Movie Analysis Project

Table of Contents

1. Business Understanding

Background

My company is looking to get into movie creation using their newly created movie studio.

Business Goals

The primary focus of this data science project is to analyze and assess which features of a movie are the most cost efficient. The movie's return on investment will be used to measure cost efficiency in order to make an informed decision regarding what features of movie creation my company should invest in.

Business Success Criteria

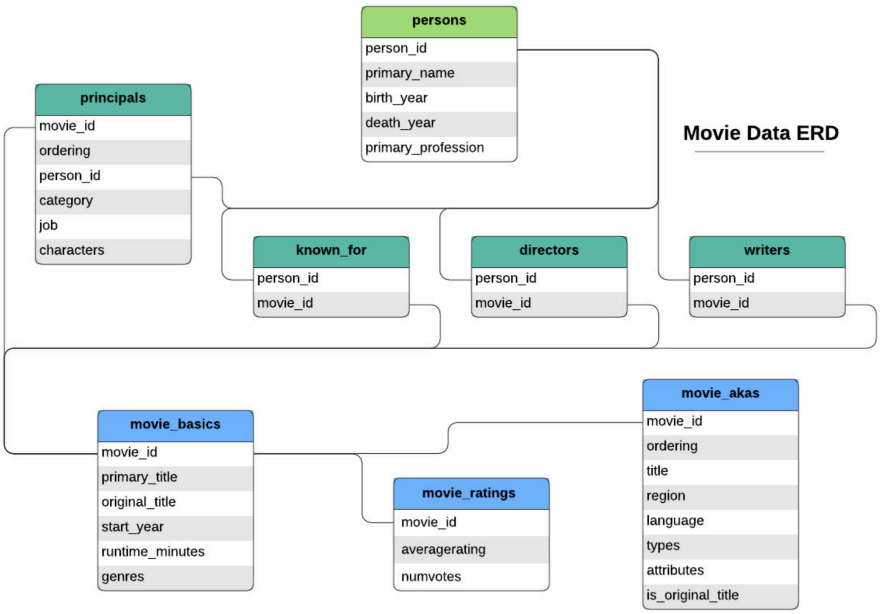
The success of this project will be measured by providing three well-supported recommendations on the most cost efficient movie features (actors, directots, genre, marketing cost, movie rating (G, PG, PG-13, etc.)) to invest in. For this project, the most "cost efficient" features are measured by their return on investment which is defined as 100% times the total revenue divided by the initial investment of the film.

2. Data Understanding

Data on movies is collected by a variety of different sources. For this project, I used data from the following sources:

- The Numbers' `budgets` dataset
  - This dataset includes 6 features and 5782 observations. Each entry in the dataset represents a different movie. For each entry, information is included about the movie's release data, production budget, domestic gross box office, and worldwide gross box office.

- IMDB's film database
  - This database includes 8 tables. Its entity-related diagram (ERD) is shown below.
  - From this database, I used the following tables: `movie_basics` , `persons` , and `principals`
  - `movie_basics` includes **6 features** with **146144 observations**. Each entry in this dataset represents a different movie, where `movie_id` is its unique ID (primary key). Additional information is included about each movie such as `original_title` , `runtime_minutes` , and `genres` .
  - The `persons` table includes **5 features** with **606648 entries.** Each entry represents a person who took part in a movie, where each person has a unique identifier (`person_id`). This table also includes information about each person such as their `primary_name` , `birth_year` , and `primary_professions`
    - The `principals` table contains **6 features** and **1028186** entries, where each entry represents a person who worked in a movie. This table contains two foreign keys (`movie_id` and `person_id`). Additional information includes the `character` the person played and their role on the film (`category`).



In [1]:
```
# importing necessary packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import sqlite3
```

In [2]:
```
#setting up connection to sqlite database

conn = sqlite3.connect('im.db')
cur = conn.cursor()
```

In [3]:
```
# reading budgets csv into a pandas dataframe
budgets = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
budgets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   release_date       5782 non-null   object
 2   movie              5782 non-null   object
 3   production_budget  5782 non-null   object
 4   domestic_gross     5782 non-null   object
 5   worldwide_gross    5782 non-null   object
dtypes: int64(1), object(5)
memory usage: 271.2+ KB
```

From this output, we see that there are 6 columns, and each column has 0 NaN values. The output also shows that the datatypes for `production_budget`, `domestic_gross`, and `worldwide_gross` are objects. Typically in this stage, it would be helpful to visualize the budget data with a histogram to get a quick sense of the distributions, but since the data types are objects instead of integers or floats, trying to plots histograms will return an error.

During data preparation these data types will need to be converted to a type that aggregate functions can be performed on. There is also no column with return on investment data, so I will need to calculate that and include it as an additional column with this dataframe.

In [4]:
```
# creating a dataframe with a list of all tables in the databas

db_tables = pd.read_sql("""

SELECT name
FROM sqlite_master
WHERE type = 'table';

""", conn)

db_tables
```

Out[4]:

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

This output is important because it provides the names of all the tables available in this database. The tables I will be using in this analysis are `movie_basics` , `persons` , and `principals` .

In [5]:
```
# prints the name of each table in the database with the count

table_names = list(db_tables['name'])

tables = {key: None for key in table_names}

for table_name in tables.keys():
    query = f"SELECT COUNT(*) AS num_rows FROM {table_name}"
    tables[table_name] = conn.execute(query).fetchone()[0]

tables
```

Out[5]:
```
{'movie_basics': 146144,
 'directors': 291174,
 'known_for': 1638260,
 'movie_akas': 331703,
 'movie_ratings': 73856,
 'persons': 606648,
 'principals': 1028186,
 'writers': 255873}
```

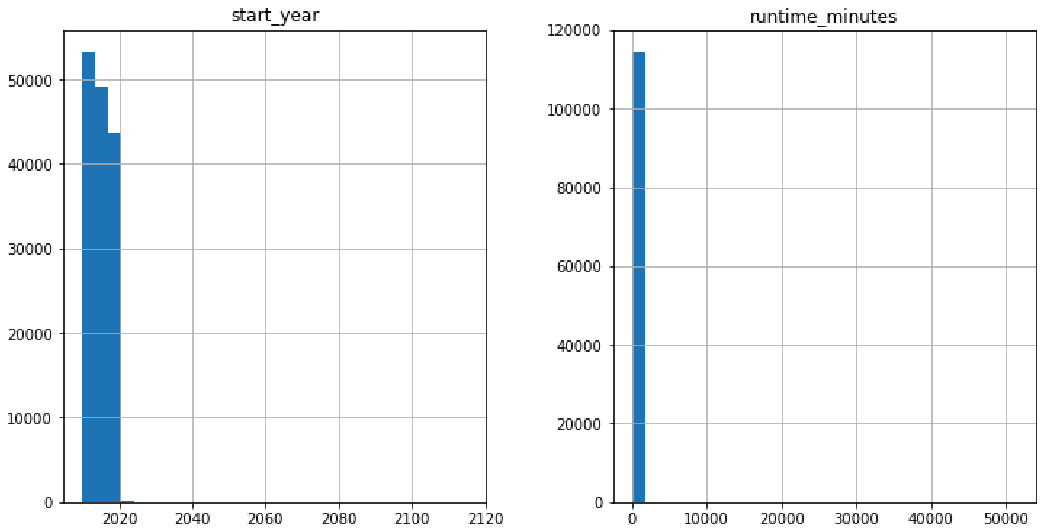
The output provides a quick overview of the number of rows in each table in the database. For example, `movie_basics` has 146,144 rows, while `principals` has 1,028,196 entries. This is important to consider when determining the relationship between tables and the type of join that may work best to optimize processing speeds.

In [6]:
```
# reading the movie_basics table into a pandas dataframe using
movie_basics = pd.read_sql("""

SELECT *
FROM movie_basics

""", conn)

movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

From this output, we see that there are 6 columns and 3 of those columns do not have NaN values. This means that during the Data Preparation phase I will have to inspect the columns with NaN values closely and make a decision about how to proceed with the NaN values.

In [7]:
```
# quickly visualizing the columns of movie_basics that have int
fig, ax = plt.subplots(nrows = 1,
                       ncols = 2,
                       figsize = (12, 6)
                      )
movie_basics.hist(ax = ax,
                  bins = 30
                 );
```



This output provides a quick visualization about the shape of any columns stored as integer or float datatype within the `movie_basics` table. The bars for both `start_year` and `runtime_minutes` appear on the left most portion of the graph. This indicates that each feature has outliers that are much greater than the majority of the data. This means during the Data Preparation phase, I will need to inspect the outliers within each of these columns.

In [8]:
```
# reading the principals table into a pandas dataframe using re
principals = pd.read_sql("""

SELECT *
FROM principals
""", conn)

principals.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 6 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   movie_id    1028186 non-null  object
 1   ordering    1028186 non-null  int64 
 2   person_id   1028186 non-null  object
 3   category    1028186 non-null  object
 4   job         177684 non-null   object
 5   characters  393360 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.1+ MB
```

In [9]:
```
# getting a preview of the principals table
principals.head()
```

Out[9]:

	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"]

From this output, we see that there are 6 columns and the `job` and `category` columns have a large number of NaN values. This means that during the Data Preparation phase I will have to inspect these columns closely and make a decision about how to proceed with the NaN values.

In [10]:
```
# reading the persons table into a pandas dataframe using read_
persons = pd.read_sql("""

SELECT *
FROM persons
""", conn)

persons.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 5 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   person_id           606648 non-null  object
 1   primary_name        606648 non-null  object
 2   birth_year          82736 non-null   float64
 3   death_year          6783 non-null    float64
 4   primary_profession  555308 non-null  object
dtypes: float64(2), object(3)
memory usage: 23.1+ MB
```

In [11]:
```
# getting a preview of the persons table
persons.head()
```

Out[11]:

	person_id	primary_name	birth_year	death_year	pri
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_r
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,s
2	nm0062070	Bruce Baum	NaN	NaN	miscella
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematograph
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_departm

From this output, we see that there are 5 columns and the `birth_year`, `death_year`, and `primary_profession` columns have NaN values. This means that during the Data Preparation phase I will have to inspect these columns closely and make a decision about how to proceed with the NaN values.

3. Data Preparation

During the data preparation stage, I focused on cleaning four datasets: `budgets`, `movie_basics`, `persons`, and `principals`.

The data cleaning process began by converting columns to their appropriate Python data types. To facilitate this, I created a function called `get_info()` to check each table's `.info()`, which allowed me to verify the data types and identify `NaN` values in each feature. For instance, I converted the `production_budget`, `domestic_gross`, and `worldwide_gross` columns in the budgets dataframe from objects to floats, as they were originally stored incorrectly.

I filtered out outliers and included only movies released before 2024. Irrelevant columns, such as `id` in `budgets`, were removed. `NaN` values were addressed by filtering out rows in movie_basics where both `runtime_minutes` and `genres` were `NaN`.

In the `principals` table, I removed the `job`, `characters`, and `ordering` columns due to redundancy or irrelevance, and in the `persons` table, I removed `birth_year`, `death_year`, and `primary_profession` for the same reasons.

After cleaning and processing all four dataframes, I used filtering and join operations to create three new dataframes: `top_people_budgets`, `top_roi_movie_basics`, and `budgets_no_outliers`. These dataframes contain information about movies in the top 25% of ROIs, which I used for further analysis.

- `top_people_budgets` includes information about individuals who worked on movies in the top 25% of ROIs, such as their names, movie titles, and job professions.
- `top_roi_movie_basics` contains details about movies with the highest 25% ROI, including titles, genres, runtimes, and budget information.
- `budgets_no_outliers` provides budget information about all movies from the cleaned dataset.

In [12]:
```
# getting a preview of the dataset
budgets.head(3)
```

Out[12]:

	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

In [13]:
```
# defining columns to format

budget_cols = ['production_budget', 'domestic_gross', 'worldwid
```

In [14]:
```
# formatting budget columns by removing the $
for col in budget_cols:
     budgets[col] = budgets[col].str.replace('$', '')


# formatting budget columns by removing commas
for col in budget_cols:
     budgets[col] = budgets[col].str.replace(',', '')
```

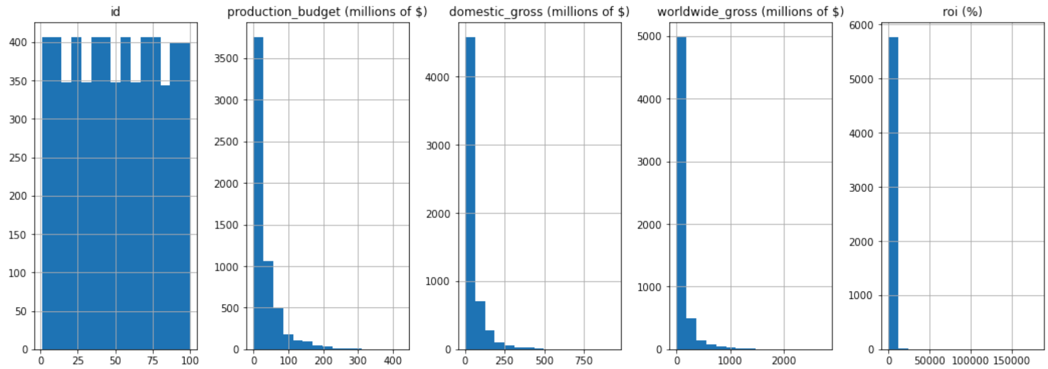
In [15]:
```
# changing dtype of budget and gross columns to int
for col in budget_cols:
     budgets[col] = budgets[col].astype(float)
```

In [16]:
```
# calculating the return on investment for each film and creati
budgets['roi (%)'] = round(((budgets['worldwide_gross']
                            - budgets['production_budget']
                              )
                            / budgets['production_budget']
                              )
                            * 100 , 1
                            )
```

In [17]:
```
#converting dollar amounts to amounts that are easier to read

for col in budget_cols:
    budgets[col] = round(budgets[col]/1000000, 3)
```

In [18]:
```
# renaming columns to include dollar sign

budgets.rename(columns={'production_budget': 'production_budget
```

In [19]:
```
# creating histograms to visualize distributions for each colum
fig, ax = plt.subplots(nrows = 1,
                       ncols = 5,
                       figsize = (18, 6)
                      )
budgets.hist(ax = ax,
             bins = 15
            );
```



Now that the data types of the budget and gross features have been converted to their true data types, histograms can be plotted. The histograms show that the budget and gross columns are skewed right, which means they may contain outliers. This is something to be explored further during the Data Preparation phase.

In [20]:
```
# gets a preview of format changes to budgets df
budgets[:2]
```

Out[20]:

	id	release_date	movie	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gross (millions of $)
0	1	Dec 18, 2009	Avatar	425.0	760.508	2776.345
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410.6	241.064	1045.664

In [21]:
```
# Calculates z-score for each movie's production budget
z = np.abs(stats.zscore(budgets['production_budget (millions of

# Identify production budget outliers as movies with a z-score
outliers = budgets[z > 3]

# Print the production budget outliers
outliers
```

Out[21]:

	id	release_date	movie	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gros (millions of
0	1	Dec 18, 2009	Avatar	425.0	760.508	2776.34
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	410.6	241.064	1045.66
2	3	Jun 7, 2019	Dark Phoenix	350.0	42.762	149.76
3	4	May 1, 2015	Avengers: Age of Ultron	330.6	459.006	1403.01
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	317.0	620.181	1316.72
...	...	...	...	...	...	
141	42	Jun 3, 2011	X-Men: First Class	160.0	146.408	355.40
142	43	Dec 25, 2008	The Curious Case of Benjamin Button	160.0	127.509	329.63
143	44	Jul 14, 2010	The Sorcerer's Apprentice	160.0	63.151	217.98
144	45	May 12, 2006	Poseidon	160.0	60.675	181.67
145	46	Jun 10, 2016	Warcraft	160.0	47.226	425.52

146 rows × 7 columns

In [22]:
```
# creating a new df of budgets with production budget outliers
budgets_no_outliers = budgets.drop(outliers.index)

# checking that the correct number of rows were removed
len(budgets) - len(budgets_no_outliers)
```

Out[22]:
```
146
```

In [23]:
```
budgets_no_outliers['id'].value_counts()
```

Out[23]:
```
74    57
54    57
68    57
72    57
76    57
      ..
30    56
34    56
38    56
42    56
4     56
Name: id, Length: 100, dtype: int64
```

In [24]:
```
# inspecting all entries with id of '4'
budgets_no_outliers[budgets_no_outliers['id'] == 4].head(15)
```

Out[24]:

	id	release_date	movie	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_g (millions
203	4	Jul 15, 2016	Ghostbusters	144.0	128.351	22
303	4	May 19, 1999	Star Wars Ep. I: The Phantom Menace	115.0	474.545	102
403	4	Dec 14, 2018	Mortal Engines	100.0	15.951	8
503	4	Sep 29, 2006	Open Season	85.0	85.105	19
603	4	Dec 25, 1997	The Postman	80.0	17.651	2
703	4	Aug 8, 2003	S.W.A.T.	70.0	116.878	20
803	4	Sep 14, 2012	Resident Evil: Retribution	65.0	42.346	24
903	4	Jan 11, 2013	Gangster Squad	60.0	46.001	10
1003	4	Apr 10, 1998	City of Angels	55.0	78.751	19
1103	4	Aug 9, 2013	Disney Planes	50.0	90.283	23
1203	4	Mar 15, 2013	Upside Down	50.0	0.102	2
1303	4	Oct 4, 1996	The Glimmer Man	45.0	20.405	3
1403	4	Feb 17, 2006	Eight Below	40.0	81.613	12
1503	4	Apr 21, 1995	Kiss of Death	40.0	14.942	1
1603	4	Dec 31, 2009	Obitaemyy ostrov	36.5	0.000	1

In [25]:
```
# dropping id column from dataset
budgets_no_outliers.drop(['id'], axis=1, inplace=True)
budgets_no_outliers.head(3)
```

Out[25]:

	release_date	movie	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gross (millions of $)
146	Sep 30, 2016	Deepwater Horizon	156.0	61.434	122.604
147	Dec 10, 2010	The Chronicles of Narnia: The Voyage of the Da...	155.0	104.387	418.187
148	Jul 1, 2015	Terminator: Genisys	155.0	89.761	432.151

In [26]:
```
# checking for duplicate movie entries
budgets_no_outliers.movie.value_counts()
```

Out[26]:
```
Halloween                 3
Home                      3
Ben-Hur                   2
Shaft                     2
The Last House on the Left    2
                         ..
Elephant                  1
The Quiet American        1
My Own Private Idaho      1
Pink Ribbons, Inc.        1
Mr. Beanâ€s Holiday      1
Name: movie, Length: 5562, dtype: int64
```

In [27]:
```
# checking for any movie titles with more than one entry to see

multiple_values = budgets_no_outliers.movie.value_counts()
budgets_no_outliers[budgets_no_outliers.movie.isin(multiple_val
```

Out[27]:

	release_date	movie	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gros (millions of $
203	Jul 15, 2016	Ghostbusters	144.000	128.351	229.00
243	Mar 27, 2015	Home	130.000	177.398	385.99
267	Aug 8, 2014	Teenage Mutant Ninja Turtles	125.000	191.205	485.00
271	Apr 1, 2010	Clash of the Titans	125.000	163.215	493.21
278	Aug 3, 2012	Total Recall	125.000	58.878	211.85
...	...	...	...	...	..
5668	Nov 16, 1942	Cat People	0.134	4.000	8.00
5676	Oct 1, 1968	Night of the Living Dead	0.114	12.087	30.08
5677	Feb 8, 1915	The Birth of a Nation	0.110	10.000	11.00
5699	Aug 30, 1972	The Last House on the Left	0.087	3.100	3.10
5718	Feb 22, 2008	The Signal	0.050	0.251	0.40

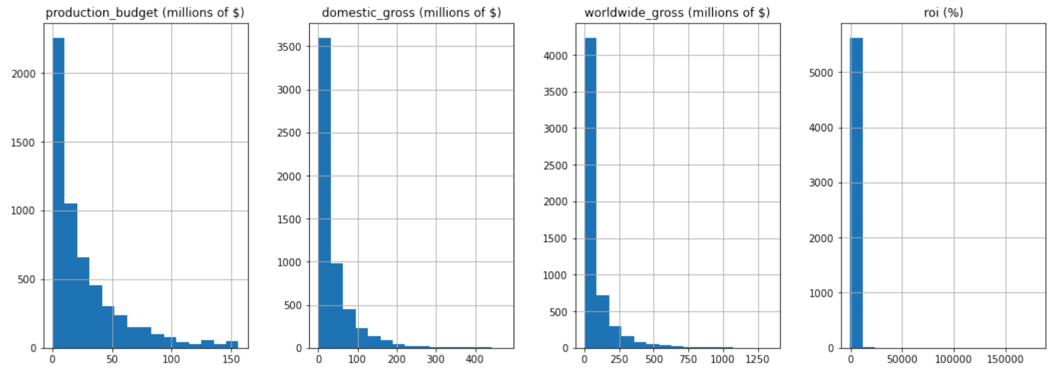
146 rows × 6 columns

In [28]:
```
# getting summary statistics for budgets df columns
round(budgets_no_outliers.describe(), 1)
```

Out[28]:

	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gross (millions of $)	roi (%)
count	5636.0	5636.0	5636.0	5636.0
mean	27.2	36.6	75.9	383.6
std	31.0	54.6	129.8	2990.8
min	0.0	0.0	0.0	-100.0
25%	5.0	1.2	3.7	-53.5
50%	16.0	16.0	26.1	66.4
75%	38.0	48.6	89.2	269.9
max	156.0	474.5	1341.7	179900.0

In [29]:
```
# creating histograms to visualize distributions for each colum
fig, ax = plt.subplots(nrows = 1,
                       ncols = 4,
                       figsize = (18, 6)
                      )

budgets_no_outliers.hist(ax = ax,
                         bins = 15
                        );
```



With the budgest data set cleaned and processed the histograms display more helpful information. The histograms still appear right skewed even though production budget outliers were removed, but this is still an improvement from earlier. Now the tails for each feature are more prominent, and we can be more confident that analysis of the budgets data will lead reliable results.

In [30]:
```
movie_basics.head(3)
```

Out[30]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genr
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drar
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drar
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drar

In [31]:
```
# returning preview of the rows of movie_basics for movies with
movie_basics[movie_basics['runtime_minutes'] > (movie_basics['r
```

Out[31]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	
6311	tt1277455	A Time to Stir	A Time to Stir	2018	1320.0	
12974	tt1674154	City of Eternal Spring	City of Eternal Spring	2010	3450.0	Documentary
15381	tt1735956	Deregulation	Foreclose	2012	4200.0	C

In [32]:
```
# returning the rows of movie_basics where the movie start year
movie_basics[movie_basics['start_year'] > 2024]
```

Out[32]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	
2949	tt10300398	Untitled Star Wars Film	Untitled Star Wars Film	2026	NaN	
52213	tt3095356	Avatar 4	Avatar 4	2025	NaN	Action,Adve
89506	tt5174640	100 Years	100 Years	2115	NaN	
96592	tt5637536	Avatar 5	Avatar 5	2027	NaN	Action,Adve

In [33]:
```
# removing any rows with movies whose start year is past 2024
movie_basics = movie_basics[movie_basics['start_year'] <= 2024]

# checks to ensure that rows were removed
movie_basics[movie_basics['start_year'] > 2024]
```

Out[33]:

movie_id	primary_title	original_title	start_year	runtime_minutes	genres

In [34]:
```
# comparing movies whose primary title does not match its origi

movie_basics[movie_basics['primary_title'] != movie_basics['ori
```

Out[34]:

	primary_title	original_title
1	One Day Before the Rainy Season	Ashad Ka Ek Din
4	The Wandering Soap Opera	La Telenovela Errante
11	So Much for Justice!	Oda az igazság
13	Children of the Green Dragon	A zöld sárkány gyermekei
15	The Tragedy of Man	Az ember tragédiája
...	...	...
146026	Journey of the Sky Goddess	Kibaiyanse! Watashi
146028	Lupin the Third: Fujiko Mine's Lie	Lupin the IIIrd: Mine Fujiko no Uso
146037	Big Three Dragons	Da San Yuan
146121	A Cherry Tale	Kirsebæreventyret
146135	The Rehearsal	O Ensaio

14504 rows × 2 columns

In [35]:
```
# getting count of all movies whose primary title doesn't match
movie_basics[movie_basics['primary_title'] != movie_basics['ori
```

Out[35]:
```
La traversée                3
Ici-bas                     2
Oro                         2
Ban shou shao nu            2
Sakura saku                 2
                           ..
Du yi wu er                 1
Onze Jongens                1
Raiâ gêmu: Za fainaru sutêji    1
Deadtime Stories 2          1
Ozen                        1
Name: original_title, Length: 14452, dtype: int64
```

In [36]:
```
# removing original title column
movie_basics.drop('original_title',
                  axis=1,
                  inplace=True
                 )
```

In [37]:
```
# get a count for the number of each genre

movie_basics['genres'].str.split(',').explode().value_counts()
```

Out[37]:
```
Documentary    51640
Drama          49882
Comedy         25312
Thriller       11883
Horror         10805
Action         10333
Romance         9372
Biography       8722
Crime           6753
Adventure       6463
Family          6227
History         6225
Mystery         4659
Music           4314
Fantasy         3513
Sci-Fi          3365
Animation       2799
Sport           2234
News            1551
Musical         1430
War             1405
Western          467
Reality-TV        98
Talk-Show         50
Adult             25
Short             11
Game-Show          4
Name: genres, dtype: int64
```

In [38]:
```
# formatting genres column to create a list containing each gen
movie_basics["genres_list"] = movie_basics["genres"].str.split(
```

In [39]:
```
# display number of nan values for each feature
movie_basics.isna().sum()
```

Out[39]:
```
movie_id               0
primary_title          0
start_year             0
runtime_minutes    31735
genres              5408
genres_list         5408
dtype: int64
```

In [40]:
```
# return the number of rows with nans in both genre and runtime
len(movie_basics.loc[movie_basics['genres'].isnull()
                     & movie_basics['runtime_minutes'].isnull()
                    ]
   )
```

Out[40]:
```
3236
```

In [41]:
```
# removing rows from movie_basics with null genre and runtime
movie_basics_clean = movie_basics.drop(movie_basics.loc[movie_b
                                            & movie_basics['runtim
                                         )
movie_basics_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 142904 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         142904 non-null  object 
 1   primary_title    142904 non-null  object 
 2   start_year       142904 non-null  int64  
 3   runtime_minutes  114405 non-null  float64
 4   genres           140732 non-null  object 
 5   genres_list      140732 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.6+ MB
```

In [42]:
```
# confirming that correct number of rows were filtered out
len(movie_basics) - len(movie_basics_clean)
```

Out[42]:
```
3236
```

In [43]:
```
# prints the number of NaNs in each column of movie_basics_clea
movie_basics_clean.isna().sum()
```

Out[43]:
```
movie_id               0
primary_title          0
start_year             0
runtime_minutes    28499
genres              2172
genres_list         2172
dtype: int64
```

In [44]:
```
principals.person_id.value_counts()
```

Out[44]:
```
nm1930572     378
nm0000636     160
nm0000616     148
nm0103977     126
nm4394575     103
             ...
nm10306370      1
nm9301008       1
nm5456973       1
nm5912469       1
nm3636291       1
Name: person_id, Length: 604546, dtype: int64
```

In [45]:
```
principals[principals['person_id'] == 'nm4394575']
```

Out[45]:

	movie_id	ordering	person_id	category	job	characters
209253	tt2414424	9	nm4394575	editor	None	None
284938	tt2418914	7	nm4394575	editor	None	None
291705	tt3445098	9	nm4394575	editor	None	None
341007	tt2419230	8	nm4394575	editor	None	None
344804	tt2410964	8	nm4394575	editor	None	None
...	...	...	...	...	...	...
983231	tt9525226	7	nm4394575	editor	None	None
984125	tt6370780	8	nm4394575	editor	None	None
984135	tt6374832	8	nm4394575	editor	None	None
1012364	tt7843050	6	nm4394575	editor	None	None
1027958	tt9520500	7	nm4394575	editor	None	None

103 rows × 6 columns

This output shows that duplicate `person_id` 's correspond to the same person working on different movies. This can be seen because the same `person_id` can be found in rows with different `movie_id` 's.

In [46]:
```
principals['category'].value_counts()
```

Out[46]:
```
actor                  256718
director               146393
actress                146208
producer               113724
cinematographer         80091
composer                77063
writer                  74357
self                    65424
editor                  55512
production_designer      9373
archive_footage          3307
archive_sound              16
Name: category, dtype: int64
```

In [47]:
```
principals['job'].value_counts()
```

Out[47]:
```
producer                                       108168
screenplay                                       8172
director of photography                          6517
writer                                           6479
co-director                                      5796
                                                 ...
novel American Woman                                1
play "The Crucible" by                              1
based on the novel 'Phoenix Hunting' by             1
Producer in Isla Mujeres Mexico                     1
book "The Princess of Suburbia                      1
Name: job, Length: 2965, dtype: int64
```

In [48]:
```
principals['characters'].value_counts()
```

Out[48]:
```
["Himself"]                       43584
["Herself"]                       16127
["Narrator"]                       2218
["Alex"]                            656
["David"]                           620
                                  ...
["Dennis Pehlke"]                     1
["Brittany Sanders"]                  1
["Wingnut"]                           1
["Wommie"]                            1
["Barack Obama - Narrator"]           1
Name: characters, Length: 174762, dtype: int64
```

In [49]:
```
# Dropping job and character columns

principals.drop(['job', 'characters', 'ordering'], axis=1, inpl
```

In [50]:
```
# get a count for the number of each profession
persons['primary_profession'].str.split(',').explode().value_co
```

Out[50]:
```
actor                       177838
producer                    150214
writer                      141504
director                    129808
actress                      95066
cinematographer              61984
editor                       55234
composer                     48823
camera_department            39466
miscellaneous                38661
soundtrack                   20748
music_department             18568
assistant_director           15916
sound_department             15280
editorial_department         14565
production_manager            9768
art_department                8913
production_designer           7592
visual_effects                6188
art_director                  4623
stunts                        4309
casting_department            2802
executive                     2657
make_up_department            2613
animation_department          2459
casting_director              2397
location_management           2253
costume_department            1938
special_effects               1856
costume_designer              1548
set_decorator                 1435
manager                        732
transportation_department      673
talent_agent                   313
legal                           92
publicist                       83
assistant                       72
Name: primary_profession, dtype: int64
```

In [51]:
```
# dropping birth_year, death_year, and primary_profression from
persons.drop(['birth_year', 'death_year', 'primary_profession']
```

In [52]:
```
# function that provides the .info() for the remaining three da
data = [movie_basics_clean, principals, persons]
def get_info(dataframes):
    for df in dataframes:
        print("--------")
        print()
        print(df.info())
```

In [53]:
```
get_info(data)
```

```
--------

<class 'pandas.core.frame.DataFrame'>
Int64Index: 142904 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   movie_id         142904 non-null  object
 1   primary_title    142904 non-null  object
 2   start_year       142904 non-null  int64
 3   runtime_minutes  114405 non-null  float64
 4   genres           140732 non-null  object
 5   genres_list      140732 non-null  object
dtypes: float64(1), int64(1), object(4)
memory usage: 7.6+ MB
None
--------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 3 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   movie_id   1028186 non-null  object
 1   person_id  1028186 non-null  object
 2   category   1028186 non-null  object
dtypes: object(3)
memory usage: 23.5+ MB
None
--------

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   person_id     606648 non-null  object
 1   primary_name  606648 non-null  object
dtypes: object(2)
memory usage: 9.3+ MB
None
```

The output from the get_info() function shows that in each table, each column's python data type matches its true data type. The `principals` and `persons` tables have zero NaN values. In the `movie_basics_clean` DataFrame, the following three columns still have a some NaN values: `runtime_minutes`, `genres`, `genres_list`. Since there is significantly less data in the budgets table, I keep the rows with the NaN values. This will help ensure that I do not lose any important data when joining the budgets table with the `movie_basics_clean` table.

In [54]:
```
# joining principals and persons dfs

film_people = pd.merge(persons,
                       principals,
                       how = 'inner',
                       on = 'person_id'
                      )

film_people.head(3)
```

Out[54]:

	person_id	primary_name	movie_id	category
0	nm0061671	Mary Ellen Bauder	tt2398241	producer
1	nm0061865	Joseph Bauer	tt0433397	composer
2	nm0061865	Joseph Bauer	tt1681372	composer

In [55]:
```
film_people.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1027912 entries, 0 to 1027911
Data columns (total 4 columns):
 #   Column        Non-Null Count    Dtype
---  ------        --------------    -----
 0   person_id     1027912 non-null  object
 1   primary_name  1027912 non-null  object
 2   movie_id      1027912 non-null  object
 3   category      1027912 non-null  object
dtypes: object(4)
memory usage: 39.2+ MB
```

In [56]:
```
# get counts of the different categories of film professions
film_people.category.value_counts()
```

Out[56]:
```
actor                  256561
director               146393
actress                146124
producer               113724
cinematographer         80091
composer                77063
writer                  74357
self                    65392
editor                  55512
production_designer      9373
archive_footage          3306
archive_sound              16
Name: category, dtype: int64
```

In [57]:
```
film_people.person_id.value_counts()
```

Out[57]:
```
nm1930572    378
nm0000636    160
nm0000616    148
nm0103977    126
nm4394575    103
            ...
nm0632884      1
nm3232305      1
nm8039614      1
nm3943132      1
nm2282568      1
Name: person_id, Length: 604290, dtype: int64
```

In [58]:
```
film_people[film_people['person_id'] == 'nm4394575']
```

Out[58]:

	person_id	primary_name	movie_id	category
415221	nm4394575	Sen Arima	tt2414424	editor
415222	nm4394575	Sen Arima	tt2418914	editor
415223	nm4394575	Sen Arima	tt3445098	editor
415224	nm4394575	Sen Arima	tt2419230	editor
415225	nm4394575	Sen Arima	tt2410964	editor
...	...	...	...	...
415319	nm4394575	Sen Arima	tt9525226	editor
415320	nm4394575	Sen Arima	tt6370780	editor
415321	nm4394575	Sen Arima	tt6374832	editor
415322	nm4394575	Sen Arima	tt7843050	editor
415323	nm4394575	Sen Arima	tt9520500	editor

103 rows × 4 columns

The above output shows that each row in `film_people` represents a movie that a professional was involved in. For example, the `person_id` that corresponds with `nm4394575` is represented in 103 rows, with each row having a different `movie_id`.

In [59]:
```
film_people.value_counts()
```

Out[59]:
```
person_id  primary_name     movie_id   category
nm3296031  Brendan Maclean  tt2815358  actor       2
nm2442121  Ivy Yi-Han Chen  tt8942260  actress     2
nm4454963  Mike Kai Sui     tt6450032  actor       2
nm3206691  Hasan Majuni     tt2258513  actor       2
nm1289422  Barbara Bacci    tt3153648  actress     2
                                                  ..
nm4885985  Rémi Goulet      tt7610830  actor       1
                            tt4027334  actor       1
                            tt2224307  actor       1
nm4885974  Mónica Portillo  tt6207386  actress     1
nm0000002  Lauren Bacall    tt0858500  actress     1
Length: 1027877, dtype: int64
```

In [60]:
```
# removes duplicate rows that share the same movie_id and perso

film_people_duplicates = film_people.duplicated(keep = False)
film_people[film_people_duplicates].value_counts()
```

Out[60]:
```
person_id  primary_name               movie_id   category
nm9161308  Sameer Deshpande           tt5489544  actor       2
nm1289422  Barbara Bacci              tt3153648  actress     2
nm3296031  Brendan Maclean            tt2815358  actor       2
nm3206691  Hasan Majuni               tt2258513  actor       2
nm3187984  Abdellatif Chaouqi         tt3592504  actor       2
nm2442121  Ivy Yi-Han Chen            tt8942260  actress     2
nm2335900  Justin Malone              tt1995481  actor       2
nm1794850  David Chalmers             tt2073120  actor       2
nm1141719  Nobuyuki Kase              tt5098626  actor       2
nm3548929  Liuyuan Ding               tt5338100  actress     2
nm0857847  Thich Nhát Hanh            tt5268106  actor       2
nm0849468  Masashi Taniguchi          tt8108180  actor       2
nm0605583  Robert Morin               tt6664852  actor       2
nm0406809  Kunihiko Ida               tt5495582  actor       2
nm0244327  Dorra Zarrouk              tt6549064  actress     2
nm0172826  Luigi Cozzi                tt4537170  actor       2
nm3414469  Mita Chatterjee            tt5282110  actress     2
nm3741291  JC Cadena                  tt7180088  actress     2
nm9115981  Andy Johnson               tt7236082  editor      2
nm5241644  Yu Li                      tt2473710  actor       2
nm8204953  Nikhil Chaudhary           tt6094992  producer    2
nm7129726  Chloe Brown                tt5974592  actress     2
nm6523411  Sereene Brown              tt4472884  actress     2
nm5992239  Shawan Emer                tt3246048  actress     2
nm5726235  Dorian Kane                tt7725546  actor       2
nm5241644  Yu Li                      tt6419578  actor       2
nm4721563  Kristen StephensonPino     tt2557902  actress     2
nm3772098  Ross Everett               tt2368182  actor       2
nm4454963  Mike Kai Sui               tt6450032  actor       2
nm3996622  Thomas Brenneck            tt5613920  actor       2
nm3979007  Rameet Sandhu              tt5805424  actress     2
nm3895623  Jacqueline Chong           tt6423408  actress     2
                                      tt3503838  actress     2
nm3782659  Ece Baykal                 tt6971730  actress     2
nm0149828  Sudiptaa Chakraborty       tt5473578  actress     2
dtype: int64
```

In [61]:
```
# checks to confirm that there are no duplicates
film_people_no_dups = film_people.drop_duplicates()
film_people_no_dups.value_counts()
```

Out[61]:
```
person_id  primary_name              movie_id    category
nm9993680  Christopher-Lawson Palmer tt10427366  actor
1
nm1822501  Reece Rios                tt1591509   actor
1
nm1822600  Anastas Tanovski          tt7610008   actor
1
nm1822582  Claude Stark              tt6046566   actor
1
nm1822570  Esham                     tt2006716   self
1


..
nm4886005  Abi Alberto               tt2224159   director
1
nm4885998  Mott Green                tt2224377   self
1
nm4885985  Rémi Goulet               tt7610830   actor
1
                                     tt4027334   actor
1
nm0000002  Lauren Bacall             tt0858500   actress
1
Length: 1027877, dtype: int64
```

Type *Markdown* and LaTeX: α^2

In [62]:
```
film_people_no_dups.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1027877 entries, 0 to 1027911
Data columns (total 4 columns):
 #   Column        Non-Null Count    Dtype 
---  ------        --------------    ----- 
 0   person_id     1027877 non-null  object
 1   primary_name  1027877 non-null  object
 2   movie_id      1027877 non-null  object
 3   category      1027877 non-null  object
dtypes: object(4)
memory usage: 39.2+ MB
```

In [63]:
```
# inner joining film_people_no_dups and movie_basics_clean usin

film_people_with_movies = pd.merge(film_people_no_dups,
                                   movie_basics_clean,
                                   how = 'inner',
                                   on = 'movie_id'
                                  )
```

In [64]:
```
# Dropping job and character columns

film_people_with_movies.drop('runtime_minutes',
                             axis=1,
                             inplace=True
                            )
```

In [65]:
```
film_people_with_movies.head()
```

Out[65]:

	person_id	primary_name	movie_id	category	primary_title	start_year	
0	nm0061671	Mary Ellen Bauder	tt2398241	producer	Smurfs: The Lost Village	2017	Adventure
1	nm0038432	Kelly Asbury	tt2398241	director	Smurfs: The Lost Village	2017	Adventure
2	nm0449549	Jordan Kerner	tt2398241	producer	Smurfs: The Lost Village	2017	Adventure
3	nm0962596	Pamela Ribon	tt2398241	writer	Smurfs: The Lost Village	2017	Adventure
4	nm0678963	Peyo	tt2398241	writer	Smurfs: The Lost Village	2017	Adventure

In [66]:
```
film_people_with_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017239 entries, 0 to 1017238
Data columns (total 8 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   person_id      1017239 non-null  object
 1   primary_name   1017239 non-null  object
 2   movie_id       1017239 non-null  object
 3   category       1017239 non-null  object
 4   primary_title  1017239 non-null  object
 5   start_year     1017239 non-null  int64
 6   genres         1006126 non-null  object
 7   genres_list    1006126 non-null  object
dtypes: int64(1), object(7)
memory usage: 69.8+ MB
```

In [67]:
```
# removing any person with a frequency less than three

#gets a count of the frequency of each person's name in the df
film_people_value_counts = film_people_with_movies.primary_name

# selects the names that appear less than 3 times
remove_people = film_people_value_counts[film_people_value_coun

# filters out rows that have a primary_name that is in remove_p
film_people_mult_movies = film_people_with_movies[~film_people_
```

In [68]:
```
film_people_mult_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 448252 entries, 1 to 1017213
Data columns (total 8 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   person_id      448252 non-null  object
 1   primary_name   448252 non-null  object
 2   movie_id       448252 non-null  object
 3   category       448252 non-null  object
 4   primary_title  448252 non-null  object
 5   start_year     448252 non-null  int64
 6   genres         443892 non-null  object
 7   genres_list    443892 non-null  object
dtypes: int64(1), object(7)
memory usage: 30.8+ MB
```

In [69]:
```
# list of movies in the top 25% of roi
top_25_percent_roi = list(budgets_no_outliers[budgets_no_outlie
                                               > budgets_no_outl
                                                 by = ['roi (%)'
                         )
```

In [70]:
```
# filtering movie_basics_clean to only include movies in the to

top_roi_movie_basics = movie_basics_clean[movie_basics_clean['p
top_roi_movie_basics.head()
```

Out[70]:

	movie_id	primary_title	start_year	runtime_minutes	genres	ge
33	tt0293429	Mortal Kombat	2021	NaN	Action,Adventure,Fantasy	A
40	tt0326592	The Overnight	2010	88.0	None	
97	tt0431021	The Possession	2012	92.0	Horror,Mystery,Thriller	
115	tt0443272	Lincoln	2012	150.0	Biography,Drama,History	[B
125	tt0448115	Shazam!	2019	132.0	Action,Adventure,Comedy	A

In [71]:
```
top_roi_movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 798 entries, 33 to 145296
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         798 non-null    object 
 1   primary_title    798 non-null    object 
 2   start_year       798 non-null    int64  
 3   runtime_minutes  720 non-null    float64
 4   genres           792 non-null    object 
 5   genres_list      792 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 43.6+ KB
```

In [72]:
```
# creates df containing the people involved in the movies in th
top_roi_film_people = film_people_mult_movies[film_people_mult_
top_roi_film_people.head()
```

Out[72]:

	person_id	primary_name	movie_id	category	primary_title	start_year	
833	nm0192984	Paul Currie	tt2119532	producer	Hacksaw Ridge	2016	Biograp
834	nm0941777	Sam Worthington	tt2119532	actor	Hacksaw Ridge	2016	Biograp
835	nm0460795	Andrew Knight	tt2119532	writer	Hacksaw Ridge	2016	Biograp
837	nm0202704	Bruce Davey	tt2119532	producer	Hacksaw Ridge	2016	Biograp
838	nm0000154	Mel Gibson	tt2119532	director	Hacksaw Ridge	2016	Biograp

In [73]:
```
# removing any person with a frequency less than three

#gets a count of the frequency of each person's name in the df
top_people_value_counts = top_roi_film_people.primary_name.valu

# selects the names that appear less than 3 times
remove_ppl = top_people_value_counts[top_people_value_counts <

# filters out rows that have a primary_name that is in remove_p
top_roi_people = top_roi_film_people[~top_roi_film_people['prim
```

In [74]:
```
top_roi_people.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 978 entries, 841 to 733268
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   person_id      978 non-null    object
 1   primary_name   978 non-null    object
 2   movie_id       978 non-null    object
 3   category       978 non-null    object
 4   primary_title  978 non-null    object
 5   start_year     978 non-null    int64 
 6   genres         978 non-null    object
 7   genres_list    978 non-null    object
dtypes: int64(1), object(7)
memory usage: 68.8+ KB
```

In [75]:
```
# joining top_roi_film_people and budgets

top_people_budgets = pd.merge(top_roi_people,
                              budgets_no_outliers,
                              how = 'inner',
                              left_on = 'primary_title',
                              right_on = 'movie'
                             )
top_people_budgets.head()
```

Out[75]:

	person_id	primary_name	movie_id	category	primary_title	start_year
0	nm1954240	Teresa Palmer	tt2119532	actress	Hacksaw Ridge	2016
1	nm0001752	Steven Soderbergh	tt2268016	cinematographer	Magic Mike XXL	2015
2	nm1475594	Channing Tatum	tt2268016	actor	Magic Mike XXL	2015
3	nm1749221	Nina Jacobson	tt1650043	producer	Diary of a Wimpy Kid: Rodrick Rules	2011
4	nm0331516	Ryan Gosling	tt1120985	actor	Blue Valentine	2010

In [76]:
```
top_people_budgets.person_id.value_counts()
```

Out[76]:
```
nm0089658    34
nm0000881    10
nm0366389     8
nm1334526     8
nm0172830     8
             ..
nm0000108     3
nm0719637     3
nm1950086     3
nm0425053     2
nm2053216     1
Name: person_id, Length: 248, dtype: int64
```

In [77]:
```
top_people_budgets.category.value_counts()
```

Out[77]:
```
actor              299
producer           268
actress            202
writer             112
director            86
composer            49
cinematographer     11
self                 3
editor               2
Name: category, dtype: int64
```

In [78]:
```
# replacing any jobs listed as actress to actor
top_people_budgets['category'] = top_people_budgets.category.re
```

In [79]:
```
top_people_budgets['category'] = top_people_budgets.category.st
```

In [80]:
```
top_people_budgets.category.value_counts()
```

Out[80]:
```
Actor              501
Producer           268
Writer             112
Director            86
Composer            49
Cinematographer     11
Self                 3
Editor               2
Name: category, dtype: int64
```

In [81]:
```
top_people_budgets[top_people_budgets['category'] == 'Cinematog
```

Out[81]:

	person_id	primary_name	movie_id	category	primary_title	start_year
1	nm0001752	Steven Soderbergh	tt2268016	Cinematographer	Magic Mike XXL	2015
526	nm1227638	Mike Gioulakis	tt4972582	Cinematographer	Split	2016
529	nm0002947	Toby Oliver	tt5052448	Cinematographer	Get Out	2017
533	nm0002947	Toby Oliver	tt5308322	Cinematographer	Happy Death Day	2017
538	nm1227638	Mike Gioulakis	tt6857112	Cinematographer	Us	2019

In [82]:
```
# creates a list of the 3 least frequent categories
categories_to_drop = top_people_budgets.category.value_counts()
categories_to_drop
```

Out[82]:
```
Index(['Editor', 'Self', 'Cinematographer'], dtype='object')
```

In [83]:
```
# drops any rows where the category is in one of the 3 least fr
top_people_budgets = top_people_budgets[~top_people_budgets['ca
```

In [84]:
```
top_people_budgets.primary_name.value_counts()
```

Out[84]:
```
Jason Blum                 34
Michael Bay                10
Kristen Wiig                8
Dwayne Johnson              8
Anna Kendrick               8
                           ..
Bear McCreary               3
Scarlett Johansson          3
Sébastien K. Lemercier      3
Steven Soderbergh           2
Gregory Plotkin             1
Name: primary_name, Length: 244, dtype: int64
```

In [85]:
```
top_people_budgets[top_people_budgets['person_id'] == 'nm151293
```

Out[85]:

person_id	primary_name	movie_id	category	primary_title	start_year	genres	gen

In [86]:
```
# verifying that there are two different Paul Walkers and no du
top_people_budgets[top_people_budgets['primary_name'] == 'Paul
```

Out[86]:

person_id	primary_name	movie_id	category	primary_title	start_year	genres	gen

In [87]:
```
top_people_budgets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1016 entries, 0 to 1031
Data columns (total 14 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   person_id                           1016 non-null   object
 1   primary_name                        1016 non-null   object
 2   movie_id                            1016 non-null   object
 3   category                            1016 non-null   object
 4   primary_title                       1016 non-null   object
 5   start_year                          1016 non-null   int64
 6   genres                              1016 non-null   object
 7   genres_list                         1016 non-null   object
 8   release_date                        1016 non-null   object
 9   movie                               1016 non-null   object
 10  production_budget (millions of $)  1016 non-null   float6
4
 11  domestic_gross (millions of $)     1016 non-null   float6
4
 12  worldwide_gross (millions of $)    1016 non-null   float6
4
 13  roi (%)                            1016 non-null   float6
4
dtypes: float64(4), int64(1), object(9)
memory usage: 119.1+ KB
```

In [88]:
```
# returns the .info() for top_people_budgets, top_roi_movie_bas
dfs_for_analysis = [top_people_budgets, top_roi_movie_basics, b
get_info(dfs_for_analysis)
```

```
--------

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1016 entries, 0 to 1031
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   person_id                        1016 non-null   object
 1   primary_name                     1016 non-null   object
 2   movie_id                         1016 non-null   object
 3   category                         1016 non-null   object
 4   primary_title                    1016 non-null   object
 5   start_year                       1016 non-null   int64
 6   genres                           1016 non-null   object
 7   genres_list                      1016 non-null   object
 8   release_date                     1016 non-null   object
 9   movie                            1016 non-null   object
 10  production_budget (millions of $)  1016 non-null   float6
4
 11  domestic_gross (millions of $)     1016 non-null   float6
4
 12  worldwide_gross (millions of $)    1016 non-null   float6
4
 13  roi (%)                          1016 non-null   float6
4
dtypes: float64(4), int64(1), object(9)
memory usage: 119.1+ KB
None
--------

<class 'pandas.core.frame.DataFrame'>
Int64Index: 798 entries, 33 to 145296
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   movie_id         798 non-null    object
 1   primary_title    798 non-null    object
 2   start_year       798 non-null    int64
 3   runtime_minutes  720 non-null    float64
 4   genres           792 non-null    object
 5   genres_list      792 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 43.6+ KB
None
--------

<class 'pandas.core.frame.DataFrame'>
Int64Index: 5636 entries, 146 to 5781
Data columns (total 6 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   release_date                     5636 non-null   object
 1   movie                            5636 non-null   object
 2   production_budget (millions of $)  5636 non-null   float6
```

```
4
 3   domestic_gross (millions of $)     5636 non-null   float6
4
 4   worldwide_gross (millions of $)    5636 non-null   float6
4
 5   roi (%)                            5636 non-null   float6
4
dtypes: float64(4), object(2)
memory usage: 308.2+ KB
None
```

The output from the get_info() function shows that in each table, each column's python data type matches its true data type. In the `top_roi_movie_basics` DataFrame, the following three columns still have a few NaN values: `runtime_minutes` , `genres` , `genres_list` . Since this DataFrame, only has 785 entries, simply removing these rows could result in significant data loss. I leave these NaN values because I still have enough data in each of those rows for meaningful analysis, and I do not want to lose any important data by removing these rows.

4. Exploratory Data Analysis

The following are findings from this analysis:

- Of the 5,636 movies with functional budget data, **37%** did ***NOT*** achieve a positive ROI.

- The typical movie had an estimated **16 million dollar production budget**, generated an estimated **26 million dollars in worldwide gross revenue**, and produced an estimated **66% return on investment**.

- **Dramas** and **comedies** were the two **most common genres** for the movies in the top 25% of ROI.

- The middle 50% of the movies with the highest ROI had **runtimes between 87 and 113 minutes.**

- The three film **professions that generate the highest ROI** for dramas and comedies are: **composers**, **directors**, & **producers**.
  - The 5 highest grossing drama & comedy ***composers*** are: Danny Elfman, Alexandre Desplat, Marco Beltrami, Thomas Newman, Theodore Shapiro
  - The 3 highest grossing drama & comedy ***directors*** are: David O. Russell, Steven Spielberg, Damien Chazelle
  - The 3 highest grossing drama & comedy ***producers*** are: Simon Kinberg, Michael De Luca, Dana Brunetti

In [89]:
```
# getting summary statistics for the int cols of the budgets ta
round(budgets_no_outliers.describe(), 1)
```

Out[89]:

	production_budget (millions of $)	domestic_gross (millions of $)	worldwide_gross (millions of $)	roi (%)
count	5636.0	5636.0	5636.0	5636.0
mean	27.2	36.6	75.9	383.6
std	31.0	54.6	129.8	2990.8
min	0.0	0.0	0.0	-100.0
25%	5.0	1.2	3.7	-53.5
50%	16.0	16.0	26.1	66.4
75%	38.0	48.6	89.2	269.9
max	156.0	474.5	1341.7	179900.0

In [90]:
```
len(budgets_no_outliers)
```

Out[90]: 5636

In [91]:
```
# total percentage of films that did NOT have a positive ROI
round(len(budgets_no_outliers[budgets_no_outliers['roi (%)'] <=
```

Out[91]: 37.62

In [92]:
```
# getting the top genres for the movies in the top 25% roi

top_genres_count = top_roi_movie_basics['genres'].str.split(","
top_genres = list(top_genres_count.index)
top_2_genres = top_genres[:2]
```

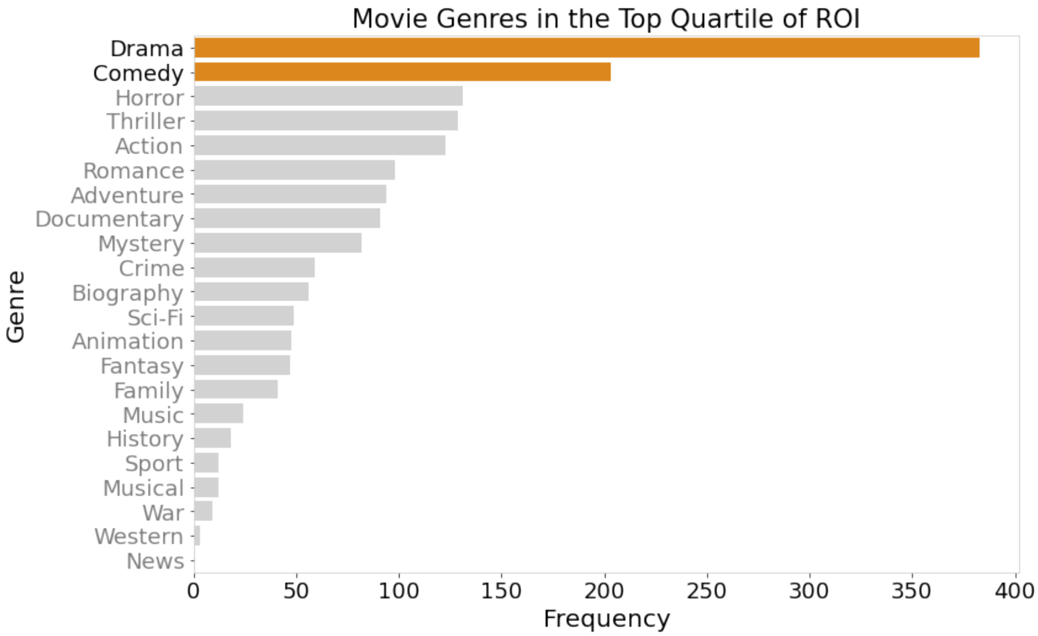
In [93]:
```
# calculates percentage of the two most common genres
top_genres_count_normalized = top_roi_movie_basics['genres'].st
round(top_genres_count_normalized[:2].sum()* 100, 1)
```

Out[93]: 34.2

This output represents the percentage of all movies that were classified as either Dramas and/or Comedies.

In [94]:
```
# setting colors for bars of barplot
top_genres_bar_colors = ['darkorange' if x in top_2_genres else

# setting colors for ticks of barplot
top_genres_tick_colors = ['black' if x in top_2_genres else 'gr
```

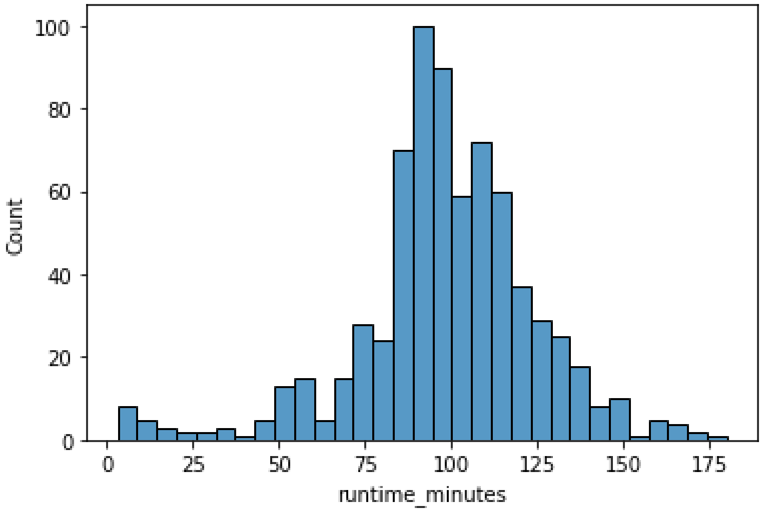
In [95]:
```
# horizontal bar plot showing the counts of genres for highest
fig, ax = plt.subplots(figsize = (12, 8))
sns.barplot(y = top_genres,
            x = top_genres_count,
            palette = top_genres_bar_colors
           );

ax.set_title('Movie Genres in the Top Quartile of ROI',
             fontsize = 21
            )

ax.set_xlabel('Frequency', fontsize = 20)
ax.set_ylabel("Genre",fontsize=20)
ax.tick_params(labelsize=18)
ax.spines['left'].set_color('lightgrey')
ax.spines['right'].set_color('lightgrey')
ax.spines['top'].set_color('lightgrey')
ax.spines['bottom'].set_color('lightgrey')

# this for loop sets the tick colors
for ticklabel, tickcolor in zip(plt.gca().get_yticklabels(), to
    ticklabel.set_color(tickcolor)

plt.savefig("top_genres.png", bbox_inches = 'tight');
```



This bar chart visualizes the the frequency genres of movies that were in the upper quartile of ROI. This visualization draws our attention to the two most common genres: **Drama** and **Comedy**. Specifically, these two genres made up more than one-third (**34.2%**) of all genres for the movies with the highest ROI.

In [96]:
```
# creating a list of the top 2 genres of movies in top 25% roi
top2_genres = list(round(top_roi_movie_basics['genres'].explode
```

In [97]:
```
# visualizes distribution of movie runtime minutes
sns.histplot(top_roi_movie_basics['runtime_minutes']);
```



This histogram shows the distribution of the runtimes for the movies with the highest ROI. This data appears to be approximately normal, but could also be considered left skewed because its left tail is relatively long.

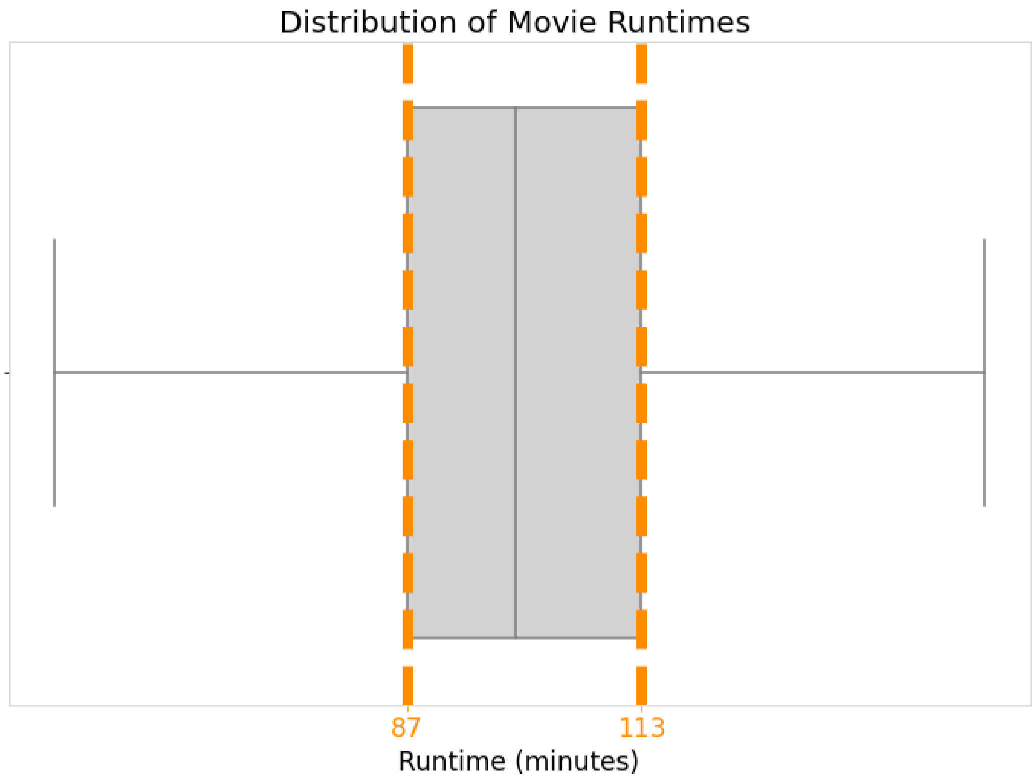
In [98]:
```
# getting summary statistics for top_roi_movie_basics runtimes
round(top_roi_movie_basics['runtime_minutes'].describe(), 1)
```

Out[98]:
```
count    720.0
mean      98.4
std       26.5
min        3.0
25%       87.0
50%       99.0
75%      113.0
max      180.0
Name: runtime_minutes, dtype: float64
```

This output provide summary statistics for movie runtimes. The 1st Quartile (25%) and 3rd Quartile (75%) are especially important because 50% of the data lies between these two values. I will be highlighting this in the boxplot used to visualize this distribution below.

In [99]:
```
# boxplot for the runtime of the movies in top 25% of roi
fig, ax = plt.subplots(figsize= (12, 8))
sns.boxplot(top_roi_movie_basics['runtime_minutes'],
            showfliers = False,
            color = 'lightgrey'
           )

# creates dashed lines to emphasis Q1 and Q3 on the graph
plt.axvline(x = 87,
            color = 'darkorange',
            label = 'axvline - full height',
            linestyle = '--',
            linewidth=7.0
           )

plt.axvline(x = 113,
            color = 'darkorange',
            label = 'axvline - full height',
            linestyle = '--',
            linewidth=7.0
           )

# formatting title and axes
ax.set_title("Distribution of Movie Runtimes ",
             fontsize = 20
            )

ax.set_xlabel('Runtime (minutes)',
              fontsize = 17
             )

ax.set_xticks([87, 113])
ax.tick_params(axis='x',
               colors='darkorange',
               labelsize = 17
              )

# makes border of figure grey
ax.spines['left'].set_color('lightgrey')
ax.spines['right'].set_color('lightgrey')
ax.spines['top'].set_color('lightgrey')
ax.spines['bottom'].set_color('lightgrey')

plt.savefig("runtimes_boxplot.png", bbox_inches = 'tight');
```

```
/Users/chriskucewicz/anaconda3/envs/learn-env/lib/python3.8/si
te-packages/seaborn/_decorators.py:36: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an err
or or misinterpretation.
  warnings.warn(
```



This boxplot visualizes the distribution of runtimes of movies that were in the upper quartile of ROI. This visualization draws our attention to the runtime for the middle 50% of movies. Specifically, the middle 50% of movies had **runtimes between 87 and 113 minutes**.

In [100]:
```
# creates a table of the median ROI for each film profession
roi_categories = top_people_budgets.groupby(['category']).media

# sorts roi_categories from greatest ROI to least
roi_categories_sorted = roi_categories.sort_values('roi (%)', a
roi_categories_sorted
```

Out[100]:

category	roi (%)	production_budget (millions of $)	worldwide_gross (millions of $)
Composer	724.7	18.0	115.5920
Director	685.7	23.0	172.0590
Producer	685.3	15.0	129.0865
Writer	613.3	41.0	207.0400
Actor	489.6	30.0	175.3620

In [101]:
```
# saves the list of the top three professions by ROI %
top_3_roi_categories = roi_categories_sorted.head(3)
top_3_roi_categories
```

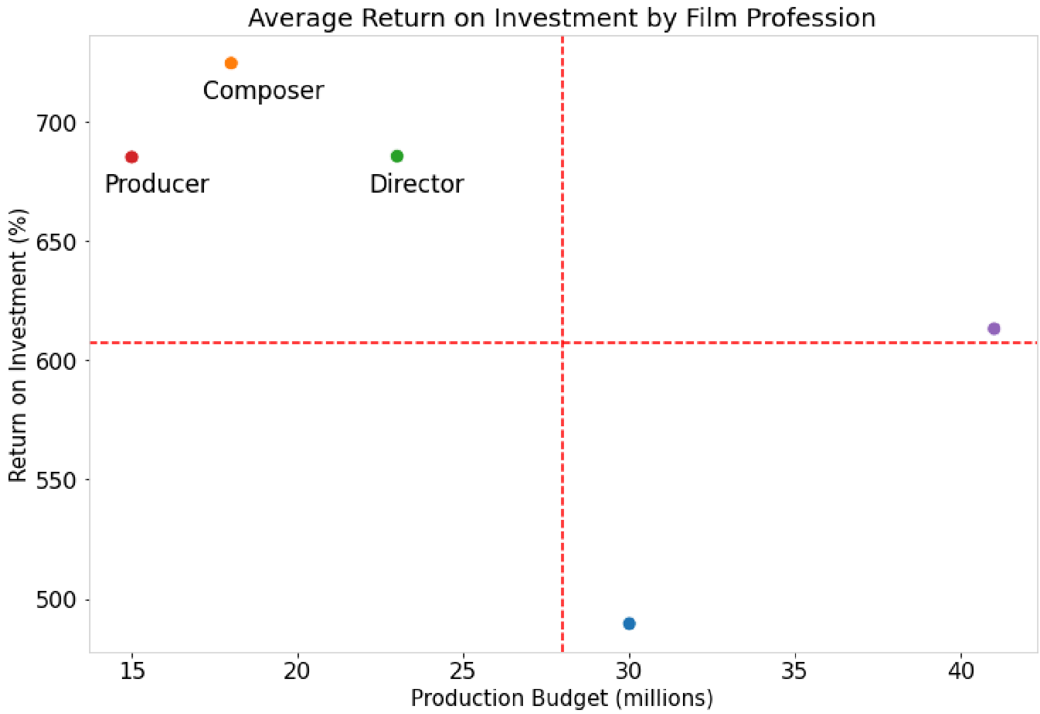
Out[101]:

	roi (%)	production_budget (millions of $)	worldwide_gross (millions of $)
category			
Composer	724.7	18.0	115.5920
Director	685.7	23.0	172.0590
Producer	685.3	15.0	129.0865

The above output is a table that shows the top 3 highest average return on investment (ROI) grouped by film profession. For example, composers had an average ROI of 725% and producers had an average ROI of 685%.

In [102]:
```
# calculates average ROI across all film professions which will
roi_h_threshold = 0.5 * (roi_categories_sorted['roi (%)'].max()
                  + roi_categories_sorted['roi (%)'].min())

# creates vertical threshold
roi_v_threshold = 0.5 * (roi_categories_sorted['production_budg
                  + roi_categories_sorted['production_budget (mi

fig, ax = plt.subplots(figsize = (12,8))

# creates and plots scatterplot
sns.scatterplot(data = top_people_budgets.groupby(['category'])
                x = 'production_budget (millions of $)',
                y = 'roi (%)',
                s = 100,
                hue='category',
                )


# plots name for top 3 professions near corresponding data poin
for line in range(0, top_3_roi_categories.shape[0]):
     ax.annotate(top_3_roi_categories.index[line],
                 (top_3_roi_categories['production_budget (mi
                  top_3_roi_categories['roi (%)'].iloc[line])
                 fontsize=17,
                 xytext=(-20, -25),
                 textcoords='offset points'
                )

# removes legend
ax.get_legend().remove()

# draws horizontal line for median roi
ax.axhline(y = roi_h_threshold,
            color = 'red',
            label = 'axvline - full height',
            linestyle = '--'
          )
# draws vertical line for average median budget
ax.axvline(x = roi_v_threshold,
            color = 'red',
            label = 'axvline - full height',
            linestyle = '--'
          )
# formatting title and axes
ax.set_title("Average Return on Investment by Film Profession",
             fontsize = 18
            )

ax.set_xlabel('Production Budget (millions)',
              fontsize = 15
             )

ax.set_ylabel('Return on Investment (%)',
```

```
              fontsize=15
             )

ax.tick_params(labelsize=16)

#ax.set_xticklabels(x.astype(int))

# makes border of figure grey
ax.spines['left'].set_color('lightgrey')
ax.spines['right'].set_color('lightgrey')
ax.spines['top'].set_color('lightgrey')
ax.spines['bottom'].set_color('lightgrey')

plt.savefig("roi_by_profession.png", bbox_inches = 'tight');
```



This scatterplot graphs the relationship between production budget and ROI based on film profession. The top left quadrant represents those professions that are involved in movies with low production budgets, but have a high return on investment. The three professions in this quadrant include composers, directors, and producers.

In [103]:
```
# function that will print a list of the top 5 highest grossing
def top_grossing(categories):

    for category in categories:

        # stores a dataframe of those in the corresponding cate
        top_category = top_people_budgets[(top_people_budgets['
                                          & (top_people_budgets

                                            )
                                         ]


        # creates a table of the top 15 highest grossing profes
        top_category_sum_gross = top_category.groupby(['primary

        # selects the top 5 names from the list
        top_5_category = list(top_category_sum_gross.index)[:5]

        print(f"The top 5 highest grossing {category}s are:" +
```

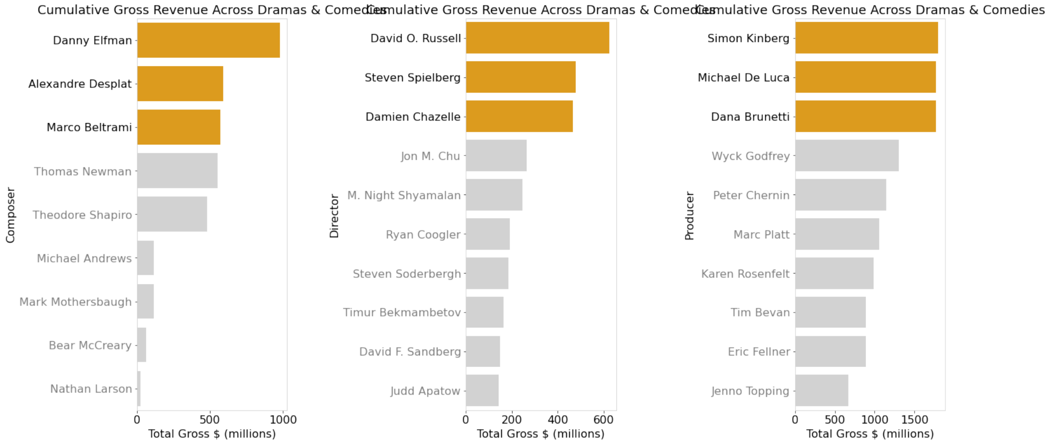
In [104]:
```
top_grossing(top_3_roi_categories.index)
```

```
The top 5 highest grossing Composers are:['Danny Elfman', 'Ale
xandre Desplat', 'Marco Beltrami', 'Thomas Newman', 'Theodore
Shapiro']
The top 5 highest grossing Directors are:['David O. Russell',
'Steven Spielberg', 'Damien Chazelle', 'Jon M. Chu', 'M. Night
Shyamalan']
The top 5 highest grossing Producers are:['Simon Kinberg', 'Mi
chael De Luca', 'Dana Brunetti', 'Wyck Godfrey', 'Peter Cherni
n']
```

In [105]:
```
def plot_top_grossing(categories):

        fig, ax = plt.subplots(nrows = 1, ncols = len(categorie

        # creating an indexed list of categories and looping th
        for index, category in enumerate(categories):

            # stores a dataframe of those in the corresponding
            top_category = top_people_budgets[(top_people_budge
                                             & (top_people_bud

            # creates a table of the top 10 highest grossing pr
            top_category_sum_gross = top_category.groupby(['pri

            # selects the top 3 names from top_category_sum_gro
            top_3_category = list(top_category_sum_gross.index)

            # setting colors for bars of barplot
            top_sum_category_colors = ['orange' if x in top_3_c

            # setting colors for ticks of barplot
            top_sum_categories_tick_colors = ['black' if x in t

            # creates barplot of top 10 highest grossing withir
            sns.barplot(
                y = top_category_sum_gross.index,
                x = top_category_sum_gross['worldwide_gross (mi
                palette = top_sum_category_colors,
                ax = ax[index]
            )

            # formatting title and axes
            ax[index].set_title(f"Cumulative Gross Revenue Acro
                                fontsize = 18
                               )

            ax[index].set_xlabel('Total Gross $ (millions)',
                                 fontsize = 16
                                )

            ax[index].set_ylabel(f'{category}',
                                 fontsize=16
                                )
            ax[index].tick_params(labelsize=16)
            ax[index].set_yticklabels(labels = top_category_sum

            # for loop adjusts the color of the tick parameters
            for ticklabel, tickcolor in zip(ax[index].get_ytick
                ticklabel.set_color(tickcolor)

            # makes border of figure grey
            ax[index].spines['left'].set_color('lightgrey')
            ax[index].spines['right'].set_color('lightgrey')
            ax[index].spines['top'].set_color('lightgrey')
```

```
            ax[index].spines['bottom'].set_color('lightgrey');

        fig.tight_layout(pad=5.0);
```

In [106]:
```
plot_top_grossing(top_3_roi_categories.index)
plt.savefig("total_gross_top3_professions.png", bbox_inches = '
```

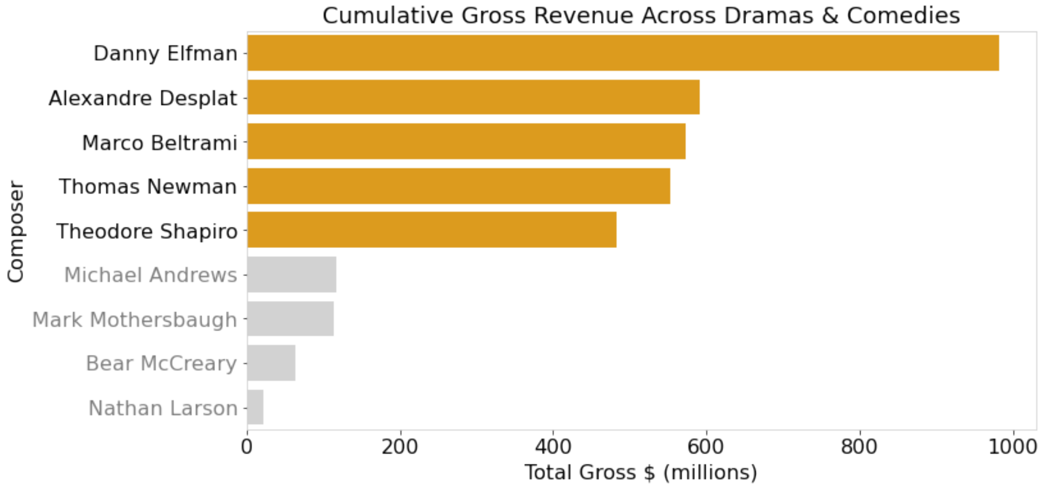


The above output shows the top 10 highest cumulative grossing composers, directors, and producers who create dramas and comedies. More specifically, each barplot highlights the top 3 in each category.

In [107]:
```
# this is the same function as above except it does not involve

def plot_individ_top_grossing(category, number):

        fig, ax = plt.subplots(figsize = (12,6))

        # stores a dataframe of those in the corresponding cate
        top_category = top_people_budgets[(top_people_budgets['
                                                    & (top_people_bud

        # creates a table of the top 10 highest grossing profes
        top_category_sum_gross = top_category.groupby(['primary


        # selects the top 3 names from top_category_sum_gross
        top_3_category = list(top_category_sum_gross.index)[:nu

        # setting colors for bars of barplot
        top_sum_category_colors = ['orange' if x in top_3_categ

        # setting colors for ticks of barplot
        top_sum_categories_tick_colors = ['black' if x in top_3

        # creates barplot of top 10 highest grossing within res
        sns.barplot(
            y = top_category_sum_gross.index,
            x = top_category_sum_gross['worldwide_gross (millio
            palette = top_sum_category_colors,
            )

        # formatting title and axes
        ax.set_title(f"Cumulative Gross Revenue Across Dramas &
                                fontsize = 18
                               )

        ax.set_xlabel('Total Gross $ (millions)',
                                 fontsize = 16
                                )

        ax.set_ylabel(f'{category}',
                                 fontsize=16
                                )
        ax.tick_params(labelsize=16)
        ax.set_yticklabels(labels = top_category_sum_gross.inde

        # for loop adjusts the color of the tick parameters to
        for ticklabel, tickcolor in zip(ax.get_yticklabels(), t
                ticklabel.set_color(tickcolor)

        # makes border of figure grey
        ax.spines['left'].set_color('lightgrey')
        ax.spines['right'].set_color('lightgrey')
        ax.spines['top'].set_color('lightgrey')
        ax.spines['bottom'].set_color('lightgrey');
```

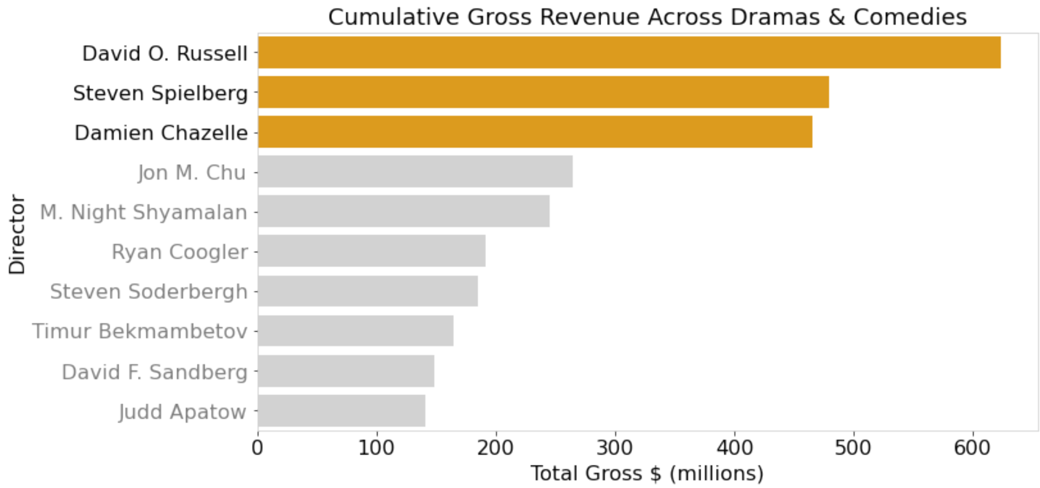
```
        fig.tight_layout(pad=3.0);
```

In [108]:
```
plot_individ_top_grossing(top_3_roi_categories.index[0], 5);
plt.savefig("total_gross_composers.png", bbox_inches = 'tight')
```



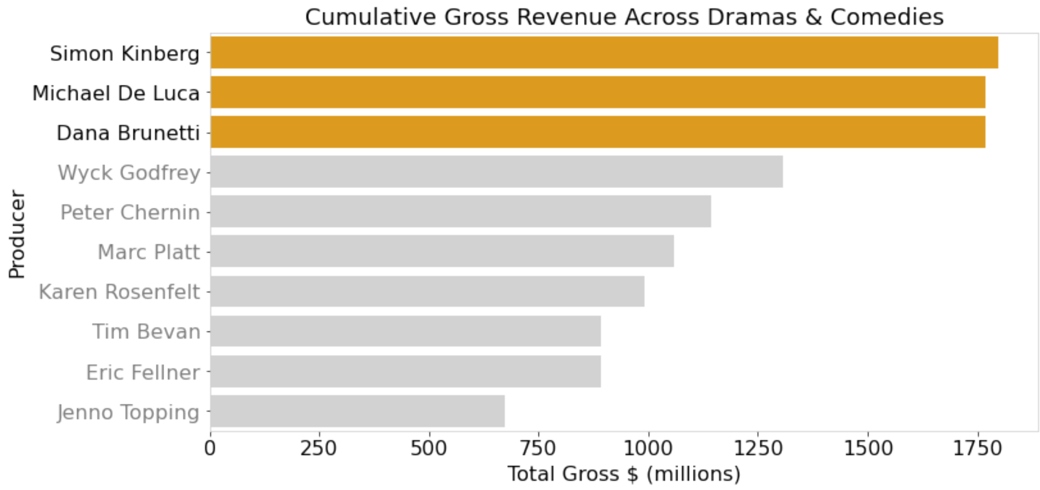
The above output shows the top 10 highest cumulative grossing composers who create dramas and comedies. More specifically, this barplot highlights the top 5 highest cumulative grossing composers: Danny Elfman, Alexandre Desplat, Marco Beltrami, Thomas Newman, Theodore Shapiro.

In [109]:
```
plot_individ_top_grossing(top_3_roi_categories.index[1], 3)
plt.savefig("total_gross_directors.png", bbox_inches = 'tight')
```



The above output shows the top 10 highest cumulative grossing directors who create dramas and comedies. More specifically, this barplot highlights the top 3 highest cumulative grossing directors: David O. Russell, Steven Spielberg, Damien Chazelle.

In [110]:
```
plot_individ_top_grossing(top_3_roi_categories.index[2], 3)
plt.savefig("total_gross_producers.png", bbox_inches = 'tight')
```



The above output shows the top 10 highest cumulative grossing producers who create dramas and comedies. More specifically, this barplot highlights the top 3 highest cumulative grossing producers: Simon Kinberg, Michael De Luca, Dana Brunetti.

5. Conclusions

Limitations

While the datasets and tables provided a variety of data, there was a notable limitation in the availability of budget information for movies. This led to a significant discrepancy between the number of entries in the budgets table (~5,000) and other tables within the IMDB database, one of which contained over 1 million entries. Consequently, the analysis was restricted by the limited amount of budget data, reducing the number of movies that could be analyzed. A more comprehensive dataset that includes budget information for a wider range of movies would enable a more thorough analysis and yield more informed recommendations about the factors that influence a movie's return on investment.

Recommendations

This analysis leads to three recommendations for movie creation:

1. Focus on creating movies within the ***drama or comedy genres***.

   - Over **one-third** of the movies with the highest ROI were classified as dramas and/or comedies.

2. Create movies with ***runtimes between 87 and 113 minutes***.

- Half of all movies with the highest ROI had runtimes between 87 and 113 minutes.

3. Focus on hiring high-quality ***composers, directors, and producers*** who specialize in comedy & drama, as these three film professions had the highest ROI.

- **Recommended drama & comedy composers** (top five highest cumulative grossing): Danny Elfman, Alexandre Desplat, Marco Beltrami, Thomas Newman, Theodore Shapiro

- **Recommended drama & comedy directors** (top three highest cumulative grossing): David O. Russell, Steven Spielberg, Damien Chazelle

- **Recommended drama & comedy producers** (top three highest cumulative grossing): Simon Kinberg, Michael De Luca, Dana Brunetti

Next Steps

With these recommendations in mind, I am interested the following next steps:

- gathering more budget data on a wider number of movies
- performing regression analysis to answer the question: Which factors most strongly correlate with a movie's ROI?

6. Resources

- During the data preparation phase, I ran into the problem of a dataframe column containing a string of multiple film genres (i.e. `comedy, drama, horror`). I needed to find a way to separate the single string into different strings for each genre. I googled 'pandas column contains list of genres' and found two different solutions. On saturncloud.io (https://saturncloud.io/blog/how-to-split-one-column-into-multiple-columns-in-pandas-dataframe/#:~:text=Using%20the%20pd.&text=Series.-,str.,list%20as%20a%20 the following code was helpful:

```
df[['First Name', 'Last Name']] = df['Name'].str.split(' ', expand=True)
```

I also found a helpful solution on reddit (https://www.reddit.com/r/learnpython/comments/krasnw/how_to_put_my_genre written by pytrashpandas which used the following code:

```
genre_count = df['genre'].str.split(',').explode().value_counts()
```

The combination of these two solutions helped me to separate each string in the pandas dataframe column into different genres based on the comma (',') delimiter

- When inspecting the tables within the the given database, I was looking for a way to return the name of each table with its corresponding number of rows. I googled 'loop to create dataframe using read_sql' and found a solution on [stackoverflow (https://stackoverflow.com/questions/71432838/for-loop-to-create-a-dataframe-using-pandas-read-sql-in-python)](https://stackoverflow.com/questions/71432838/for-loop-to-create-a-dataframe-using-pandas-read-sql-in-python) which contained the following code:

  ```
  for table_name in tables.keys(): sqlite_table = f"SELECT * FROM {table_name} WHERE symbol='{company}'"
  tables[table_name] = pd.read_sql(sqlite_table, database)
  ```

  This solution helped me create a sql query using read_sql to return the number of rows in each table of the database.

- After joining dataframes, I was trying to figure out how to return duplicated rows in a dataframe so I could see if a `person_id` and/or `movie_id` was duplicated in the dataframe. I googled 'subset dataframe pandas where count for value is more than 1' and found a solution on [stackoverflow (https://stackoverflow.com/questions/48628417/how-to-select-rows-in-pandas-dataframe-where-value-appears-more-than-once)](https://stackoverflow.com/questions/48628417/how-to-select-rows-in-pandas-dataframe-where-value-appears-more-than-once) written by cs95 which contained the following code:

  ```
  v = df.Parameter.value_counts()
  df[df.Parameter.isin(v.index[v.gt(5)])]
  ```

  This solution helped me return rows that included matching `movie_id` and `person_id` .

- During the EDA phase, I was looking for a way to automate creating a series of bar graphs, so I tried creating a function using a for loop. I wasn't sure how to get the graph to appear on a specific axes, so I googled 'barplots using for loop matplotlib' and found a solution on [stackoverflow (https://stackoverflow.com/questions/43962735/creating-barplots-using-for-loop-using-pandas-matplotlib)](https://stackoverflow.com/questions/43962735/creating-barplots-using-for-loop-using-pandas-matplotlib) written by Robbie which contained the following code:

  ```
  for i, zone in enumerate(zones):

  data.loc[data.zone==zone].hist(column='OS Usage' ,

  bins=np.linspace(0,1,10) ,

  ax=axes[i],
  ```

`sharey=True)`

`axes[i].set_title('OS Usage in {0}'.format(zone))`

`axes[i].set_xlabel('Value')`

`axes[i].set_ylabel('Count')`

This solution helped me to be able to create three side-by-side horizontal bar plots with each bar plot on a different axes.

- During the EDA phase, I wanted a to create a count of all the different film professions. The issue was that some entries has multiple film professions saved as a single string (i.e. `writer, director, producer`]. I needed a strategy to separate and count each individual profession so I googled 'pandas column contains any values in list' and found a solution on [stackoverflow (https://stackoverflow.com/questions/50355825/pandas-using-isin-to-return-if-column-contains-any-values-in-a-list-rather-th)](https://stackoverflow.com/questions/50355825/pandas-using-isin-to-return-if-column-contains-any-values-in-a-list-rather-th) written by piRSquared which contained the code:

  `df['Description'].str.split(expand = True).isin(keywords_list).any(1)`

  This solution helped me to create a count of each profession and then search the original dataframe for everyone who matched a certain profession.

- When I was creating a visualization for the return on investment for each film profession, I was looking for a way to label the individual points on the scatter plot rather than relying on the key. I googled 'labeling points on a scatterplot seaborn' and found a solution on [stackoverflow (https://stackoverflow.com/questions/46027653/adding-labels-in-x-y-scatter-plot-with-seaborn)](https://stackoverflow.com/questions/46027653/adding-labels-in-x-y-scatter-plot-with-seaborn) written by Scott Boston and edited by Trenton McKinney which contained the following code:

  `def label_point(x, y, val, ax):`

  `a = pd.concat({'x': x, 'y': y, 'val': val}, axis=1)`

  `for i, point in a.iterrows():`

  `ax.text(point['x']+.02, point['y'], str(point['val']))`

  This solution helped me to label specific points on my scatterplot and align the label positioning.

- When I was creating visualizations for to show the top genres or the top grossing composers or directors, I wanted to make sure the top categories really stood out to my audience, so I wanted a way to adjust the tick labels to make the names of the top categories stand out while making the remaining categories less prominent. I googled '' and found a solution on [stackoverflow (https://stackoverflow.com/questions/39409530/every-tick-label-in-different-color)](https://stackoverflow.com/questions/39409530/every-tick-label-in-different-color) written by tmdavison which contained the code:

```
 for ticklabel, tickcolor in
zip(plt.gca().get_xticklabels(), my_colors):
   ticklabel.set_color(tickcolor)
```

This solution helped me to make the ticks I wanted my audience to focus on a darker shade than the other ticks.