Final Project Submission

- Student name: Milena Afeworki
- Student pace: Full time
- Scheduled project review date/time: May 7, 2021 @ 1:00pm PT
- Instructor name: Abhineet Kulkarni
- Blog post URL: TBD

Business Problem

Microsoft sees all the big companies creating original video content and they want to get in on the fun. They have decided to create a new movie studio, but they don't know anything about creating movies. We are charged with exploring what types of films are currently doing the best at the box office. We must then translate those findings into actionable insights that the head of Microsoft's new movie studio can use to help decide what type of films to create.

Questions and Recommendations

- Which genres generate the highest revenue per movie?
- Who are the top 10 directors and writers knows for those highest revenue genres?
- What is the ideal financial level of investment on movie production?

In this next step I am going to import all the necessary libraies and packages that I may need.

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import os
from glob import glob
%matplotlib inline
```

Next I will explore the folder I have on file to extract the files and will also display the first 5 rows of each table to look at the data I have available to carry out my analysis and decide on which table to use for each question.

In [2]:
```
folder = "C:/Users/milen/Flatiron-April05/Final_Project1/movie-exploration/zi
os.listdir(folder)
```

Out[2]:
```
['bom.movie_gross.csv.gz',
 'imdb.name.basics.csv.gz',
 'imdb.title.akas.csv.gz',
 'imdb.title.basics.csv.gz',
 'imdb.title.crew.csv.gz',
 'imdb.title.principals.csv.gz',
 'imdb.title.ratings.csv.gz',
 'rt.movie_info.tsv.gz',
 'rt.reviews.tsv.gz',
 'tmdb.movies.csv.gz',
 'tn.movie_budgets.csv.gz']
```

In [3]:
```
csv_files = glob(f"{folder}*.csv*")
```

In [4]:
```
# Here is a code to load in files and display preview

tables = {}


for file in csv_files:
    ## Save a variable-friendly version of the file name
    table_name = file.replace('.csv.gz','').split('/')[-1].replace('.','_')
    print('====================='*5)

    ## Load and preview dataframe
    print(f"Preview of {table_name}")
    tables[table_name] = pd.read_csv(file)
    display(tables[table_name].head(5))
    print()
```

```
=========================================================================
================================
Preview of zippedData\bom_movie_gross
```

	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

```
=========================================================================
================================
Preview of zippedData\imdb_name_basics
```

Description of each table and column content

Description of each table and column content

- **imdb_title_crew** : title id linked to crew id
- **tmdb_movies** : titles and stats from IMDB
- **imdb_title_akas** : link between id and movie title
- **imdb_title_ratings** : link between title and IMDB ratings
- **imdb_name_basics** : name of cast and ids
- **imdb_title_basics** : movie title, title id, start year and runtime
- **tn_movie_budgets** : movie title, release date, and earnings/costs
- **bom_movie_gross** : movie title, studio, and earnings
- **imdb_title_principals** : link between movie title, cast id and their category

In [5]:
```
csv_files_dict = {}

for filename in csv_files:
    # cleaning the filenames
    filename_cleaned = os.path.basename(filename).replace(".csv", "").replace
    filename_df      = pd.read_csv(filename, index_col=0)
    csv_files_dict[filename_cleaned] = filename_df
```

Question 1. Which genres generate the highest revenue per movie?

Looking at the revenue returns for different genre can give us an idea of what people are more likely to be interested on. For this analysis I will explore the 'tn_movie_budgets_gz' file to get an insight on:

- do some genres earn more revenue than others?
- how is the production budget related to the revenue return of those genres?

Cleaning the data.

In [6]:
```
#csv to the dataframe.
movie_budgets_df = csv_files_dict['tn_movie_budgets_gz']
```

In [7]:
```
type(movie_budgets_df)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
movie_budgets_df.shape
```

Out[8]:
```
(5782, 5)
```

In [9]:
```
movie_budgets_df.head()
```

Out[9]:

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

In [10]:
```
movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   object
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
dtypes: object(5)
memory usage: 271.0+ KB
```

In [11]:
```
# Checking if our data frame has any null values
movie_budgets_df.isna().sum()
```

Out[11]:
```
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

In [12]:
```
# make a note of the percentage of the earnings/cost of each movie.
for col in movie_budgets_df:
    print(col)
    print(movie_budgets_df[col].value_counts(normalize = True)[:5])
    print("==============================================")
```

```
release_date
Dec 31, 2014    0.004151
Dec 31, 2015    0.003978
Dec 31, 2010    0.002594
Dec 31, 2008    0.002421
Dec 31, 2012    0.002248
Name: release_date, dtype: float64
==============================================
movie
King Kong    0.000519
Halloween    0.000519
Home         0.000519
Venom        0.000346
Unknown      0.000346
Name: movie, dtype: float64
==============================================
production_budget
$20,000,000    0.039952
$10,000,000    0.036666
$30,000,000    0.030612
$15,000,000    0.029920
$25,000,000    0.029575
Name: production_budget, dtype: float64
==============================================
domestic_gross
$0             0.094777
$8,000,000     0.001557
$7,000,000     0.001211
$2,000,000     0.001211
$10,000,000    0.001038
Name: domestic_gross, dtype: float64
==============================================
worldwide_gross
$0            0.063473
$8,000,000    0.001557
$2,000,000    0.001038
$7,000,000    0.001038
$4,000,000    0.000692
Name: worldwide_gross, dtype: float64
==============================================
```

In [13]:
```
# I will have to change the data types of the cost/earn columns and change to
# write a function for that

def convert_amt_to_int(df, col):
    df[col] = df[col].replace('[\$,]', '', regex=True).astype(np.int64)
#     df[col] = df[col].replace("$", "").replace(",", "").astype(np.int64)
    return df
```

In [14]:
```
#making a list of all the cols where we want to change the dtype

money_cols = ['production_budget', 'domestic_gross', 'worldwide_gross']

for col in money_cols:
    movie_budgets_df = convert_amt_to_int(movie_budgets_df, col)
```

In [15]:
```
movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5782 entries, 1 to 82
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       5782 non-null   object
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   int64 
dtypes: int64(3), object(2)
memory usage: 271.0+ KB
```

In [16]:
```
movie_budgets_df.head()
```

Out[16]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross
id					
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747

In [17]:
```
#confirming no na values
movie_budgets_df.isna().sum()
```

Out[17]:
```
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

In [18]:
```
#looking at the most occuring values to see if there are any weird values
for col in movie_budgets_df:
    print(f'Viewing values in col: {col}')
    print(f'Top 5 values:\n{movie_budgets_df[col].value_counts(normalize = Tr
    print("-------------------")
```

```
Viewing values in col: release_date
Top 5 values:
Dec 31, 2014    0.004151
Dec 31, 2015    0.003978
Dec 31, 2010    0.002594
Dec 31, 2008    0.002421
Dec 31, 2012    0.002248
Name: release_date, dtype: float64
-------------------
Viewing values in col: movie
Top 5 values:
King Kong    0.000519
Halloween    0.000519
Home         0.000519
Venom        0.000346
Unknown      0.000346
Name: movie, dtype: float64
-------------------
Viewing values in col: production_budget
Top 5 values:
20000000    0.039952
10000000    0.036666
30000000    0.030612
15000000    0.029920
25000000    0.029575
Name: production_budget, dtype: float64
-------------------
Viewing values in col: domestic_gross
Top 5 values:
0           0.094777
8000000     0.001557
2000000     0.001211
7000000     0.001211
10000000    0.001038
Name: domestic_gross, dtype: float64
-------------------
Viewing values in col: worldwide_gross
Top 5 values:
0          0.063473
8000000    0.001557
7000000    0.001038
2000000    0.001038
4000000    0.000692
Name: worldwide_gross, dtype: float64
-------------------
```

Creating new Column for Gross Profit

In [19]:
```
# Now that I know the dataframe is clean  I will get the required difference
# so I can start working on the logic needed to get the top 5 grossing movies
# As we have the production budget available, we can create a new column whic


movie_budgets_df['budget_gross_diff'] = movie_budgets_df['worldwide_gross'] -
                                        movie_budgets_df['production_budget']
movie_budgets_df.head()
```

Out[19]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	6350
3	Jun 7, 2019	Dark Phoenix	350000000	42762350	149762350	-2002
4	May 1, 2015	Avengers: Age of Ultron	330600000	459005868	1403013963	10724
5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	9997

In [20]:
```
# arranging in order to obtain top 50 most grossing films
movie_budgets_df = movie_budgets_df.sort_values(by='budget_gross_diff', ascen
```

In [21]:
```
movie_budgets_df.shape
```

Out[21]: (5782, 6)

In [22]:
```
movie_budgets_df.head()
```

Out[22]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
id						
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
43	Dec 19, 1997	Titanic	200000000	659363944	2208208395	20082
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	17481
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	17473
34	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	14338

In [23]:
```
movie_budgets_df_top50 = movie_budgets_df[:50]
```

In [24]:
```
movie_budgets_df_top50.shape
```

Out[24]: (50, 6)

In [25]:
```
movie_budgets_df_top50.head()
```

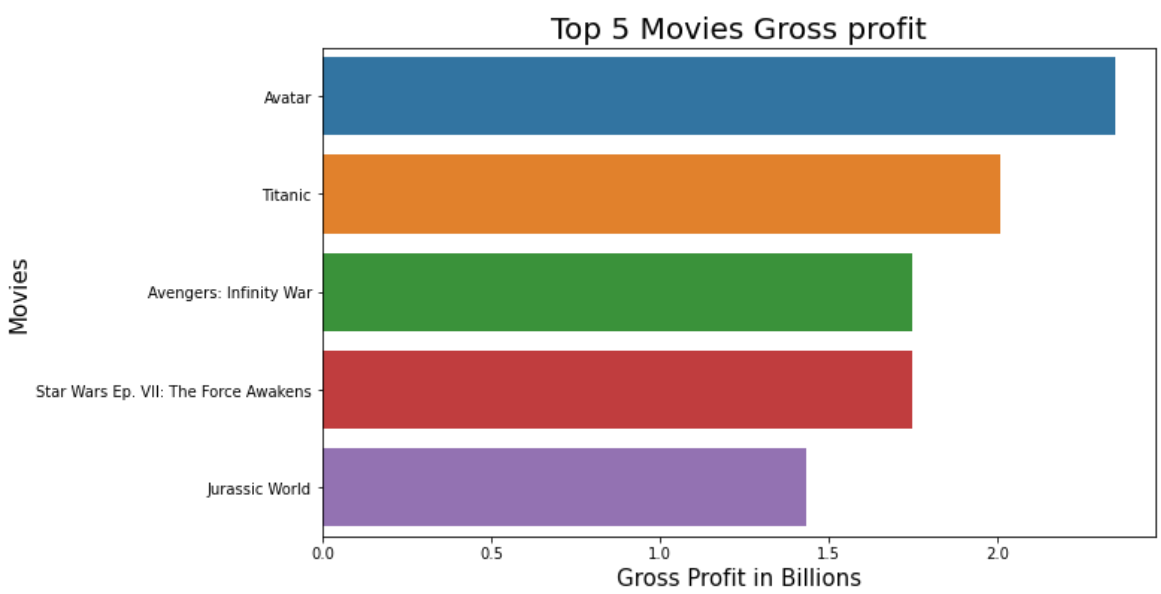
Out[25]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
43	Dec 19, 1997	Titanic	200000000	659363944	2208208395	20082
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	17481
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	17473
34	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	14338

Visualizing Gross Profit

In [26]:
```
# set up the figure size and plot movie title vs gross profit

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = "budget_gross_diff", y = "movie", data = movie_budgets_df_top
ax.set_xlabel('Gross Profit in Billions', fontsize=15)
ax.set_ylabel('Movies', fontsize=15)
ax.set_title('Top 5 Movies Gross profit', fontsize=20)
ax.xaxis.offsetText.set_visible(False)
```



Using the budget difference method I realized that with this direct dollar value I may be ignoring any movies that might have profited a little less than our top 5 movies at the box office. Hence I am interested in exploring the percent of return on the production budget.

Creating 'roi' column for Return Percentage

In [27]:
```
# For getting the percent of return I'll just need to divide the new column
# I created by the production budget and store the values in a new column
movie_budgets_df['roi'] = movie_budgets_df['budget_gross_diff'] / movie_budge
```

In [28]:
```
movie_budgets_df.head()
```

Out[28]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
id						
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
43	Dec 19, 1997	Titanic	200000000	659363944	2208208395	20082
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	17481
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	17473
34	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	14338

In [29]:
```
# sorting the values of 'roi'
movie_budgets_df = movie_budgets_df.sort_values(by='roi', ascending=False)
```

In [30]:
```
movie_budgets_df.head()
```

Out[30]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gr
id						
46	Jun 30, 1972	Deep Throat	25000	45000000	45000000	4
14	Mar 21, 1980	Mad Max	200000	8750000	99750000	9
93	Sep 25, 2009	Paranormal Activity	450000	107918810	194183034	19
80	Jul 10, 2015	The Gallows	100000	22764410	41656474	4
7	Jul 14, 1999	The Blair Witch Project	600000	140539099	248300000	24

In [31]:
```
movie_budgets_df.isna().sum()
```

Out[31]:
```
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
budget_gross_diff    0
roi                  0
dtype: int64
```

I now have a dataframe sorted by the return on the production budget that I started with.

Main problem I may run into now is that there are movies with really low budgets coming at the top. We know Microsoft will want to enter the movie industry with a bang assuming they will have a budget of at least $10 million.

I will ignore any movies with a budget smaller than that.

In [32]:
```
movie_budgets_df = movie_budgets_df[movie_budgets_df['production_budget'] > 1
```

In [33]:
```
movie_budgets_df.shape
```

Out[33]:
```
(3535, 7)
```

In [34]:
```
movie_budgets_df.head()
```

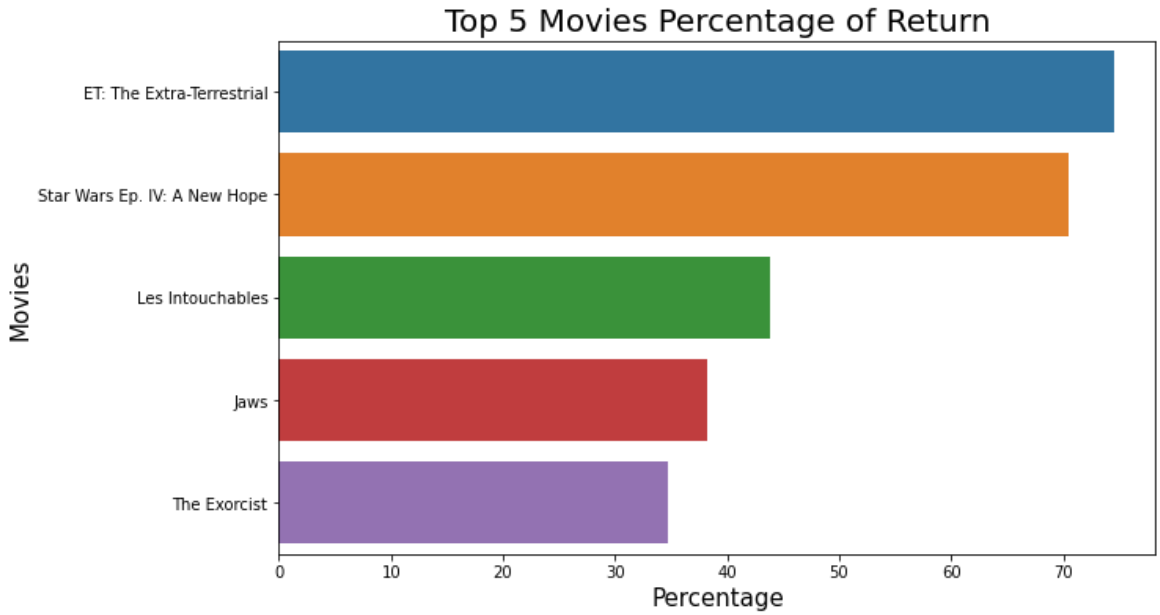
Out[34]:

id	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_g
26	Jun 11, 1982	ET: The Extra-Terrestrial	10500000	435110554	792965326	78
65	May 25, 1977	Star Wars Ep. IV: A New Hope	11000000	460998007	786598007	77
18	May 25, 2012	Les Intouchables	10800000	13182281	484873045	47
41	Jun 20, 1975	Jaws	12000000	260000000	470700000	45
42	Dec 26, 1973	The Exorcist	12000000	230347346	428214478	41

Visualizing Return Percentage

In [35]:
```
# set up the figure size and plot movie title vs Return Percentage

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = "roi", y = "movie", data = movie_budgets_df[:5])
ax.set_xlabel('Percentage', fontsize=15)
ax.set_ylabel('Movies', fontsize=15)
ax.set_title('Top 5 Movies Percentage of Return', fontsize=20)
ax.xaxis.offsetText.set_visible(False)
```



What are the most popular genres based on their returns? Now that I have the movies details I will explore another dataframe with genre details supposedly having some more information about the movies like their genre will give some advanced insights.

In [36]:
```
#title basics df has all the required info
imdb_title_basics_df = csv_files_dict['imdb_title_basics_gz']
```

In [37]:
```
imdb_title_basics_df.head()
```

Out[37]:

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

In [38]:
```
movie_budgets_df_top50.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 26
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       50 non-null     object
 1   movie              50 non-null     object
 2   production_budget  50 non-null     int64 
 3   domestic_gross     50 non-null     int64 
 4   worldwide_gross    50 non-null     int64 
 5   budget_gross_diff  50 non-null     int64 
dtypes: int64(4), object(2)
memory usage: 2.7+ KB
```

In [39]:
```
# I will need to create a new column with only the release year so I could me
# for a more precise match.
movie_budgets_df_top50['release_year'] = pd.to_datetime(
                                        movie_budgets_df_top50['release_date'
```

In [40]:
```
# Checking if the new column is added.
movie_budgets_df_top50.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 26
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       50 non-null     object
 1   movie              50 non-null     object
 2   production_budget  50 non-null     int64 
 3   domestic_gross     50 non-null     int64 
 4   worldwide_gross    50 non-null     int64 
 5   budget_gross_diff  50 non-null     int64 
 6   release_year       50 non-null     int64 
dtypes: int64(5), object(2)
memory usage: 3.1+ KB
```

In [41]:
```
movie_budgets_df_top50.head()
```

Out[41]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
id						
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
43	Dec 19, 1997	Titanic	200000000	659363944	2208208395	20082
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	17481
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	17473
34	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	14338

In [42]:
```
# Merging the two tables on the common columns content they have.
movie_details_df2 = pd.merge(movie_budgets_df_top50, imdb_title_basics_df,
                             left_on=  ['movie', 'release_year'],
                             right_on= ['primary_title', 'start_year'],
                             how = 'left')
```

In [43]:
```
movie_details_df2.shape
```

Out[43]: (51, 12)

In [44]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51 entries, 0 to 50
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   release_date       51 non-null     object 
 1   movie              51 non-null     object 
 2   production_budget  51 non-null     int64  
 3   domestic_gross     51 non-null     int64  
 4   worldwide_gross    51 non-null     int64  
 5   budget_gross_diff  51 non-null     int64  
 6   release_year       51 non-null     int64  
 7   primary_title      33 non-null     object 
 8   original_title     33 non-null     object 
 9   start_year         33 non-null     float64
 10  runtime_minutes    33 non-null     float64
 11  genres             33 non-null     object 
dtypes: float64(2), int64(5), object(5)
memory usage: 5.2+ KB
```

In [45]:
```
movie_details_df2.head()
```

Out[45]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
0	Dec 18, 2009	Avatar	425000000	760507625	2776345279	235134
1	Dec 19, 1997	Titanic	200000000	659363944	2208208395	200820
2	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
3	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	174731
4	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	143385

In [46]:
```
# according to the info we have 18 null values. Let's drop them since they
# don't have the necessary data for out analysis.
movie_details_df2.dropna(inplace=True)
```

In [47]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33 entries, 2 to 49
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   release_date       33 non-null     object 
 1   movie              33 non-null     object 
 2   production_budget  33 non-null     int64  
 3   domestic_gross     33 non-null     int64  
 4   worldwide_gross    33 non-null     int64  
 5   budget_gross_diff  33 non-null     int64  
 6   release_year       33 non-null     int64  
 7   primary_title      33 non-null     object 
 8   original_title     33 non-null     object 
 9   start_year         33 non-null     float64
 10  runtime_minutes    33 non-null     float64
 11  genres             33 non-null     object 
dtypes: float64(2), int64(5), object(5)
memory usage: 3.4+ KB
```

Although some data and information was lost, those films were being matched inaccurately and that would've led us to having incorrect conclusions anyway.

Creating a separate column for each genre

To investigate which genres have the most returns I will first create a new column for each genre and assign a value of 1 for the genre of each movie. Note that some movies are known for multiple genres.

In [48]:
```
# removing the comma sign in the list of genres bieng treated
# as a string data type and change to list data type.
movie_details_df2['genres'] = movie_details_df2['genres'].apply(lambda x: x.s
```

In [49]:
```
movie_details_df2.head()
```

Out[49]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
2	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
4	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	143385
5	Apr 3, 2015	Furious 7	190000000	353007020	1518722794	132872
6	May 4, 2012	The Avengers	225000000	623279547	1517935897	129293
8	Feb 16, 2018	Black Panther	200000000	700059566	1348258224	114825

In [50]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33 entries, 2 to 49
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   release_date       33 non-null     object 
 1   movie              33 non-null     object 
 2   production_budget  33 non-null     int64  
 3   domestic_gross     33 non-null     int64  
 4   worldwide_gross    33 non-null     int64  
 5   budget_gross_diff  33 non-null     int64  
 6   release_year       33 non-null     int64  
 7   primary_title      33 non-null     object 
 8   original_title     33 non-null     object 
 9   start_year         33 non-null     float64
 10  runtime_minutes    33 non-null     float64
 11  genres             33 non-null     object 
dtypes: float64(2), int64(5), object(5)
memory usage: 3.4+ KB
```

In [51]:
```
#making a set of all genres available as set doesn't allow duplicate values.

all_genres = set()
for genres in movie_details_df2['genres']:
    if genres:
        all_genres.update(genres)
```

In [52]:
```
all_genres
```

Out[52]:
```
{'Action',
 'Adventure',
 'Animation',
 'Biography',
 'Comedy',
 'Crime',
 'Drama',
 'Family',
 'Fantasy',
 'Music',
 'Musical',
 'Sci-Fi',
 'Thriller'}
```

In [53]:
```
#adding cols with zeros for all the genres. Will modify genre to 1 if the fil

for genre in all_genres:
    movie_details_df2[genre] = np.zeros(shape=movie_details_df2.shape[0])

movie_details_df2.head()
movie_details_df2.columns
```

Out[53]:
```
Index(['release_date', 'movie', 'production_budget', 'domestic_gross',
       'worldwide_gross', 'budget_gross_diff', 'release_year', 'primary_tit
le',
       'original_title', 'start_year', 'runtime_minutes', 'genres', 'Sci-F
i',
       'Adventure', 'Crime', 'Musical', 'Biography', 'Thriller', 'Fantasy',
       'Drama', 'Music', 'Action', 'Animation', 'Comedy', 'Family'],
      dtype='object')
```

In [54]:
```
#setting the genre to be 1 if the film is of that genre

for index, row in movie_details_df2.iterrows():
    if row['genres']:
        for genre in row['genres']:
            movie_details_df2.loc[index, genre] = 1

movie_details_df2.head()
```

Out[54]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
2	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
4	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	143385
5	Apr 3, 2015	Furious 7	190000000	353007020	1518722794	132872
6	May 4, 2012	The Avengers	225000000	623279547	1517935897	129293
8	Feb 16, 2018	Black Panther	200000000	700059566	1348258224	114825

5 rows × 25 columns

In [55]:
```
movie_details_df_modified = movie_details_df2.drop(columns = 'genres')
```

In [56]:
```
#checking the counts for all different genres

for col in movie_details_df_modified:
    print(f'Viewing values in col: {col}')
    print(f'Top 5 values:\n{movie_details_df_modified[col].value_counts()}')
```

```
May 1, 2015     1
Jun 17, 2016    1
Apr 14, 2017    1
Jul 13, 2012    1
Jun 22, 2018    1
Feb 16, 2018    1
Nov 2, 2018     1
Dec 21, 2018    1
Jun 29, 2011    1
Jul 20, 2012    1
Jul 3, 2013     1
Apr 3, 2015     1
Jun 30, 2017    1
May 4, 2012     1
Jul 8, 2016     1
Mar 4, 2016     1
Mar 8, 2019     1
Name: release_date, dtype: int64
Viewing values in col: movie
Top 5 values:
```

In [57]:
```
#making a list of all genres

cols = list(movie_details_df_modified.columns)
cols
```

Out[57]:
```
['release_date',
 'movie',
 'production_budget',
 'domestic_gross',
 'worldwide_gross',
 'budget_gross_diff',
 'release_year',
 'primary_title',
 'original_title',
 'start_year',
 'runtime_minutes',
 'Sci-Fi',
 'Adventure',
 'Crime',
 'Musical',
 'Biography',
 'Thriller',
 'Fantasy',
 'Drama',
 'Music',
 'Action',
 'Animation',
 'Comedy',
 'Family']
```

In [58]:
```
genre_cols = cols[11:]
genre_cols
```

Out[58]:
```
['Sci-Fi',
 'Adventure',
 'Crime',
 'Musical',
 'Biography',
 'Thriller',
 'Fantasy',
 'Drama',
 'Music',
 'Action',
 'Animation',
 'Comedy',
 'Family']
```

In [59]:
```
#getting a dict with genre counts

genre_count = {}
for col in genre_cols:
    count = np.sum(movie_details_df2[col] == 1)
    genre_count[col] = count
```

In [60]:
```
genre_count
```

Out[60]:
```
{'Sci-Fi': 12,
 'Adventure': 27,
 'Crime': 2,
 'Musical': 2,
 'Biography': 1,
 'Thriller': 4,
 'Fantasy': 4,
 'Drama': 2,
 'Music': 1,
 'Action': 19,
 'Animation': 10,
 'Comedy': 10,
 'Family': 3}
```

Visualizing Genre Frequency.

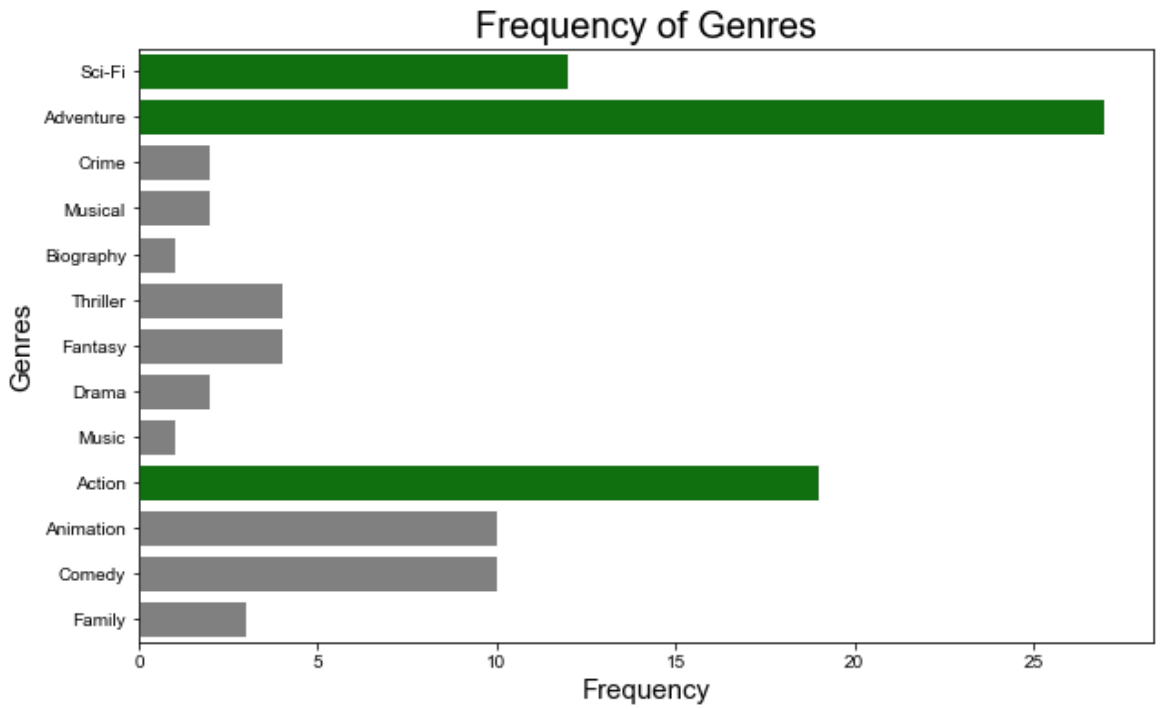
In [61]:
```
# assign the list of keys and values to visualize what our top 3 recurring ge
keys   = list(genre_count.keys())
values = list(genre_count.values())


sorted_top3_values = sorted(values, key = lambda x: x, reverse = True)[:3]
print(sorted_top3_values)
```

```
[27, 19, 12]
```

In [62]:
```
fig, ax = plt.subplots(figsize=(10,6))

clrs = ['green' if (x in sorted_top3_values) else 'grey' for x in values ]
sns.set_theme(style='whitegrid')
sns.barplot(y = keys, x = values, palette=clrs)
ax.set_xlabel('Frequency', fontsize=15)
ax.set_ylabel('Genres', fontsize=15)
ax.set_title('Frequency of Genres', fontsize=20);
```



Notice from the plot that the top 3 recurring genres are 'Action', 'Adventure', and 'SciFi'. I will explore this more to learn if this applies true for the revenue returns as well.

In [63]:
```
# create a dict for the genres and mean of their worldwide gross.
worldwide_gross = {}
for genre in all_genres:
    grouped                = movie_details_df_modified.groupby(by = "".join(g
    worldwide_gross[genre] = grouped.iloc[1]["worldwide_gross"]

    print(worldwide_gross[genre])
```

```
1335618804.75
1177740759.5925925
1376784530.5
1142345408.0
894985342.0
1237133785.25
1114269141.5
928919944.5
894985342.0
1280519596.9473684
1056131138.1
1028328686.3
1082515121.0
```

In [64]:
```
worldwide_gross = dict(sorted(worldwide_gross.items(), key = lambda item : it
worldwide_gross
```

Out[64]:
```
{'Crime': 1376784530.5,
 'Sci-Fi': 1335618804.75,
 'Action': 1280519596.9473684,
 'Thriller': 1237133785.25,
 'Adventure': 1177740759.5925925,
 'Musical': 1142345408.0,
 'Fantasy': 1114269141.5,
 'Family': 1082515121.0,
 'Animation': 1056131138.1,
 'Comedy': 1028328686.3,
 'Drama': 928919944.5,
 'Biography': 894985342.0,
 'Music': 894985342.0}
```

Visualizing Genre vs Worldwide Gross.

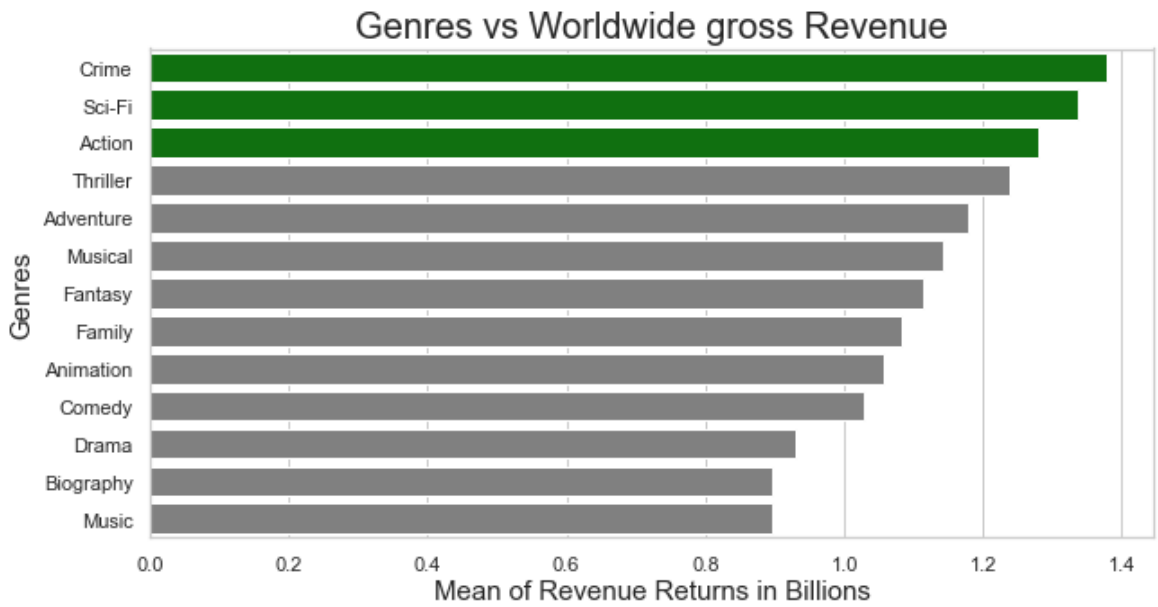
In [65]:

```
# Visualize the figures.
keys   = list(worldwide_gross.keys())
values = list(worldwide_gross.values())

sorted_top3_values = sorted(values, key = lambda x: x, reverse = True)[:3]
print(sorted_top3_values)

sns.set_theme(style='whitegrid')
fig, ax       = plt.subplots(figsize=(10,5))
clrs          = ['green' if (x in sorted_top3_values) else 'grey' for x in va
sns.barplot(y = keys, x = values, palette=clrs)
ax.set_xlabel('Mean of Revenue Returns in Billions', fontsize=15)
ax.set_ylabel('Genres', fontsize=15)
ax.set_title('Genres vs Worldwide gross Revenue', fontsize=20)
ax.xaxis.offsetText.set_visible(False)
```

```
[1376784530.5, 1335618804.75, 1280519596.9473684]
```



In [66]:
```
# movie_details_df2
budget_gross_diff = {}
for genre in all_genres:
    grouped = movie_details_df2.groupby(by = "".join(genre)).median()
    budget_gross_diff[genre] = grouped.iloc[1]["budget_gross_diff"]


budget_gross_diff
```

Out[66]:
```
{'Sci-Fi': 1043903117.5,
 'Adventure': 928790543.0,
 'Crime': 1156784530.5,
 'Musical': 962345408.0,
 'Biography': 839985342.0,
 'Thriller': 947686624.0,
 'Fantasy': 906192875.0,
 'Drama': 813919944.5,
 'Music': 839985342.0,
 'Action': 986894640.0,
 'Animation': 884323225.5,
 'Comedy': 871962904.5,
 'Family': 825491110.0}
```

In [67]:
```
budget_gross_diff = dict(sorted(budget_gross_diff.items(), key = lambda item
budget_gross_diff
```

Out[67]:
```
{'Crime': 1156784530.5,
 'Sci-Fi': 1043903117.5,
 'Action': 986894640.0,
 'Musical': 962345408.0,
 'Thriller': 947686624.0,
 'Adventure': 928790543.0,
 'Fantasy': 906192875.0,
 'Animation': 884323225.5,
 'Comedy': 871962904.5,
 'Biography': 839985342.0,
 'Music': 839985342.0,
 'Family': 825491110.0,
 'Drama': 813919944.5}
```

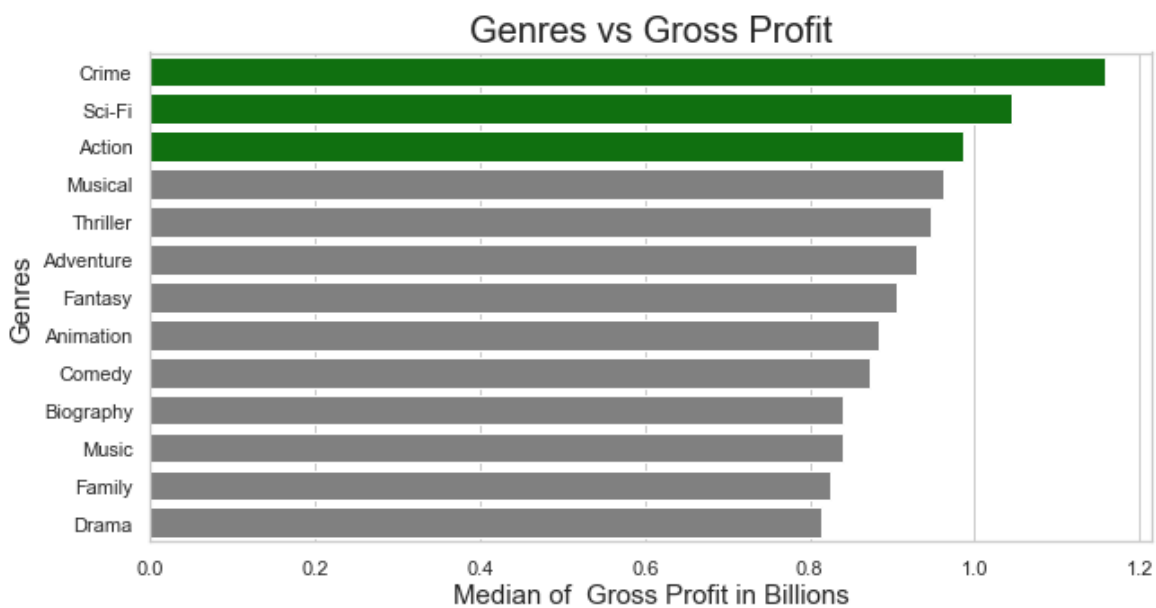
Visualizing Genre vs Gross Profit.

In [68]:
```
key   = list(budget_gross_diff.keys())
value = list(budget_gross_diff.values())

sorted_top3_values = sorted(value, key = lambda x: x, reverse = True)[:3]
print(sorted_top3_values)

sns.set_theme(style='whitegrid')
clrs    = ['green' if (x in sorted_top3_values) else 'grey' for x in value]
fig, ax = plt.subplots(figsize=(10,5))
sns.barplot(y = key, x = value, palette=clrs)
ax.set_xlabel('Median of  Gross Profit in Billions', fontsize=15)
ax.set_ylabel('Genres', fontsize=15)
ax.set_title('Genres vs Gross Profit', fontsize=20)
ax.xaxis.offsetText.set_visible(False)
```

```
[1156784530.5, 1043903117.5, 986894640.0]
```



Great! Once again 'Crime', 'SciFi', and 'Action' make it to the top3 revenue generating genres. From this I am able to conclude that the mean and median resulting in similar outcomes signifies the uniform distribution of the data. Hence, coming to the conclusion of the first question ***'Which genres generate the highest revenue per movie?'***. According to the findings so far I would recommend Microsoft to focus more on 'Crime', 'SciFi' and 'Action' movies for competetive revenue returns.

Question 2 Who are the most frequent directors in the top 3 grossing genres?

Looking into the table with list of titles the professionals are known for and try to extract the directors and writers involved in those movies. Next I will try and merge the title ids of the directors and writers to see who are the too grossing directors first I will try to find the tconst of the genres with top grossing revenue

In [69]:
```
imdb_title_crew_df = csv_files_dict['imdb_title_crew_gz']
imdb_title_crew_df.head()
```

Out[69]:

tconst	directors	writers
tt0285252	nm0899854	nm0899854
tt0438973	NaN	nm0175726,nm1802864
tt0462036	nm1940585	nm1940585
tt0835418	nm0151540	nm0310087,nm0841532
tt0878654	nm0089502,nm2291498,nm2292011	nm0284943

In [70]:
```
imdb_title_crew_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146144 entries, tt0285252 to tt9010172
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   directors  140417 non-null  object
 1   writers    110261 non-null  object
dtypes: object(2)
memory usage: 3.3+ MB
```

Data Cleaning.

Now in the title_crew data frame there are a couple of things to fix. The first would be to drop the null values since they are small in number and also carrying on with those null values would not help in finding the answers I am looking for. Second point would be to adjust and reset the index column to a numeric value.

In [71]:
```
imdb_title_crew_df.dropna(inplace=True)
```

In [72]:
```
imdb_title_crew_df = imdb_title_crew_df.reset_index()
```

In [73]:
```
imdb_title_crew_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109008 entries, 0 to 109007
Data columns (total 3 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   tconst     109008 non-null  object
 1   directors  109008 non-null  object
 2   writers    109008 non-null  object
dtypes: object(3)
memory usage: 2.5+ MB
```

In [74]:
```
imdb_title_crew_df.head()
```

Out[74]:

	tconst	directors	writers
0	tt0285252	nm0899854	nm0899854
1	tt0462036	nm1940585	nm1940585
2	tt0835418	nm0151540	nm0310087,nm0841532
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943
4	tt0996958	nm2286991	nm2286991,nm2651190

Ok, now it looks good and data has been cleaned for further analysis. The next step would be to load in the 'imdb_title_basics_gz' to take a look at the connection of title ids to their genres.

In [75]:
```
# assign a variable to the dataframe.
imdb_title_basics_df = csv_files_dict['imdb_title_basics_gz']
imdb_title_basics_df.head()
```

Out[75]:

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

In [76]:
```
imdb_title_basics_df.shape
```

Out[76]: (146144, 5)

In [77]:
```
imdb_title_basics_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146144 entries, tt0063540 to tt9916754
Data columns (total 5 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   primary_title    146144 non-null  object 
 1   original_title   146123 non-null  object 
 2   start_year       146144 non-null  int64  
 3   runtime_minutes  114405 non-null  float64
 4   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(3)
memory usage: 6.7+ MB
```

In [78]:
```
imdb_title_basics_df.isna().any()
```

Out[78]:
```
primary_title      False
original_title      True
start_year         False
runtime_minutes     True
genres              True
dtype: bool
```

Clearly there are some null values in this data frame. I'll start by droping the null values, which are very few as compared to the data, as well as removing the 'runtime_minutes' column since I don't need it for my analysis.

In [79]:
```
imdb_title_basics_df.dropna(inplace=True)
```

In [80]:
```
imdb_title_basics_df = imdb_title_basics_df.drop(['runtime_minutes'], axis=1)
```

In [81]:
```
# also reset the index column to be numeric instead of tconst.
imdb_title_basics_df = imdb_title_basics_df.reset_index()
```

In [82]:
```
imdb_title_basics_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112232 entries, 0 to 112231
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   tconst          112232 non-null  object 
 1   primary_title   112232 non-null  object 
 2   original_title  112232 non-null  object 
 3   start_year      112232 non-null  int64  
 4   genres          112232 non-null  object 
dtypes: int64(1), object(4)
memory usage: 4.3+ MB
```

Merging tables to obtain title ids of genres.

Now that the data frame is cleaned I will start by merging the 'imdb_title_crew_df' data frame and the 'imdb_title_basics_df' based on the common tconst column. I am using a left join on 'imdb_title_crew_df' because I need all the directors and writers to be my base point for matching with the rest of the columns on the 'imdb_title_basics_df'.

From there I will start cleaing the data and find the directors and writers known for the top grossing genres.

In [83]:

```
# merging the title crew (directors and writers) dataframe with the title bas
director_writer_df = pd.merge(imdb_title_crew_df, imdb_title_basics_df,
                              left_on=  ['tconst'], right_on = ['tconst'],
                              how = 'left')
director_writer_df.head()
```

Out[83]:

	tconst	directors	writers	primary_title	original_titl
0	tt0285252	nm0899854	nm0899854	Life's a Beach	Life's Beac
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Stev Phoenix: Th Untold Stor
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Th Babymaker
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	Bulletfac
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend the Re Reape

In [84]:

```
director_writer_df.shape
```

Out[84]: (109008, 7)

In [85]:
```
# removing the 'start year' column since it is not necessary for our analysis
director_writer_df = director_writer_df.drop(['start_year'], axis=1)
director_writer_df.head()
```

Out[85]:

	tconst	directors	writers	primary_title	original_titl
0	tt0285252	nm0899854	nm0899854	Life's a Beach	Life's Beac
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Stev Phoenix: Th Untold Stor
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Th Babymaker
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	Bulletfac
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend the Re Reape

In [86]:
```
director_writer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109008 entries, 0 to 109007
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   tconst          109008 non-null  object
 1   directors       109008 non-null  object
 2   writers         109008 non-null  object
 3   primary_title   87417 non-null   object
 4   original_title  87417 non-null   object
 5   genres          87417 non-null   object
dtypes: object(6)
memory usage: 5.8+ MB
```

In [87]:
```
# Removing the null values from our data since they don't correlate with the
director_writer_df.dropna(inplace=True)
```

In [88]:
```
director_writer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 87417 entries, 0 to 109006
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   tconst          87417 non-null  object
 1   directors       87417 non-null  object
 2   writers         87417 non-null  object
 3   primary_title   87417 non-null  object
 4   original_title  87417 non-null  object
 5   genres          87417 non-null  object
dtypes: object(6)
memory usage: 4.7+ MB
```

In [89]:
```
director_writer_df.head()
```

Out[89]:

	tconst	directors	writers	primary_title	original_titl
0	tt0285252	nm0899854	nm0899854	Life's a Beach	Life's Beac
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Stev Phoenix: Th Untold Stor
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Th Babymaker
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	Bulletfac
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend the Re Reape

Create column for only top3 genres.

Now that I have the data cleaned, noticing that the 'genres' column has list values as well I will be creating a new series 'top3_gross_genres' to extract only the genres metioned in top3_grossing_genres. For this code I will use numpy's np.zeros to return a new column with zero values which will be later replaced.

In [90]:
```
director_writer_df['top3_gross_genres'] = np.zeros(shape=director_writer_df.s
director_writer_df
```

Out[90]:

	tconst	directors	writers	primary_title	ori
0	tt0285252	nm0899854	nm0899854	Life's a Beach	
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Ph Ur
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Ba
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	
...	...	...	...	...	
109002	tt8992954	nm4927970	nm4927970	Rossotrevi - The red fountain	R
109003	tt8998302	nm10121510	nm10121510	Vietnam@55	Vie
109004	tt8999892	nm10122247	nm10122247,nm10122246	Dumpster Fire: A Time Of Current Times	Fi
109005	tt8999974	nm10122357	nm10122357	Madre Luna	M
109006	tt9001390	nm6711477	nm6711477	The woman and the river	T an

87417 rows × 7 columns

In [91]:

```
# Return list of genres from the string in the 'genres' column
director_writer_df['genres'] = director_writer_df['genres'].apply(lambda x: x
director_writer_df
```

Out[91]:

	tconst	directors	writers	primary_title	ori
0	tt0285252	nm0899854	nm0899854	Life's a Beach	
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Ph Ur
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Ba
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	
...	...	...	...	...	
109002	tt8992954	nm4927970	nm4927970	Rossotrevi - The red fountain	R
109003	tt8998302	nm10121510	nm10121510	Vietnam@55	Vie
109004	tt8999892	nm10122247	nm10122247,nm10122246	Dumpster Fire: A Time Of Current Times	Fi
109005	tt8999974	nm10122357	nm10122357	Madre Luna	M
109006	tt9001390	nm6711477	nm6711477	The woman and the river	T an

87417 rows × 7 columns

In [92]:

```
# These are the top3 grossing genres
# Unpack the lists in the 'genres' column and store only the members of
# the top3 genres in the new column.

top3_genres    = ['Crime', 'Action', 'Sci-Fi']

top_list       = []
total_top_list = []

for index, row in director_writer_df.iterrows():
    if row['genres']:
        for genre_list in row['genres']:
            if genre_list in top3_genres:
                top_list.append(genre_list)
            else:
                continue
        total_top_list.append(top_list)
        top_list = []

director_writer_df = director_writer_df.assign(top3_gross_genres = total_top_
```

In [93]:

```
director_writer_df.head()
```

Out[93]:

	tconst	directors	writers	primary_title	original_titl
0	tt0285252	nm0899854	nm0899854	Life's a Beach	Life's Beac
1	tt0462036	nm1940585	nm1940585	Steve Phoenix: The Untold Story	Stev Phoenix: Th Untold Stor
2	tt0835418	nm0151540	nm0310087,nm0841532	The Babymakers	Th Babymaker
3	tt0878654	nm0089502,nm2291498,nm2292011	nm0284943	Bulletface	Bulletfac
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend the Re Reape

In [94]:
```
director_writer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 87417 entries, 0 to 109006
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   tconst             87417 non-null  object
 1   directors          87417 non-null  object
 2   writers            87417 non-null  object
 3   primary_title      87417 non-null  object
 4   original_title     87417 non-null  object
 5   genres             87417 non-null  object
 6   top3_gross_genres  87417 non-null  object
dtypes: object(7)
memory usage: 5.3+ MB
```

Perfect! Since the empty values were returned as an emplty list I will filter them out using the next code.

In [95]:
```
# Removes rows with empty top3_gross_genres.
director_writer_df = director_writer_df[director_writer_df['top3_gross_genres
```

In [96]:
```
director_writer_df.head()
```

Out[96]:

	tconst	directors	writers	pr
4	tt0996958	nm2286991	nm2286991,nm2651190	
5	tt0999913	nm0527109	nm0527109,nm0329051,nm0001603,nm0930684	S
9	tt10011102	nm4853354	nm2215938,nm0219964	
36	tt1138442	nm1012219	nm0000247,nm4221403,nm4220872,nm4220087,nm1109...	
40	tt1323592	nm2598931,nm2442244	nm2598931,nm0132843,nm3032994,nm3197208	

In [97]:
```
director_writer_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13157 entries, 4 to 108982
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   tconst             13157 non-null  object
 1   directors          13157 non-null  object
 2   writers            13157 non-null  object
 3   primary_title      13157 non-null  object
 4   original_title     13157 non-null  object
 5   genres             13157 non-null  object
 6   top3_gross_genres  13157 non-null  object
dtypes: object(7)
memory usage: 822.3+ KB
```

Exploring more dataframes for name ids

Alright, I will keep this dataframe on hold while I explore another table with the name-ids of the directors and writers involved in those genres.

In [98]:
```
# Convert csv file to dataframe
imdb_title_principals = csv_files_dict["imdb_title_principals_gz"]
imdb_title_principals.head()
```

Out[98]:

tconst	ordering	nconst	category	job	characters
tt0111414	1	nm0246005	actor	NaN	["The Man"]
tt0111414	2	nm0398271	director	NaN	NaN
tt0111414	3	nm3739909	producer	producer	NaN
tt0323808	10	nm0059247	editor	NaN	NaN
tt0323808	1	nm3579312	actress	NaN	["Beth Boothby"]

In [99]:
```
imdb_title_principals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1028186 entries, tt0111414 to tt9692684
Data columns (total 5 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   ordering    1028186 non-null  int64 
 1   nconst      1028186 non-null  object
 2   category    1028186 non-null  object
 3   job         177684 non-null   object
 4   characters  393360 non-null   object
dtypes: int64(1), object(4)
memory usage: 47.1+ MB
```

Lets do some data cleaning by reseting the index to be numeric as well as dealing with the null values.

In [100]:
```
imdb_title_principals = imdb_title_principals.reset_index()
```

In [101]:
```
imdb_title_principals = imdb_title_principals.drop(columns =['job', 'characte
```

In [102]:
```
imdb_title_principals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 4 columns):
 #   Column    Non-Null Count    Dtype
---  ------    --------------    -----
 0   tconst    1028186 non-null  object
 1   ordering  1028186 non-null  int64
 2   nconst    1028186 non-null  object
 3   category  1028186 non-null  object
dtypes: int64(1), object(3)
memory usage: 31.4+ MB
```

The next step would be to first join the 'imdb_title_principals' and the 'director_writer_df' to find the name-ids of the directors & writers.

In [103]:
```
# Creating a new dataframe with a left join on 'tconst'
director_writer_df_modified = pd.merge(director_writer_df, imdb_title_princip
                                       left_on=  ['tconst'], right_on= ['tcon
                                       how = 'left')
director_writer_df_modified.head()
```

Out[103]:

	tconst	directors	writers	primary_title	original_title	genres	top3_gı
0	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend of the Red Reaper	[Action, Adventure, Fantasy]	
1	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend of the Red Reaper	[Action, Adventure, Fantasy]	
2	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend of the Red Reaper	[Action, Adventure, Fantasy]	
3	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend of the Red Reaper	[Action, Adventure, Fantasy]	
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	Legend of the Red Reaper	[Action, Adventure, Fantasy]	

In [104]:
```
director_writer_df_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115018 entries, 0 to 115017
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   tconst             115018 non-null  object 
 1   directors          115018 non-null  object 
 2   writers            115018 non-null  object 
 3   primary_title      115018 non-null  object 
 4   original_title     115018 non-null  object 
 5   genres             115018 non-null  object 
 6   top3_gross_genres  115018 non-null  object 
 7   ordering           115017 non-null  float64
 8   nconst             115017 non-null  object 
 9   category           115017 non-null  object 
dtypes: float64(1), object(9)
memory usage: 9.7+ MB
```

I am more interested in the nconst and it only has one row of missing value so I am happily removing that. Also I am going to remove all the other columns that are not necessary for the analysis.

In [105]:
```
director_writer_df_modified = director_writer_df_modified.drop(columns = ['or
director_writer_df_modified.head()
```

Out[105]:

	tconst	directors	writers	primary_title	genres	top3_gross_genres	
0	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action, Adventure, Fantasy]	[Action]	
1	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action, Adventure, Fantasy]	[Action]	
2	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action, Adventure, Fantasy]	[Action]	
3	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action, Adventure, Fantasy]	[Action]	
4	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action, Adventure, Fantasy]	[Action]	

In [106]:
```
director_writer_df_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115018 entries, 0 to 115017
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   tconst             115018 non-null  object
 1   directors          115018 non-null  object
 2   writers            115018 non-null  object
 3   primary_title      115018 non-null  object
 4   genres             115018 non-null  object
 5   top3_gross_genres  115018 non-null  object
 6   nconst             115017 non-null  object
 7   category           115017 non-null  object
dtypes: object(8)
memory usage: 7.9+ MB
```

As we can see from the dataframe some directors/ writers also have other roles or professions. I will extract only the roles I am looking for into the data frame and ignore the other professions one may have.

In [107]:
```
director_writer_df_modified.dropna(inplace=True)
```

In [108]:
```
director_writer_df_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 115017 entries, 0 to 115017
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   tconst             115017 non-null  object
 1   directors          115017 non-null  object
 2   writers            115017 non-null  object
 3   primary_title      115017 non-null  object
 4   genres             115017 non-null  object
 5   top3_gross_genres  115017 non-null  object
 6   nconst             115017 non-null  object
 7   category           115017 non-null  object
dtypes: object(8)
memory usage: 7.9+ MB
```

In [109]:
```
# Include data of only a 'writer' or 'directory' in the catgory section
options = ['director', 'writer']
director_writer_df_modified = director_writer_df_modified.loc[
                              director_writer_df_modified['category'].isin(op
```

In [110]:
```
director_writer_df_modified.head()
```

Out[110]:

	tconst	directors	writers	primary_title	genres
5	tt0996958	nm2286991	nm2286991,nm2651190	Legend of the Red Reaper	[Action Adventure Fantasy
15	tt0999913	nm0527109	nm0527109,nm0329051,nm0001603,nm0930684	Straw Dogs	[Action Drama Thriller
16	tt0999913	nm0527109	nm0527109,nm0329051,nm0001603,nm0930684	Straw Dogs	[Action Drama Thriller
17	tt0999913	nm0527109	nm0527109,nm0329051,nm0001603,nm0930684	Straw Dogs	[Action Drama Thriller
18	tt0999913	nm0527109	nm0527109,nm0329051,nm0001603,nm0930684	Straw Dogs	[Action Drama Thriller

In [111]:
```
director_writer_df_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24813 entries, 5 to 115013
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   tconst             24813 non-null  object
 1   directors          24813 non-null  object
 2   writers            24813 non-null  object
 3   primary_title      24813 non-null  object
 4   genres             24813 non-null  object
 5   top3_gross_genres  24813 non-null  object
 6   nconst             24813 non-null  object
 7   category           24813 non-null  object
dtypes: object(8)
memory usage: 1.7+ MB
```

Matching name ids to names and title ids.

Now that we have all the name-ids of the directors and writers it is time to put a name on them. So lets investigate another file with the name of those ids.

In [112]:
```
# assign a variable to the dataframe.
imdb_name_basics = csv_files_dict['imdb_name_basics_gz']
imdb_name_basics
```

Out[112]:

nconst	primary_name	birth_year	death_year	primary_profess
nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_manager,produ
nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,sound_departm
nm0062070	Bruce Baum	NaN	NaN	miscellaneous,actor,wr
nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographer,art_departm
nm0062798	Pete Baxter	NaN	NaN	production_designer,art_department,set_decora
...	...	...	...	
nm9990381	Susan Grobes	NaN	NaN	actr
nm9990690	Joo Yeon So	NaN	NaN	actr
nm9991320	Madeline Smith	NaN	NaN	actr
nm9991786	Michelle Modigliani	NaN	NaN	produ
nm9993380	Pegasus Envoyé	NaN	NaN	director,actor,wr

606648 rows × 5 columns

In [113]:
```
imdb_name_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 606648 entries, nm0061671 to nm9993380
Data columns (total 5 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   primary_name        606648 non-null  object
 1   birth_year          82736 non-null   float64
 2   death_year          6783 non-null    float64
 3   primary_profession  555308 non-null  object
 4   known_for_titles    576444 non-null  object
dtypes: float64(2), object(3)
memory usage: 27.8+ MB
```

In [114]:
```
# drop all the columns that are not essential to the analysis.
imdb_name_basics = imdb_name_basics.drop(columns = ['birth_year','death_year'
```

In [115]:
```
imdb_name_basics
```

Out[115]:

nconst	primary_name	primary_profession
nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer
nm0061865	Joseph Bauer	composer,music_department,sound_department
nm0062070	Bruce Baum	miscellaneous,actor,writer
nm0062195	Axel Baumann	camera_department,cinematographer,art_department
nm0062798	Pete Baxter	production_designer,art_department,set_decorator
...	...	...
nm9990381	Susan Grobes	actress
nm9990690	Joo Yeon So	actress
nm9991320	Madeline Smith	actress
nm9991786	Michelle Modigliani	producer
nm9993380	Pegasus Envoyé	director,actor,writer

606648 rows × 2 columns

In [116]:
```
# Merge the two tables to find the names based on the common 'nconst'
imdb_name_basics_modified = pd.merge(imdb_name_basics, director_writer_df_mod
                                     left_on=  ['nconst'], right_on= ['nconst
                                     how = 'right')
```

In [117]:
```
imdb_name_basics_modified.head()
```

Out[117]:

	nconst	primary_name	primary_profession	tconst	directors	
0	nm2651190	Kim Pritekel	writer,director,actress	tt0996958	nm2286991	
1	nm2651190	Kim Pritekel	writer,director,actress	tt1656191	nm2651190	
2	nm0527109	Rod Lurie	writer,director,producer	tt0999913	nm0527109	nm0527109,nm03290
3	nm0329051	David Zelag Goodman	writer	tt0999913	nm0527109	nm0527109,nm03290
4	nm0001603	Sam Peckinpah	writer,director,producer	tt0999913	nm0527109	nm0527109,nm03290

Data Cleaning

In [118]:
```
imdb_name_basics_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24813 entries, 0 to 24812
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   nconst              24813 non-null  object
 1   primary_name        24813 non-null  object
 2   primary_profession  24750 non-null  object
 3   tconst              24813 non-null  object
 4   directors           24813 non-null  object
 5   writers             24813 non-null  object
 6   primary_title       24813 non-null  object
 7   genres              24813 non-null  object
 8   top3_gross_genres   24813 non-null  object
 9   category            24813 non-null  object
dtypes: object(10)
memory usage: 2.1+ MB
```

What does this data tell me? I see a number of duplicate 'nconst' in the table. A groupby method should give a better insight into the purpose of the duplicates.

In [119]:
```
# Using groupby method to investigate some more.
grouped = imdb_name_basics_modified.groupby(by = 'primary_name').count()
```

In [120]:
```
grouped
```

Out[120]:

primary_name	nconst	primary_profession	tconst	directors	writers	primary_title	genres
42nd Street Pete	1	1	1	1	1	1	1
A Type Machine	1	1	1	1	1	1	1
A'Ali de Sousa	1	1	1	1	1	1	1
A. Deepakraj	5	5	5	5	5	5	5
A. Lee Lee	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...
Ödön von Horvath	1	1	1	1	1	1	1
Ögünç Ersöz	1	1	1	1	1	1	1
Özgür Yildirim	2	2	2	2	2	2	2
Özhan Eren	1	1	1	1	1	1	1
Øystein Karlsen	1	1	1	1	1	1	1

19341 rows × 9 columns

Notice that some of the professionals in the 'primary_name' are known for their roles in more than one title of movie which explains why there were duplicates. But just to be sure I will look into one of the writers.

In [121]:
```
imdb_name_basics_modified.loc[imdb_name_basics_modified['primary_name'] == 'A
```

Out[121]:

	nconst	primary_name	primary_profession	tconst	directors	
1126	nm1189085	A. Deepakraj	writer,assistant_director,miscellaneous	tt6734984	nm2356507	
1127	nm1189085	A. Deepakraj	writer,assistant_director,miscellaneous	tt6522398	nm3008574	nm
1128	nm1189085	A. Deepakraj	writer,assistant_director,miscellaneous	tt3142764	nm2050878	nm
1129	nm1189085	A. Deepakraj	writer,assistant_director,miscellaneous	tt6468814	nm6580663	
1130	nm1189085	A. Deepakraj	writer,assistant_director,miscellaneous	tt2398340	nm3008574	

Alright, that looks good so far. I will go ahead and merge another table 'movie_budgets_top50' with 'imdb_title_basics_df', which we had already worked on, to link the names to the genres.

In [122]:
```
imdb_title_basics_df.head()
```

Out[122]:

	tconst	primary_title	original_title	start_year	genres
0	tt0063540	Sunghursh	Sunghursh	2013	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	Drama
3	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	Comedy,Drama,Fantasy
4	tt0111414	A Thin Life	A Thin Life	2018	Comedy

In [123]:
```
imdb_title_basics_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112232 entries, 0 to 112231
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype 
---  ------          --------------   ----- 
 0   tconst          112232 non-null  object
 1   primary_title   112232 non-null  object
 2   original_title  112232 non-null  object
 3   start_year      112232 non-null  int64 
 4   genres          112232 non-null  object
dtypes: int64(1), object(4)
memory usage: 4.3+ MB
```

In [124]:
```
movie_budgets_df_top50.head()
```

Out[124]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gros
id						
1	Dec 18, 2009	Avatar	425000000	760507625	2776345279	23513
43	Dec 19, 1997	Titanic	200000000	659363944	2208208395	20082
7	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	17481
6	Dec 18, 2015	Star Wars Ep. VII: The Force Awakens	306000000	936662225	2053311220	17473
34	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	14338

In [125]:
```
movie_budgets_df_top50.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 26
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       50 non-null     object
 1   movie              50 non-null     object
 2   production_budget  50 non-null     int64 
 3   domestic_gross     50 non-null     int64 
 4   worldwide_gross    50 non-null     int64 
 5   budget_gross_diff  50 non-null     int64 
 6   release_year       50 non-null     int64 
dtypes: int64(5), object(2)
memory usage: 3.1+ KB
```

In [126]:
```
# Merge the two tables to find out the names involved with the top 50 highest
movie_details_df3 = pd.merge(movie_budgets_df_top50, imdb_title_basics_df,
                             left_on =  ['movie', 'release_year'],
                             right_on= ['primary_title', 'start_year'],
                             how = 'inner')
movie_details_df3.head()
```

Out[126]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
0	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
1	Jun 12, 2015	Jurassic World	215000000	652270625	1648854864	143385
2	Apr 3, 2015	Furious 7	190000000	353007020	1518722794	132872
3	May 4, 2012	The Avengers	225000000	623279547	1517935897	129293
4	Feb 16, 2018	Black Panther	200000000	700059566	1348258224	114825

In [127]:
```
movie_details_df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33 entries, 0 to 32
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       33 non-null     object
 1   movie              33 non-null     object
 2   production_budget  33 non-null     int64 
 3   domestic_gross     33 non-null     int64 
 4   worldwide_gross    33 non-null     int64 
 5   budget_gross_diff  33 non-null     int64 
 6   release_year       33 non-null     int64 
 7   tconst             33 non-null     object
 8   primary_title      33 non-null     object
 9   original_title     33 non-null     object
 10  start_year         33 non-null     int64 
 11  genres             33 non-null     object
dtypes: int64(6), object(6)
memory usage: 3.4+ KB
```

In [128]:
```
# Drop columns not essential for our analysis.
movie_details_df3 = movie_details_df3.drop(columns=
                                           ['primary_title', 'start_year',
                                            'release_year', 'original_title']
                                           axis=1)
```

In [129]:
```
movie_details_df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33 entries, 0 to 32
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   release_date       33 non-null     object
 1   movie              33 non-null     object
 2   production_budget  33 non-null     int64 
 3   domestic_gross     33 non-null     int64 
 4   worldwide_gross    33 non-null     int64 
 5   budget_gross_diff  33 non-null     int64 
 6   tconst             33 non-null     object
 7   genres             33 non-null     object
dtypes: int64(4), object(4)
memory usage: 2.3+ KB
```

Next join the table of names and the budget table to finally make a graph of the revenues those genres return.

In [130]:
```
dir_writer_modified = pd.merge(movie_details_df3, imdb_name_basics_modified,
                               left_on=  ['tconst'], right_on= ['tconst'],
                               how = 'left')
dir_writer_modified.head()
```

Out[130]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
0	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
1	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
2	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
3	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
4	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813

In [131]:
```
dir_writer_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 95 entries, 0 to 94
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   release_date        95 non-null     object
 1   movie               95 non-null     object
 2   production_budget   95 non-null     int64 
 3   domestic_gross      95 non-null     int64 
 4   worldwide_gross     95 non-null     int64 
 5   budget_gross_diff   95 non-null     int64 
 6   tconst              95 non-null     object
 7   genres_x            95 non-null     object
 8   nconst              81 non-null     object
 9   primary_name        81 non-null     object
 10  primary_profession  81 non-null     object
 11  directors           81 non-null     object
 12  writers             81 non-null     object
 13  primary_title       81 non-null     object
 14  genres_y            81 non-null     object
 15  top3_gross_genres   81 non-null     object
 16  category            81 non-null     object
dtypes: int64(4), object(13)
memory usage: 13.4+ KB
```

Before going into further analysis I will have to deal with the columns not valuable to my analysis as well as the null values which would be of no use since they don't contain the necessary data I am looking for.

In [132]:
```
dir_writer_modified = dir_writer_modified.drop(
                                        columns=['top3_gross_genres','genres_
                                        'writers', 'directors'])
```

In [133]:
```
dir_writer_modified.dropna(inplace=True)
```

In [134]:
```
dir_writer_modified.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 81 entries, 0 to 92
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   release_date        81 non-null     object
 1   movie               81 non-null     object
 2   production_budget   81 non-null     int64
 3   domestic_gross      81 non-null     int64
 4   worldwide_gross     81 non-null     int64
 5   budget_gross_diff   81 non-null     int64
 6   tconst              81 non-null     object
 7   genres_x            81 non-null     object
 8   nconst              81 non-null     object
 9   primary_name        81 non-null     object
 10  primary_profession  81 non-null     object
 11  primary_title       81 non-null     object
 12  category            81 non-null     object
dtypes: int64(4), object(9)
memory usage: 8.9+ KB
```

In [135]:
```
dir_writer_modified.head()
```

Out[135]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
0	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
1	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
2	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
3	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
4	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813

In [136]:
```
dir_writer_modified.shape
```

Out[136]: (81, 13)

Select top 10 Directors/ Writers

By arranging the dataframe in descending order I will find the top 10 reccuring directors and writers generating the highest revenue returns.

In [137]:
```
# arrange in descending order by budget_gross_diff.
dir_writer_modified = dir_writer_modified.sort_values(by='budget_gross_diff',
dir_writer_modified.head()
```

Out[137]:

	release_date	movie	production_budget	domestic_gross	worldwide_gross	budget_gross
0	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
4	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
5	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
1	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813
3	Apr 27, 2018	Avengers: Infinity War	300000000	678815482	2048134200	174813

In [138]:
```
# Select top 10 values.
dir_writer_top10 = dir_writer_modified[:10]
dir_writer_top10.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10 entries, 0 to 9
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   release_date        10 non-null     object
 1   movie               10 non-null     object
 2   production_budget   10 non-null     int64 
 3   domestic_gross      10 non-null     int64 
 4   worldwide_gross     10 non-null     int64 
 5   budget_gross_diff   10 non-null     int64 
 6   tconst              10 non-null     object
 7   genres_x            10 non-null     object
 8   nconst              10 non-null     object
 9   primary_name        10 non-null     object
 10  primary_profession  10 non-null     object
 11  primary_title       10 non-null     object
 12  category            10 non-null     object
dtypes: int64(4), object(9)
memory usage: 1.1+ KB
```

In [139]:
```
# storing all the names from the top 10 movies with no duplicates.
all_names = []
for names in dir_writer_top10['primary_name']:
    all_names.append(names)
#
all_names = set(all_names)
all_names
```

Out[139]:
```
{'Amanda Silver',
 'Anthony Russo',
 'Christopher Markus',
 'Colin Trevorrow',
 'Derek Connolly',
 'Jack Kirby',
 'Joe Russo',
 'Rick Jaffa',
 'Stan Lee',
 'Stephen McFeely'}
```

In [140]:
```
# group by primary name of budget_gross_diff median value.
grouped = dir_writer_modified.groupby(by = "primary_name").median()
grouped.info
```

Out[140]:
```
<bound method DataFrame.info of                                   producti
on_budget  domestic_gross  \
primary_name
Amanda Silver                          215000000.0     652270625.0
Anna Boden                             175000000.0     426525952.0
Anthony Russo                          275000000.0     543449915.5
Bob Kane                               275000000.0     448139099.0
Brad Bird                              200000000.0     608581744.0
Chris McKenna                           90000000.0     404508916.0
Chris Morgan                           220000000.0     289385892.5
Chris Van Allsburg                      90000000.0     404508916.0
Chris Weitz                            200000000.0     532177324.0
Christopher Markus                     275000000.0     543449915.5
Christopher Nolan                      275000000.0     448139099.0
Colin Trevorrow                        192500000.0     534995192.5
David Leslie Johnson-McGoldrick        160000000.0     335061807.0
David S. Goyer                         275000000.0     448139099.0
Derek Connolly                         192500000.0     534995192.5
Don Heck                               200000000.0     408992272.0
```

In [141]:
```
#Store key and values of the grouped data as a dictionary.
names_dir_writer = {}

for name in all_names:
    names_dir_writer[name] = grouped.loc[name]['budget_gross_diff']
names_dir_writer
```

Out[141]:
```
{'Jack Kirby': 1072413963.0,
 'Derek Connolly': 1284813831.5,
 'Christopher Markus': 1319101806.5,
 'Amanda Silver': 1433854864.0,
 'Joe Russo': 1319101806.5,
 'Anthony Russo': 1319101806.5,
 'Colin Trevorrow': 1284813831.5,
 'Stephen McFeely': 1319101806.5,
 'Rick Jaffa': 1433854864.0,
 'Stan Lee': 1110336093.5}
```

In [142]:
```
# arrange in descending order and get it ready for visualisation.
names_dir_writer = dict(sorted(names_dir_writer.items(), key = lambda item :
names_dir_writer
```

Out[142]:
```
{'Amanda Silver': 1433854864.0,
 'Rick Jaffa': 1433854864.0,
 'Christopher Markus': 1319101806.5,
 'Joe Russo': 1319101806.5,
 'Anthony Russo': 1319101806.5,
 'Stephen McFeely': 1319101806.5,
 'Derek Connolly': 1284813831.5,
 'Colin Trevorrow': 1284813831.5,
 'Stan Lee': 1110336093.5,
 'Jack Kirby': 1072413963.0}
```

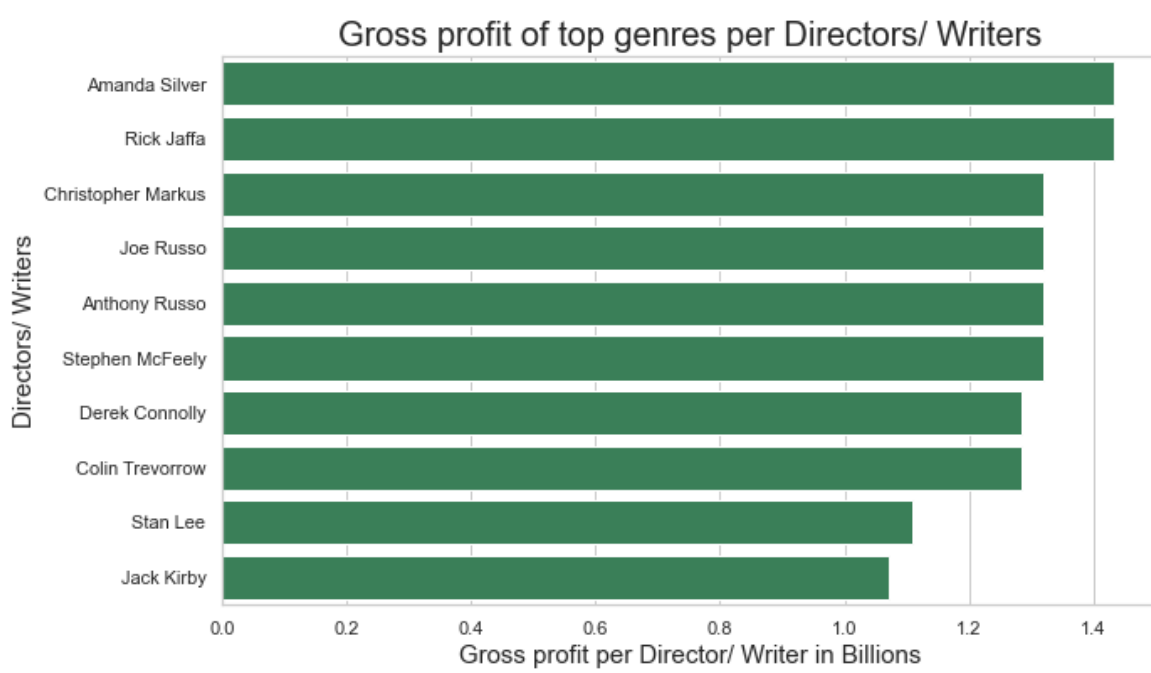
Visualize names vs Gross Profit.

In [143]:
```
keys   = list(names_dir_writer.keys())
values = list(names_dir_writer.values())

sorted_top3_values = sorted(values, key = lambda x: x, reverse = True)[:3]


fig, ax = plt.subplots(figsize=(10,6))
sns.set_theme(style='whitegrid')
sns.barplot(y = keys, x = values, data = dir_writer_top10, color = 'seagreen'

ax.set_xlabel('Gross profit per Director/ Writer in Billions', fontsize=15)
ax.set_ylabel('Directors/ Writers', fontsize=15)
ax.set_title('Gross profit of top genres per Directors/ Writers', fontsize=20
ax.xaxis.offsetText.set_visible(False)
```



This brings us to the conclusion for our second question of 'Who are the most frequent directors in the top 3 grossing genres?' From our findings above I would recommend these directors and writers if Microsoft is to perform well in the movie industry.

- Rick Jaffa
- Amanda Silver
- Stephen McFeely
- Christopher Markus
- Anthony Russo
- Joe Russo
- Derek Connolly
- Colin Trevorrow
- Stan Lee
- Jack Kirby

Question 3. What is the recommended budget investment for Movie production?

Coming to our final question I will try to incorporate all the information and findings gathered above to identify how much is the effective level of investment recommended for Microsoft to enter the movie industry with a bang. Although I would assume the more the production-budget the better the return, I will look into the figures we have and back it up with some facts. First lets explore the budget and return values to get some insight on whether they are correlated.

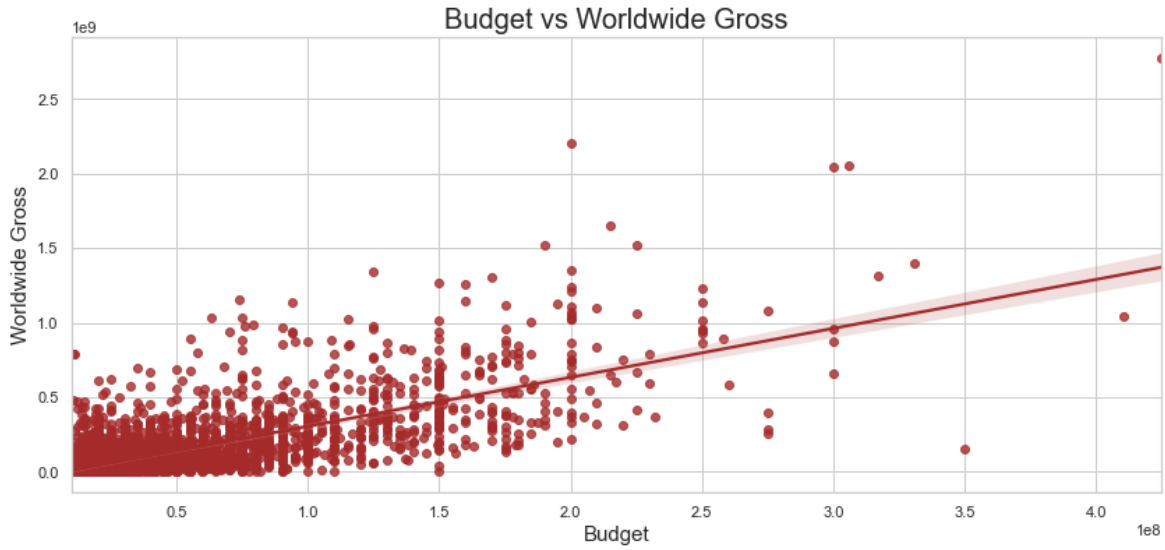
- Does more production budget necessarily mean greater returns?

Analysing Correlation between Budget and Revenue.

Let's plot a 'production budget' vs 'worldwide gross' to comprehend how strongly they are related.

In [144]:
```
fig, ax = plt.subplots(figsize=(14,6))
sns.regplot(x='production_budget', y='worldwide_gross', ax=ax, data=movie_bud
ax.set_title('Budget vs Worldwide Gross', fontsize=20)
ax.set_xlabel('Budget', fontsize=15)
ax.set_ylabel('Worldwide Gross', fontsize=15)
```

Out[144]: Text(0, 0.5, 'Worldwide Gross')



In [145]:
```
# Find the correlation between the budget and the gross profit earned.
production_budget = list(movie_budgets_df['production_budget'])
budget_gross_diff = list(movie_budgets_df['budget_gross_diff'])

corr = np.corrcoef(production_budget,budget_gross_diff)[0][1]
corr = np.round(corr, 2)
```

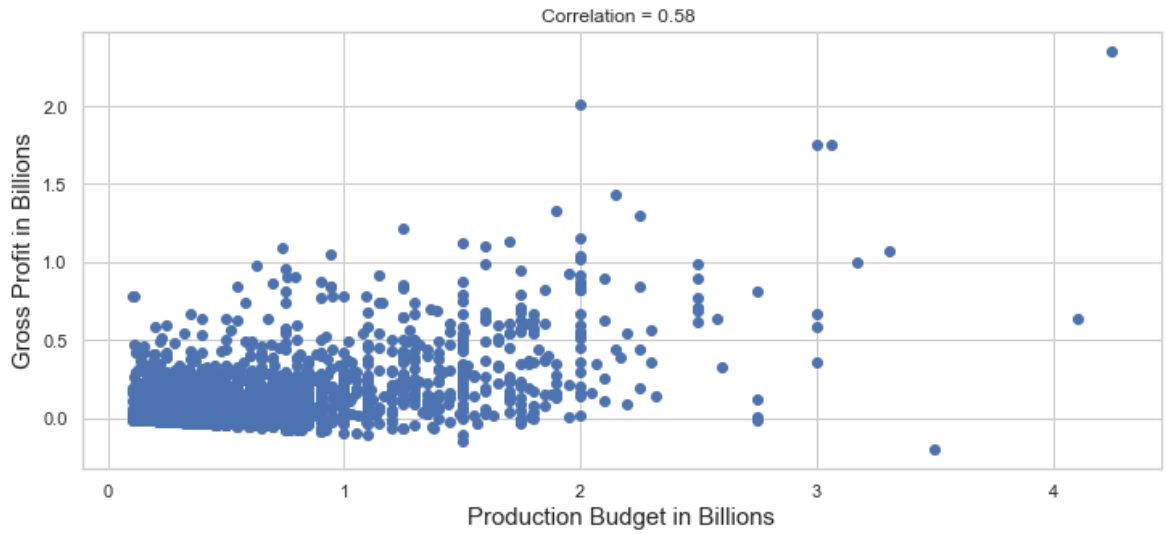
In [146]:
```
corr
```

Out[146]: 0.58

In [147]:
```
x = list(movie_budgets_df['production_budget'])
y = list(movie_budgets_df['budget_gross_diff'])



fig, ax = plt.subplots(figsize=(12, 5))
ax.scatter(x,y)
ax.set_title('Correlation = {} '.format(corr))
ax.set_xlabel('Production Budget in Billions', fontsize=15)
ax.set_ylabel('Gross Profit in Billions', fontsize=15)
ax.xaxis.offsetText.set_visible(False)
ax.yaxis.offsetText.set_visible(False);
```



A Correlation coefficient value of 0.58 signifies a low positive correlation. Therefore, although increasing the investment budget does have a role in the return values, it doesn't necessarily promise highest return or profit. Let's look at other factors responsible for a greater return.

Exploring relationship of Genres, Budget and Revenues.

In this next step looking into the genres and their production budget might give a better understanding of their relationship with the Gross profit. So I will explore more on the median of the production budget and the worldwide gross revenue by replicating each genre in a row and transforming it with the financial columns to a new data frame and hope to find a different perspective on other factors involved.

In [148]:
```
movie_details_df2.drop(movie_details_df2.iloc[:,12:25], inplace=True, axis=1)
```

In [149]:
```
movie_details_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 33 entries, 2 to 49
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   release_date       33 non-null     object 
 1   movie              33 non-null     object 
 2   production_budget  33 non-null     int64  
 3   domestic_gross     33 non-null     int64  
 4   worldwide_gross    33 non-null     int64  
 5   budget_gross_diff  33 non-null     int64  
 6   release_year       33 non-null     int64  
 7   primary_title      33 non-null     object 
 8   original_title     33 non-null     object 
 9   start_year         33 non-null     float64
 10  runtime_minutes    33 non-null     float64
 11  genres             33 non-null     object 
dtypes: float64(2), int64(5), object(5)
memory usage: 4.6+ KB
```

In [150]:
```
# Transform each element of the list-like values to a row, replicating index
# and storing that in a new data frame.

genre_median_df = movie_details_df2.explode('genres').groupby('genres').media
```

In [151]:
```
genre_median_df.head()
```

Out[151]:

genres	production_budget	domestic_gross	worldwide_gross	budget_gross_diff	release_ye
Action	200000000.0	417719760.0	1.215392e+09	986894640.0	201
Adventure	200000000.0	404508916.0	1.123062e+09	928790543.0	201
Animation	122500000.0	368224858.0	1.027971e+09	884323226.0	201
Biography	55000000.0	216303339.0	8.949853e+08	839985342.0	201
Comedy	92500000.0	368224858.0	1.020322e+09	871962904.0	201

In [152]:
```
# Assign 'genre' to all the genres we have available
genre =  genre_median_df.index
```

In [153]:
```
genre_median_df.reset_index(drop=True, inplace=True)
```

In [154]:
```
genre_median_df.insert (0, "genre", genre)
```

In [155]:
```
genre_median_df
```

Out[155]:

	genre	production_budget	domestic_gross	worldwide_gross	budget_gross_diff	rel
0	Action	200000000.0	417719760.0	1.215392e+09	9.868946e+08	
1	Adventure	200000000.0	404508916.0	1.123062e+09	9.287905e+08	
2	Animation	122500000.0	368224858.0	1.027971e+09	8.843232e+08	
3	Biography	55000000.0	216303339.0	8.949853e+08	8.399853e+08	
4	Comedy	92500000.0	368224858.0	1.020322e+09	8.719629e+08	
5	Crime	220000000.0	289385892.0	1.376785e+09	1.156785e+09	
6	Drama	115000000.0	290152231.0	9.289199e+08	8.139199e+08	
7	Family	175000000.0	364001123.0	1.025491e+09	8.254911e+08	
8	Fantasy	180000000.0	334626458.0	1.086193e+09	9.061929e+08	
9	Music	55000000.0	216303339.0	8.949853e+08	8.399853e+08	
10	Musical	180000000.0	419102638.0	1.142345e+09	9.623454e+08	
11	Sci-Fi	205000000.0	442765910.0	1.260583e+09	1.043903e+09	
12	Thriller	225000000.0	328683648.0	1.172687e+09	9.476866e+08	

In [ ]:

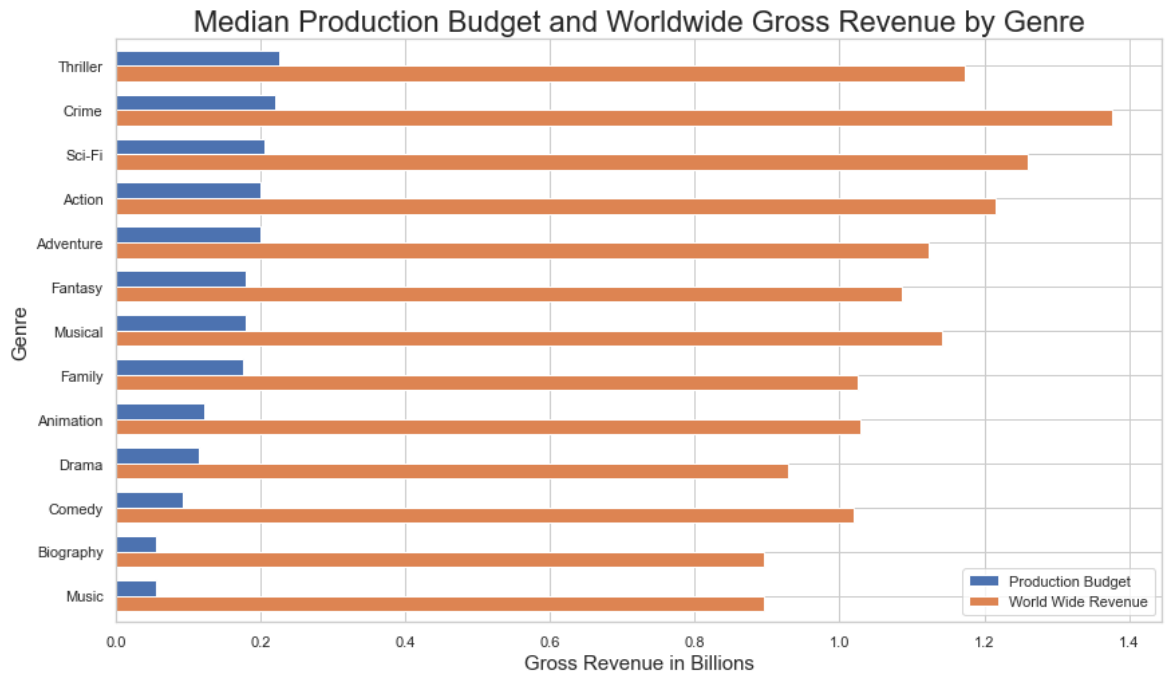
Visualization of Genres versus Gross Revenue.

In [156]:
```
data     = genre_median_df.sort_values('production_budget', ascending=False)
data.plot(x='genre', y=["production_budget", "worldwide_gross"],
          kind="barh",width=0.7, figsize=(14,8), legend='reverse')

plt.legend((["Production Budget", "World Wide Revenue"]))

ax       = plt.gca()
ax.set_title('Median Production Budget and Worldwide Gross Revenue by Genre',
plt.ylabel('Genre',fontsize=15)
plt.xlabel('Gross Revenue in Billions', fontsize=15)
ax.xaxis.offsetText.set_visible(False)
ax.invert_yaxis()


plt.show()
```



Although investing more in producing a film doesn't necessarily mean more returns, investing strategically on specific genres does affect the out come. The top3 genres returning the highest revenues are 'Crime', 'SciFi', and 'Action' as we have seen from our first analysis of Question 1. However, looking at this graph the top3 genres with the most production budget are 'Thriller', 'Crime', and 'Scifi', meaning although much was invested on Thriller movies, it didn't make sufficient returns to be in the top3 highest revenue generating genres.

In [157]:
```
movie_details_df2.describe()
```

Out[157]:

	production_budget	domestic_gross	worldwide_gross	budget_gross_diff	release_year
count	3.300000e+01	3.300000e+01	3.300000e+01	3.300000e+01	33.000000
mean	1.794121e+08	4.084434e+08	1.176263e+09	9.968511e+08	2014.939394
std	6.656082e+07	1.329953e+08	2.430436e+08	2.111769e+08	2.691879
min	5.500000e+07	1.613218e+08	8.797651e+08	7.847651e+08	2010.000000
25%	1.500000e+08	3.350618e+08	1.025491e+09	8.491029e+08	2013.000000
50%	2.000000e+08	4.007380e+08	1.123062e+09	9.287905e+08	2016.000000
75%	2.000000e+08	4.590059e+08	1.259200e+09	1.086336e+09	2017.000000
max	3.306000e+08	7.000596e+08	2.048134e+09	1.748134e+09	2019.000000

In the next steps lets dive in more deeper into the top 3 genres and explore their minimum and 1st quartile production budget. That should give me a better idea of where our recommended budget plan should lie.

Grouping genres by minimum production budget and visualize.

In [158]:
```
genre_min_df = movie_details_df2.explode('genres').groupby('genres').min().rc
genre_min_df.reset_index(inplace=True)
genre_min_df.insert (0, "genre", genre)
```

In [159]:
```
top3_genres_min = genre_min_df.loc[genre_min_df['genres'].isin(['Crime', 'Sci
top3_genres_min
```

Out[159]:

	genre	genres	release_date	movie	production_budget	domestic_gross	worldwide_gro
0	Action	Action	Apr 14, 2017	Aquaman	90000000	225764765	9644961
5	Crime	Crime	Apr 14, 2017	Furious 7	190000000	225764765	12348462
11	Sci-Fi	Sci-Fi	Apr 27, 2018	Avengers: Age of Ultron	170000000	245439076	10491028

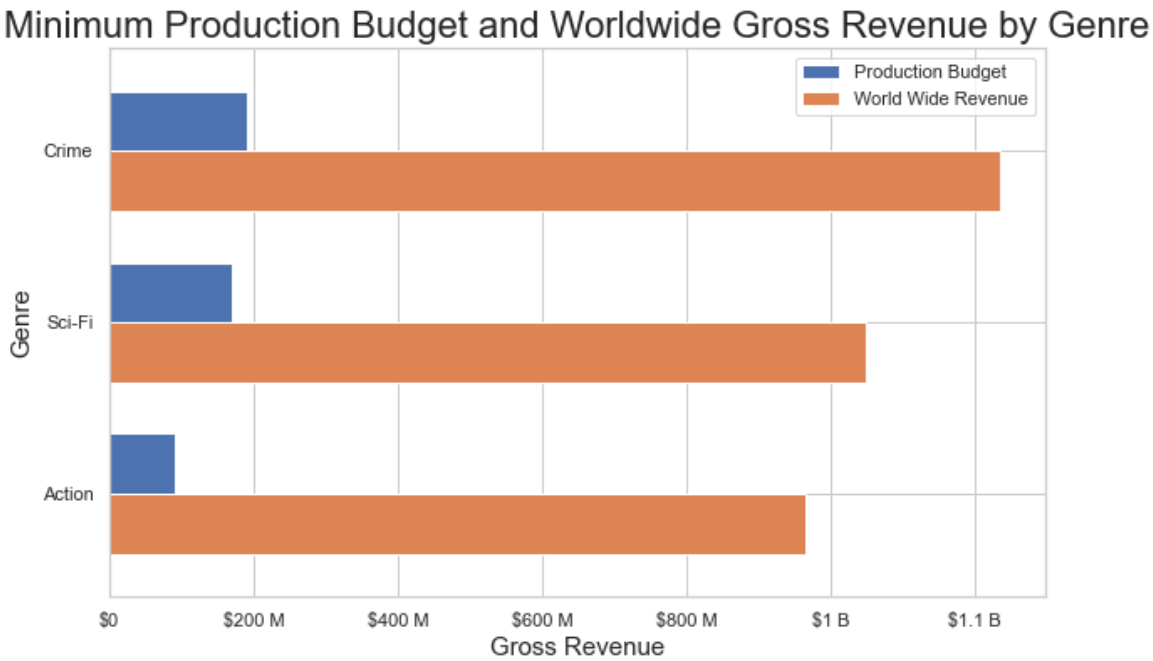
In [160]:
```
data     = top3_genres_min.sort_values('production_budget', ascending=False)
data.plot(x='genre', y=["production_budget", "worldwide_gross"],
          kind="barh",width=0.7, figsize=(10,6), legend='reverse')

plt.legend((["Production Budget", "World Wide Revenue"]))

ax       = plt.gca()
ax.set_title('Minimum Production Budget and Worldwide Gross Revenue by Genre'
ax.set_xticks([0, 2*10**8, 4*10**8, 6*10**8, 8*10**8, 10**9, 12*10**8])
ax.set_xticklabels(['$0', '$200 M', '$400 M', '$600 M', '$800 M', '$1 B', '$1

plt.ylabel('Genre',fontsize=15)
plt.xlabel('Gross Revenue', fontsize=15)
ax.xaxis.offsetText.set_visible(False)
ax.invert_yaxis()


plt.show()
```



Grouping genres by first quartile production budget and visualize.

In [161]:
```
genre_25_df = movie_details_df2.explode('genres').groupby('genres').quantile(
genre_25_df.reset_index(inplace=True)
genre_25_df.insert (0, "genre", genre)
```

In [162]:
```
genre_25_df.head()
```

Out[162]:

	genre	genres	production_budget	domestic_gross	worldwide_gross	budget_gross_d
0	Action	Action	192500000.0	352698782.0	1.116794e+09	902283028
1	Adventure	Adventure	150000000.0	338657009.0	1.023353e+09	869154569
2	Animation	Animation	75250000.0	337351390.0	9.862700e+08	833131275
3	Biography	Biography	55000000.0	216303339.0	8.949853e+08	839985342
4	Comedy	Comedy	75250000.0	337351390.0	9.671764e+08	833131275

In [163]:
```
top3_genres_25 = genre_min_df.loc[genre_min_df['genres'].isin(['Crime', 'Sci-
top3_genres_25
```

Out[163]:

	genre	genres	release_date	movie	production_budget	domestic_gross	worldwide_gro
0	Action	Action	Apr 14, 2017	Aquaman	90000000	225764765	9644961
5	Crime	Crime	Apr 14, 2017	Furious 7	190000000	225764765	12348462
11	Sci-Fi	Sci-Fi	Apr 27, 2018	Avengers: Age of Ultron	170000000	245439076	10491028

In [164]:

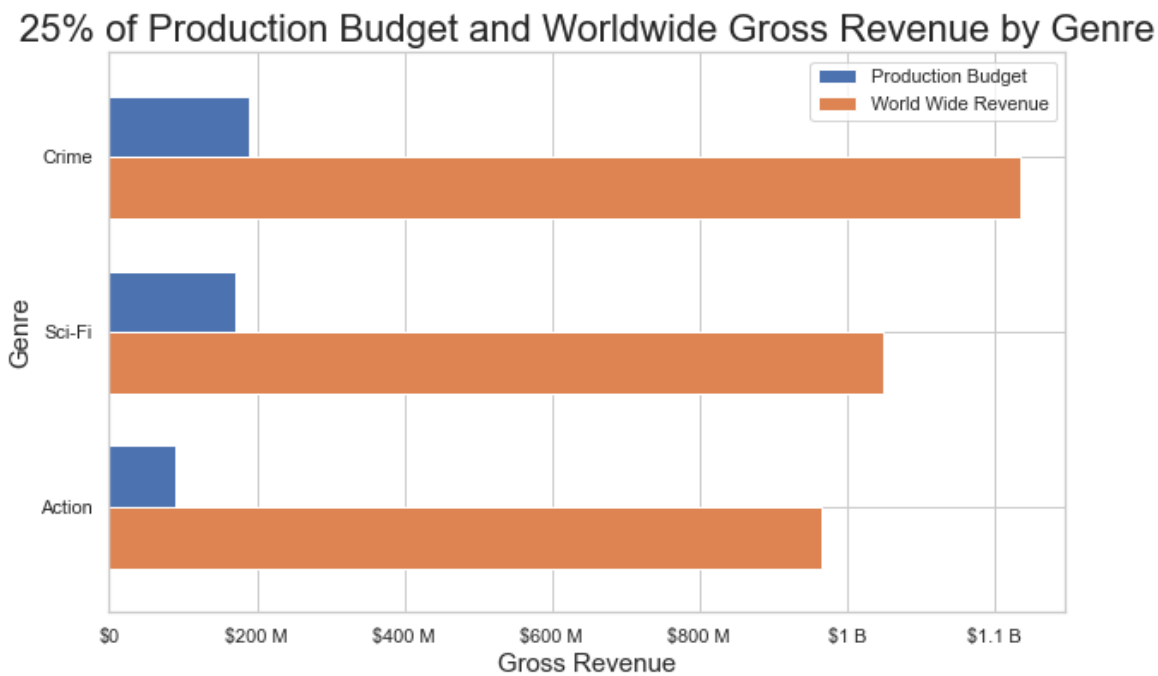
```
data     = top3_genres_25.sort_values('production_budget', ascending=False)
data.plot(x='genre', y=["production_budget", "worldwide_gross"],
          kind="barh",width=0.7, figsize=(10,6), legend='reverse')

plt.legend((["Production Budget", "World Wide Revenue"]))

ax       = plt.gca()
ax.set_title('25% of Production Budget and Worldwide Gross Revenue by Genre',
ax.set_xticks([0, 2*10**8, 4*10**8, 6*10**8, 8*10**8, 10**9, 12*10**8])
ax.set_xticklabels(['$0', '$200 M', '$400 M', '$600 M', '$800 M', '$1 B', '$1

plt.ylabel('Genre',fontsize=15)
plt.xlabel('Gross Revenue', fontsize=15)
ax.xaxis.offsetText.set_visible(False)
ax.invert_yaxis()


plt.show()
```



Finally going over the analysis and looking at the figures above I would recommend the most ideal investment Microsoft would profit from would be a Production Budget between the Minimum and 1st Quartile, i.e. 90 million to 205 million. As noticed from the table those range of budgets manage to generate a Gross profit of atleast 4 times their Production budget.

Summary + Recommendations:

1. Genres 'Crime', 'Action', and 'SciFi' to become the cornerstones of Movie Production.
2. Recruit the following writers and directors.
    - Rick Jaffa
    - Amanda Silver
    - Stephen McFeely
    - Christopher Markus

    - Anthony Russo
    - Joe Russo
    - Derek Connolly
    - Colin Trevorrow
    - Stan Lee
    - Jack Kirby
3. Invest a budget of 90,000,000 - 205,000,000 dollars to compete with some of highest earning production companies.

Further Research

1. Getting insights on how metrics like "popularity" and "rating" would potentially affect the amount of time a customer would spend within a streaming of a movie watching platform, that way aside from the financial gain Microsoft could guarantee keeping loyal customers.
2. Explore the times of year movies are released and look out for any correlation on when they usually have their peak returns on.