, y='worldwide_gross')

#Creation of a barplot to show relationship between studio and the money th
```

Out[414]: <Axes: xlabel='studio', ylabel='worldwide_gross'>




In [393]:
```
rtmovie_info= pd.read_csv('rt.movie_info.tsv.gz',sep='\t')
```

In [230]:
```
RT_reviews= pd.read_csv('rt.Reviews.tsv.gz',sep='\t', encoding = 'latin')
```

In [233]:
```
df4= RT_reviews
df5= rtmovie_info
```

In [261]:
```
tmdb_movies['year'] = pd.to_datetime(tmdb_movies['release_date']).dt.year
```

In [239]:
```
newmoviedata= Newmoviedata.drop(['original_language','title_x','start_year'
```

In [265]:
```
domestic_gross= newmoviedata.domestic_gross_y.head(50)
worldwide_gross=newmoviedata.worldwide_gross.head(50)
productionbudget= newmoviedata.production_budget.head(50)

#Looking at the top fifty earning movies from each column
```

In [266]:
```
newmoviedata['production_budget'] = newmoviedata['production_budget'].str.s
newmoviedata['worldwide_gross'] = newmoviedata['worldwide_gross'].str.strip


newmoviedata['worldwide_gross']=newmoviedata['worldwide_gross'].str.replace
newmoviedata['production_budget']=newmoviedata['production_budget'].str.rep



#Eliminating dilimiters in order to change from string to int
```

In [267]:
```
newmoviedata.dropna(subset='worldwide_gross')
```

Out[267]:

| | Unnamed: 0 | genre_ids | id_x | original_title | popularity | release_date_x | vote_average | vote |
|---|---|---|---|---|---|---|---|---|
| **1** | 1.000 | [14, 12, 16, 10751] | 10191.000 | How to Train Your Dragon | 28.734 | 2010-03-26 | 7.700 | 7 |
| **2** | 2.000 | [12, 28, 878] | 10138.000 | Iron Man 2 | 28.515 | 2010-05-07 | 6.800 | 12 |
| **3** | 4.000 | [28, 878, 12] | 27205.000 | Inception | 27.920 | 2010-07-16 | 8.300 | 22 |
| **4** | 5.000 | [12, 14, 10751] | 32657.000 | Percy Jackson & the Olympians: The Lightning T... | 26.691 | 2010-02-11 | 6.100 | 4 |
| **5** | 7.000 | [16, 10751, 35] | 10193.000 | Toy Story 3 | 24.445 | 2010-06-17 | 7.700 | 8 |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **21293** | 26425.000 | [10402] | 509306.000 | The Box | 0.600 | 2018-03-04 | 6.000 | |
| **21294** | 26425.000 | [10402] | 509306.000 | The Box | 0.600 | 2018-03-04 | 6.000 | |
| **21296** | 26425.000 | [10402] | 509306.000 | The Box | 0.600 | 2018-03-04 | 6.000 | |
| **21321** | 26092.000 | [35, 16] | 546674.000 | Enough | 0.719 | 2018-03-22 | 8.700 | |
| **21322** | 26092.000 | [35, 16] | 546674.000 | Enough | 0.719 | 2018-03-22 | 8.700 | |

3863 rows × 21 columns

Creating a new dataframe that has the studios/Gross Income/ and genres in one

In [275]:
```
BVmovies= newmoviedata[newmoviedata['studio'] == 'BV']
WBmovies= newmoviedata[newmoviedata['studio'] == 'WB']
Parmovies= newmoviedata[newmoviedata['studio'] == 'Par.']
PDWmovies= newmoviedata[newmoviedata['studio'] == 'P/DW']
Unimovies= newmoviedata[newmoviedata['studio'] == 'Uni.']
```

# Second recomendation: Examining the most profitable genres based upon the top earning studios

In [276]:
```
TopStudioMovies= pd.concat([WBmovies.head(10), BVmovies.head(10),Parmovies.
```

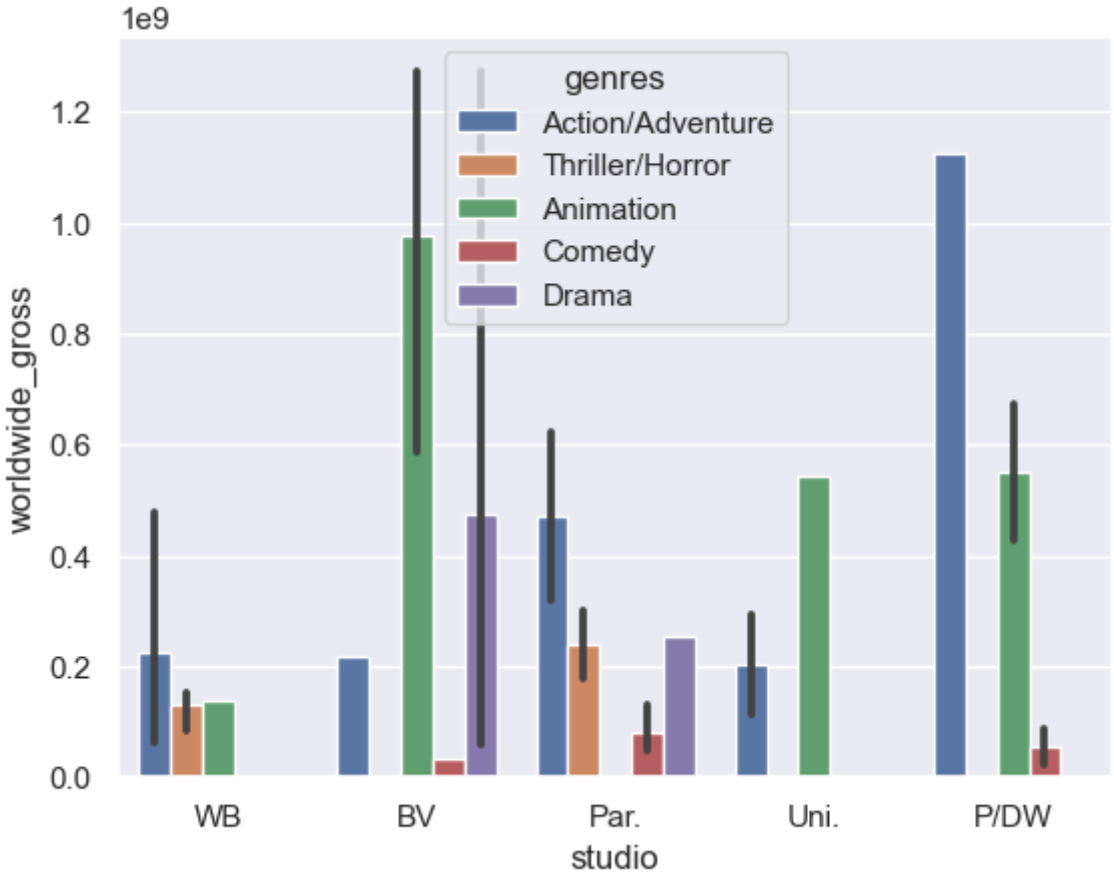
In [365]:
```
TopStudioMovies
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **2157** | 2489.000 | [16, 10751] | 49444.000 | Kung Fu Panda 2 | 17.363 | 2011-05-25 | 6.700 | 362 |
| **2383** | 2668.000 | [28, 12, 16, 10751, 14, 35] | 417859.000 | Puss in Boots | 9.078 | 2011-10-28 | 6.300 | 142 |
| **2703** | 2966.000 | [28, 878, 12] | 38356.000 | Transformers: Dark of the Moon | 5.339 | 2011-06-28 | 6.100 | 496 |
| **4546** | 5196.000 | [16, 10751, 35] | 80321.000 | Madagascar 3: Europe's Most Wanted | 18.705 | 2012-06-08 | 6.500 | 338 |
| **4571** | 5225.000 | [16, 10751, 14] | 81188.000 | Rise of the Guardians | 14.934 | 2012-11-21 | 7.300 | 369 |
| **4783** | 5399.000 | [18, 35] | 72358.000 | A Thousand Words | 8.017 | 2012-03-08 | 6.100 | 68 |

45 rows × 21 columns

In [366]:
```
sns.set_theme(style="darkgrid")
sns.barplot(x = TopStudioMovies['studio'], hue = TopStudioMovies['genres'],
```

Out[366]: <Axes: xlabel='studio', ylabel='worldwide_gross'>




Retitling the data into common genere themes

In [281]:
```
TopStudioMovies['genres'] = TopStudioMovies['genres'].replace(['Action,Adve
TopStudioMovies['genres'] = TopStudioMovies['genres'].replace(['Crime,Drama
TopStudioMovies['genres'] = TopStudioMovies['genres'].replace(['Action,Adve
TopStudioMovies['genres'] = TopStudioMovies['genres'].replace(['Action Dram
TopStudioMovies['genres'] = TopStudioMovies['genres'].replace(['Comedy,Fami
```

In [319]:
```
TopStudioMovies.dropna(subset='worldwide_gross')
```

Out[319]:

| | Unnamed: 0 | genre_ids | id_x | original_title | popularity | release_date_x | vote_average | vote_ |
|---|---|---|---|---|---|---|---|---|
| **3** | 4.000 | [28, 878, 12] | 27205.000 | Inception | 27.920 | 2010-07-16 | 8.300 | 2218 |
| **24** | 16.000 | [28, 53, 878] | 20504.000 | The Book of Eli | 18.985 | 2010-01-11 | 6.700 | 349 |
| **80** | 52.000 | [80, 18, 53] | 23168.000 | The Town | 13.476 | 2010-09-17 | 7.100 | 238 |
| **81** | 52.000 | [80, 18, 53] | 23168.000 | The Town | 13.476 | 2010-09-17 | 7.100 | 238 |
| **82** | 53.000 | [28, 37, 18, 14, 53] | 20533.000 | Jonah Hex | 13.467 | 2010-06-18 | 4.600 | 70 |
| **94** | 72.000 | [35, 18] | 41733.000 | Due Date | 12.445 | 2010-11-04 | 6.300 | 297 |
| **114** | 97.000 | [80, 18, 9648, 53] | 12201.000 | Edge of Darkness | 11.252 | 2010-01-29 | 6.300 | 79 |

# Action Movie Data

Creation of a new data frame containing only action movies from the top three studios. From there the information is going to be reassigned by whether the movie grossed over the mean score. If the movie earns more than the mean it will be determined to be successful or not successful.

In [399]:
```
disney = newmoviegross2[newmoviegross2['studio_x'].isin(['BV','WB', 'Par.']
```

In [400]:
```
disney.dropna(subset = ['genres'], inplace = True)
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/90862696
1.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  disney.dropna(subset = ['genres'], inplace = True)
```

In [401]:
```
disney['genres'] = disney['genres'].apply(lambda x: x.split(','))
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/1506458889.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  disney['genres'] = disney['genres'].apply(lambda x: x.split(','))
```

In [402]:
```
disney['genre1'] = disney['genres'].apply(lambda x: x[0])
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/726305095.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  disney['genre1'] = disney['genres'].apply(lambda x: x[0])
```

In [403]:
```
action = disney.loc[disney['genre1']=='Action']
```

In [404]:
```
action.dropna(subset = ['worldwide_gross'], inplace = True)
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/907924199.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  action.dropna(subset = ['worldwide_gross'], inplace = True)
```

In [405]:
```
action['worldwide_gross'] =action['worldwide_gross'].apply(lambda x: int(x))
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/3309173416.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  action['worldwide_gross'] =action['worldwide_gross'].apply(lambda x: int(x))
```

In [407]:
```
action['worldwide_gross'].mean()
```

Out[407]:
```
428453316.4411765
```

In [408]:
```
action['over_mean'] = ''
```

```
/var/folders/tl/4k7m4vv922qd25wl53zg2x240000gn/T/ipykernel_95575/3898943411.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  action['over_mean'] = ''
```

In [409]:
```
action.loc[action['worldwide_gross'] > action['worldwide_gross'].mean() , '
```

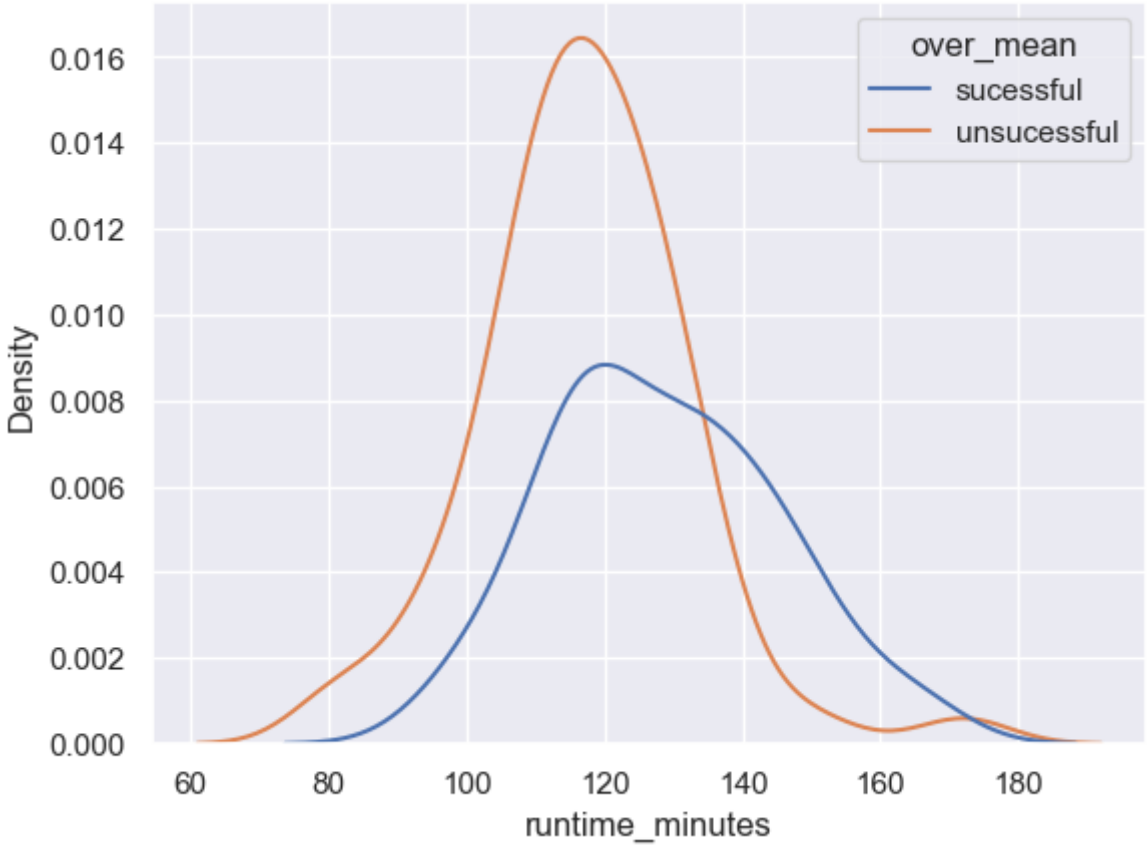
# Third Recomnedation :Successful vs. Unsuccesful Movies

Determined by looking at the distribution of succesful and unsucccesful movies compared to their runtime

In [410]:
```
action.loc[action['worldwide_gross'] <= action['worldwide_gross'].mean() ,
```

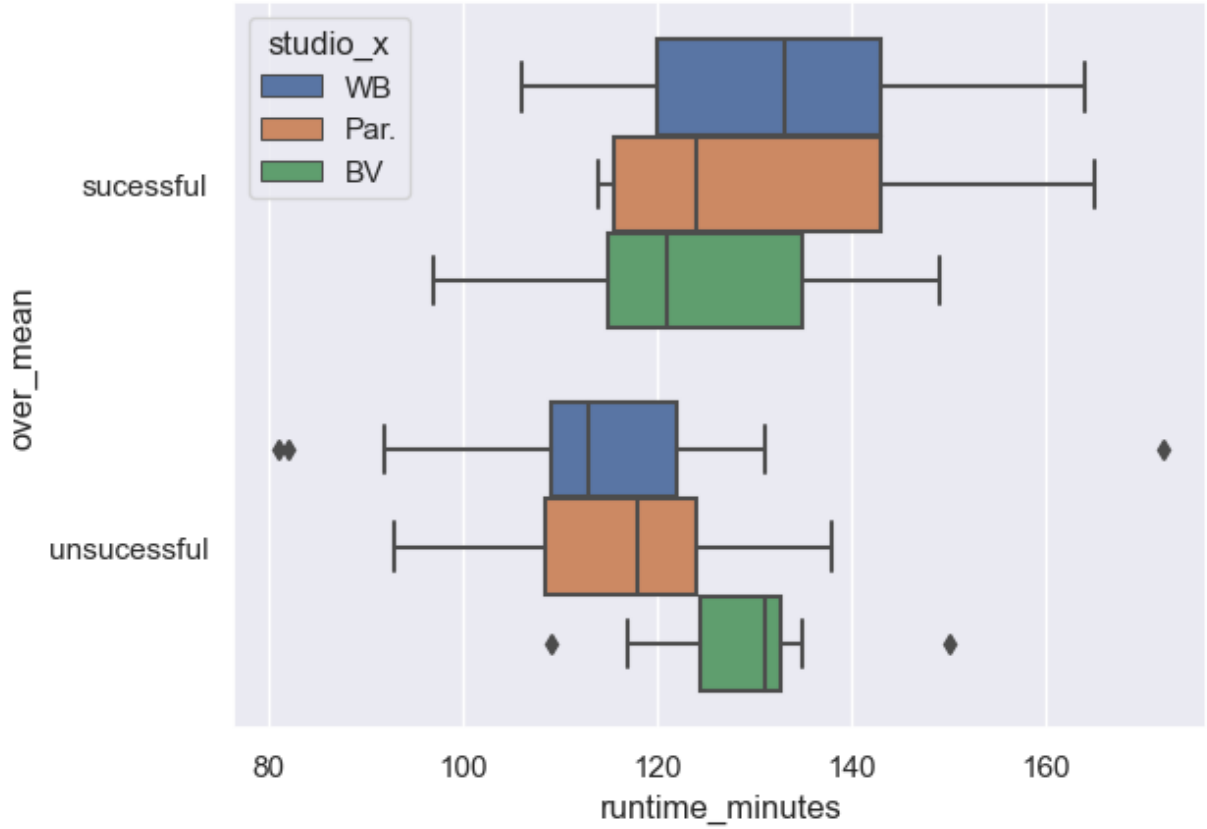
In [411]:
```
sns.kdeplot(data  = action, x = 'runtime_minutes', hue = 'over_mean')
```

Out[411]:
```
<Axes: xlabel='runtime_minutes', ylabel='Density'>
```



In [412]:
```
sns.boxplot(x=action["runtime_minutes"],y=action['over_mean'], hue=action['
```

Out[412]:
```
<Axes: xlabel='runtime_minutes', ylabel='over_mean'>
```




In [ ]:

In [ ]: