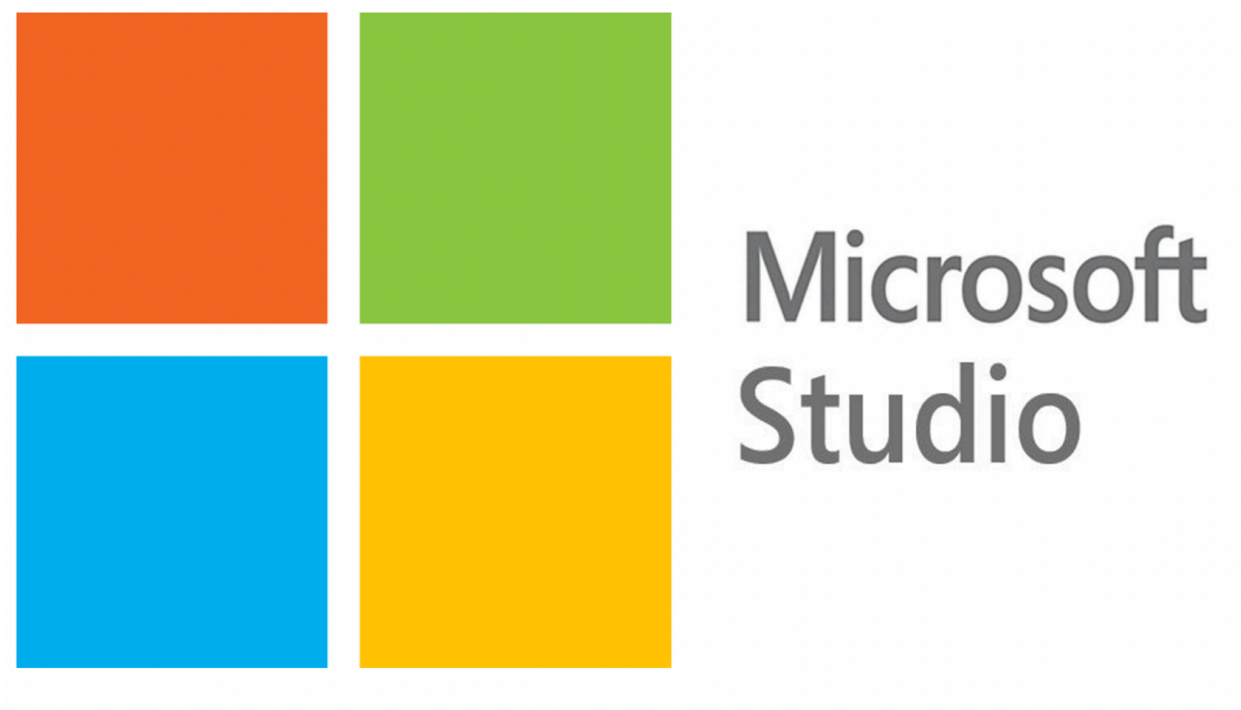


Microsoft Movie Studio Recommendations

Overview

This project uses data analysis to understand the performance of movie productions through various performance metrics in an effort to inform Microsoft executives about the most financially viable opportunities to enter the movie industry.

Business Understanding

In 2019, prior to the pandemic, movie box office sales were at $11B, and in that same year there were over 1 billion tickets sold to the various films released at the time. The industry is recovering as we grow in our understanding of how to navigate the current environment with the coronavirus still prevalent. This year, box office sales are projected to be in excess of $6B with almost 750M tickets sold. (The numbers - https://www.the-numbers.com/market/ (https://www.the-numbers.com/market/))

The leadership of Microsoft is eager to learn about the potential to enter the movie industry with financial success in the near term. This analysis works to provide insight into important metrics of financial and public interest performance within the movie industry to better inform the Microsoft executive team as they determine their clear path forward to be a new entrant into the well established movie industry.

Data Understanding

The information necessary to complete this detailed data analysis is gathered by various respected industry organizations. The datasets available for this analysis were sourced from the resources listed below:

- Box Office Mojo (https://www.boxofficemojo.com)
  - This dataset includes information about movie financial performance with gross domestic and foreign box office sales. The dataset includes information for 3,387 different titles.
- Rotten Tomatoes (https://www.rottentomatoes.com)
  - Movie info dataset: This dataset includes information regarding a synopsis, runtime, the director and writer team, release date (Theater and DVD), the production studio, genre, and the rating of the film. Altogether, the dataset includes information for 1,560 different titles.
  - Reviews dataset: This dataset includes information for movie reviews and ratings, along with the critic who is reviewing. In total, this data set provides information for 54,432 entries.
- TheMovieDB (https://www.themoviedb.org)
  - This dataset includes information about the genre, language, popularity, and release date of the various movies represented. The dataset includes 26,517 entries.
- The Numbers (https://www.the-numbers.com)
  - This dataset includes information about the release date, production budget, and the domestic and international gross sales. The dataset includes 5,782 entries.
- IMDB (https://www.imdb.com)
  - Principals: This dataset relationally connects to the other datasets from IMDB through movie ID, and person ID. This data set also include information particular to a persons role (e.g. if an actress is listed the category column shares the name of their character). This dataset has 1,028,186 entries
  - Known for: This data set provides a list of each person available from the principals dataset and connects them to a movie that they are known for. In total this dataset has 1,638,259 entries.
  - Directors | Writers: These data sets house information for the writers and directors of films and what films they are associated with. Directors has 291,174 entries and writers has 255,873 entries.
  - Persons: This dataset includes personal identifiable information such a persons birth and death year along with their profession. This dataset includes 606,648 entries.
  - Movie Basics: This dataset includes information to identify a movie's foundational details such as the year it was released, the runtime and the genre of the film. This dataset has 146,144 entries.
  - Movie Ratings: This dataset includes information about movie rating and the number of votes received by reviewers. This dataset includes 73,856 entries.
  - Movie Akas: This dataset includes information about the movie region, language, type and attribute. This dataset includes 331,703 entries.

Below, we will further explore the information available from the datasets that will be used to better understand the critical metrics of performance that will inform the Microsoft executive teams next steps.

Loading Data

In [1]:
```
# import the necessary support packages to manipulate the available in
import pandas as pd
import numpy as np
```

executed in 202ms, finished 13:45:43 2022-06-16

Box Office Mojo

In [2]:
```
# load the box office mojo csv file as a pandas dataframe
box_office_mojo = pd.read_csv('zippedData/bom.movie_gross.csv.gz')
```

executed in 23ms, finished 13:45:43 2022-06-16

In [3]:
```
# look at the information within the box office mojo dataset
box_office_mojo.info()
```

executed in 18ms, finished 13:45:44 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

We now understand that the Box Office Mojo dataset has 5 columns of information that we have available to us to use for the analysis project. It is important to note that there are 3 data types represented in this dataset. Most importantly as we approach the data cleaning steps of this analysis we will want to critically look at the null values throughout the dataset. We can see that the title and year columns are fully represented with 3,387 entries, though the foreign_gross, domestic_gross, and studio columns have entries that are empty. Once we are to the data cleaning stage looking into the missing data will allow us to better understand how to address the empty cells.

In [4]:
```
# look at the first 5 rows of the box office mojo dataset
box_office_mojo.head()
```

executed in 24ms, finished 13:45:44 2022-06-16

Out[4]:

	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

Now we can begin to see the data that is available to purposefully assess the business problem. Now we will duplicate this process for the 4 additional datasets below

Rotten Tomatoes

In [5]:
```
# load the rotten tomatoes movie info csv file as a pandas dataframe
RT_movie_info = pd.read_csv('zippedData/rt.movie_info.tsv.gz', sep='\t
```

executed in 32ms, finished 13:45:45 2022-06-16

In [6]:
```
# look at the information within the rotten tomatoes movie info datase
RT_movie_info.info()
```

executed in 6ms, finished 13:45:45 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

In [7]:
```
# look at the first 5 rows of the rotten tomatoes movie info dataset
RT_movie_info.head()
```

executed in 8ms, finished 13:45:45 2022-06-16

Out[7]:

	id	synopsis	rating	genre	director	writer	theater_da
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9,
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17,
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13,
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9,
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	

In [8]:
```
# load the rotten tomatoes reviews csv file as a pandas dataframe
RT_reviews = pd.read_csv('zippedData/rt.reviews.tsv.gz', sep='\t', enc
```

executed in 181ms, finished 13:45:46 2022-06-16

In [9]:
```
# look at the information within the rotten tomatoes reviews dataset
RT_reviews.info()
```

executed in 15ms, finished 13:45:46 2022-06-16

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

In [10]:
```
# look at the first 5 rows of the rotten tomatoes reviews dataset
RT_reviews.head()
```

executed in 6ms, finished 13:45:46 2022-06-16

Out[10]:

	id	review	rating	fresh	critic	top_critic	publisher	date
0	3	A distinctly gallows take on contemporary fina...	3/5	fresh	PJ Nabarro	0	Patrick Nabarro	November 10, 2018
1	3	It's an allegory in search of a meaning that n...	NaN	rotten	Annalee Newitz	0	io9.com	May 23, 2018
2	3	... life lived in a bubble in financial dealin...	NaN	fresh	Sean Axmaker	0	Stream on Demand	January 4, 2018
3	3	Continuing along a line introduced in last yea...	NaN	fresh	Daniel Kasman	0	MUBI	November 16, 2017
4	3	... a perverse twist on neorealism...	NaN	fresh	NaN	0	Cinema Scope	October 12, 2017

The Movie Database

In [11]:

```
# load the the movie database csv file as a pandas dataframe
movie_database = pd.read_csv('zippedData/tmdb.movies.csv.gz', index_co
```

executed in 63ms, finished 13:45:47 2022-06-16

In [12]:

```
# look at the information within the movie database dataset
movie_database.info()
```

executed in 9ms, finished 13:45:47 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 26517 entries, 0 to 26516
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   genre_ids          26517 non-null  object 
 1   id                 26517 non-null  int64  
 2   original_language  26517 non-null  object 
 3   original_title     26517 non-null  object 
 4   popularity         26517 non-null  float64
 5   release_date       26517 non-null  object 
 6   title              26517 non-null  object 
 7   vote_average       26517 non-null  float64
 8   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(2), object(5)
memory usage: 2.0+ MB
```

In [13]:
```
# look at the first 5 rows of the movie database dataset
movie_database.head()
```

executed in 7ms, finished 13:45:47 2022-06-16

Out[13]:

	genre_ids	id	original_language	original_title	popularity	release_date	title
0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Ha Po and Deat Hallo Pa
1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	How Tr Y Drag
2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron M
3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	St
4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Incept

The Numbers

In [14]:
```
# load the the numbers movie budget csv file as a pandas dataframe
TN_movie_budget = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
```

executed in 13ms, finished 13:45:48 2022-06-16

In [15]:
```
# look at the information within the numbers movie budget dataset
TN_movie_budget.info()
```

executed in 7ms, finished 13:45:48 2022-06-16

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

In [16]:
```
# look at the first 5 rows of the numbers movie budget dataset
TN_movie_budget.head()
```

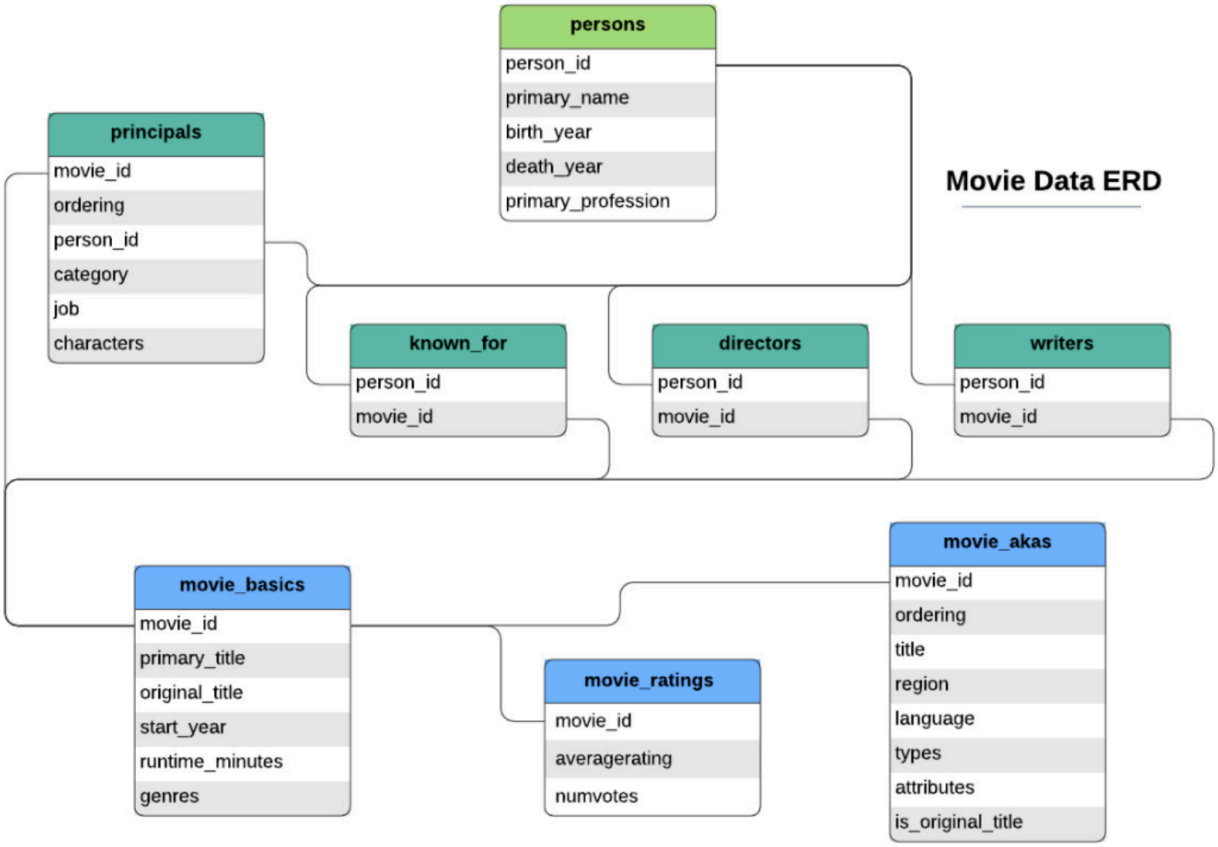
executed in 6ms, finished 13:45:49 2022-06-16

Out[16]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,27
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,87
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,35
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,96
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,74

IMDB

The IMDB database is a dataframe that we will query through SQL to look into the layout of the data further. The data available through this database is summarized below in the Movie Data ERD image.



Below, we will further explore the information available from the IMDB datasets that will be used to better understand the critical metrics of performance that will inform the Microsoft executive teams next steps. We will only further explore the datasets that are anticipated to be used in the scope of this analysis.

In [17]:
```
# import the necessary support packages to manipulate the available in
import sqlite3

# create a connection through SQLite to the stored dataset
conn = sqlite3.connect('zippedData/im.db')
```

executed in 6ms, finished 13:45:49 2022-06-16

In [18]:
```
# look at the first 5 rows of the movie basics dataset
q1 = """SELECT *
          FROM movie_basics;
"""
IMDB_movie_basics = pd.read_sql(q1, conn)
IMDB_movie_basics.head()
```

executed in 509ms, finished 13:45:50 2022-06-16

Out[18]:

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

In [19]:
```
# look at the information within the movie basics dataset
IMDB_movie_basics.info()
```

executed in 30ms, finished 13:45:50 2022-06-16

```
<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
```

In [20]:
```
# look at the first 5 rows of the movie ratings dataset
q2 = """SELECT *
          FROM movie_ratings;
"""
IMDB_movie_ratings = pd.read_sql(q2, conn)
IMDB_movie_ratings.head()
```

executed in 116ms, finished 13:45:50 2022-06-16

Out[20]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In [21]:
```
# look at the information within the movie rating dataset
IMDB_movie_ratings.info()
```

executed in 22ms, finished 13:45:51 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   movie_id       73856 non-null  object
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
```

In [22]:
```
# look at the first 5 rows of the persons dataset
q3 = """SELECT *
          FROM persons;
"""
IMDB_persons = pd.read_sql(q3, conn)
IMDB_persons.head()
```

executed in 1.28s, finished 13:45:52 2022-06-16

Out[22]:

	person_id	primary_name	birth_year	death_year	primar
0	nm0061671	Mary Ellen Bauder	NaN	NaN	miscellaneous,production_m
1	nm0061865	Joseph Bauer	NaN	NaN	composer,music_department,so
2	nm0062070	Bruce Baum	NaN	NaN	miscellan
3	nm0062195	Axel Baumann	NaN	NaN	camera_department,cinematographe
4	nm0062798	Pete Baxter	NaN	NaN	production_designer,art_departme

In [23]:
```
# look at the information within the persons dataset
IMDB_persons.info()
```

executed in 81ms, finished 13:45:52 2022-06-16

```
<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 [24]:
```
# look at the first 5 rows of the principals dataset
q4 = """SELECT *
          FROM principals;
"""
IMDB_principals = pd.read_sql(q4, conn)
IMDB_principals.head()
```

executed in 2.56s, finished 13:45:55 2022-06-16

Out[24]:

	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"]

In [25]:
```
# look at the information within the writers dataset
IMDB_principals.info()
```

executed in 190ms, finished 13:45:55 2022-06-16

```
<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
```

Data Cleaning & Feature Engineering

It is important to ensure that the data used in this and any analysis is thoughtful and reliable in an effort to instill confidence in the recommendations that are created as a result of the analysis. In this step we will work to remove missing data values and ensure that the data we are working with is compatible and consistent to allow for true representation of the metrics to be measured and assessed.

In this analysis, we will utilize the IMDB and the Numbers Movie Budget datasets. We will work through cleaning these datasets below.

In [26]:
```
# check the numbers dataset for null entries
TN_movie_budget.isna().sum()
```

executed in 5ms, finished 13:45:55 2022-06-16

Out[26]:
```
id                   0
release_date         0
movie                0
production_budget    0
domestic_gross       0
worldwide_gross      0
dtype: int64
```

In [27]:
```
# check the IMBD movie ratings dataset for null entries
IMDB_movie_ratings.isna().sum()
```

executed in 7ms, finished 13:45:55 2022-06-16

Out[27]:
```
movie_id         0
averagerating    0
numvotes         0
dtype: int64
```

In [28]:
```
# check the IMBD movie basics dataset for null entries
IMDB_movie_basics.isna().sum()
```

executed in 30ms, finished 13:45:56 2022-06-16

Out[28]:
```
movie_id              0
primary_title         0
original_title       21
start_year            0
runtime_minutes    31739
genres              5408
dtype: int64
```

In [29]:
```
# check the IMBD persons dataset for null entries
IMDB_persons.isna().sum()
```

executed in 77ms, finished 13:45:56 2022-06-16

Out[29]:
```
person_id                0
primary_name             0
birth_year          523912
death_year          599865
primary_profession   51340
dtype: int64
```

In [30]:
```
# check the IMBD principals dataset for null entries
IMDB_principals.isna().sum()
```

executed in 198ms, finished 13:45:56 2022-06-16

Out[30]:
```
movie_id           0
ordering           0
person_id          0
category           0
job           850502
characters    634826
dtype: int64
```

Above, the movie budget dataset and the IMDB movie ratings dataset does not appear to have any null entries, while we can see that there are null entries that we will need to address in the movie basics, persons, and principals datasets.

Drop Unused Columns

Within the IMDB movie basics dataset there are three columns of data that have missing values. From the above information we can see that the IMDB movie basics dataset has over 140K rows of data. Original title is missing 21 entries, compared to the large volume of data represented, generally this level of missing data would not significantly impact results. Although, this field is an object and cannot be normalized by using a mean to better represent the data more wholly. In the future analysis the original title is not utilized, it is assumed that the best path forward for this field is to remove the column completely since it will not impact further analysis.

As it relates to the runtime_minutes, this field is numerical as a float so we could normalize the data by calculating the mean or mode and use that to represent the missing fields, though similar to the original title, this data is not critical to scope of this analysis and can be removed from the dataset.

We will use similar logic to remove unused columns from the persons, and principals datasets as well.

In [31]:
```
# look at the number of null entries for each column within the movie
IMDB_movie_basics.info()
```

executed in 31ms, finished 13:45:57 2022-06-16

```
<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
```

In [32]:
```
# drop columns with missing values
IMDB_movie_basics.drop(columns = ['original_title', 'runtime_minutes']
```

executed in 13ms, finished 13:45:57 2022-06-16

In [33]:
```
# check that the columns have been removed
IMDB_movie_basics.isna().sum()
```

executed in 21ms, finished 13:45:57 2022-06-16

Out[33]:
```
movie_id            0
primary_title       0
start_year          0
genres           5408
dtype: int64
```

In [34]:
```
# look at the number of null entries for each column within the person
IMDB_persons.info()
```

executed in 77ms, finished 13:45:57 2022-06-16

```
<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 [35]:
```
# drop columns with missing values
IMDB_persons.drop(columns = ['birth_year', 'primary_profession'], inpl
```

executed in 21ms, finished 13:45:58 2022-06-16

In [36]:
```
# check that the columns have been removed
IMDB_persons.isna().sum()
```

executed in 65ms, finished 13:45:58 2022-06-16

Out[36]:
```
person_id            0
primary_name         0
death_year      599865
dtype: int64
```

Special Note: Although the persons dataset has a number of null entries in the death year column We will not remove those entries as the assumption would be that those persons without a death year are still alive. This will be valuable information later on in our analysis as we look to identify current directors and writers to partner with.

In [37]:
```
# look at the number of null entries for each column within the pricip
IMDB_principals.info()
```

executed in 214ms, finished 13:45:58 2022-06-16

```
<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 [38]:
```
# drop columns with missing values
IMDB_principals.drop(columns = ['job', 'characters'], inplace=True )
```

executed in 55ms, finished 13:45:58 2022-06-16

In [39]:
```
# check that the columns have been removed
IMDB_principals.isna().sum()
```

executed in 138ms, finished 13:45:58 2022-06-16

Out[39]:
```
movie_id     0
ordering     0
person_id    0
category     0
dtype: int64
```

Dropping Unused Rows

One significant area of our focus for the scope of this analysis is on the financial and public interest performance of movies based on genre. Given the importance of ensuring this information is handled correctly for the viability of proper analysis it is critical to be thoughtful about how to proceed with addressing the 5,408 missing values within this column.

In [40]:
```
# calculate the % of the entries that have missing values
print (round(len(IMDB_movie_basics[IMDB_movie_basics.genres.isna()]) /
       '% of the entries in the dataset have missing values in the gen
```

executed in 14ms, finished 13:45:59 2022-06-16

```
3.7 % of the entries in the dataset have missing values in the genre
column
```

Given the low % representation of missing data, it is safe to say that the reliability or trustworthiness of the data that is represented is still viable. Given that the data type is an object, it is not possible to calculate a mean or mode value, so the next viable option is to drop the rows where the genre column is empty.

In [41]:
```
# remove all remaining rows with missing values
IMDB_movie_basics = IMDB_movie_basics.dropna()

# check the dataset to see if there are any remaining missing values
IMDB_movie_basics.isna().sum()
```

executed in 55ms, finished 13:45:59 2022-06-16

Out[41]:
```
movie_id         0
primary_title    0
start_year       0
genres           0
dtype: int64
```

Data Analysis & Recommendations

Business Recommendation 1: Optimizing Portfolio Success Through Financial Performance

In this portion of the analysis we will look at the following elements to better understand the movies that are most profitable to better inform the Microsoft executive team's approach towards entering the movie industry. To accomplish this we will focus on:

- Genre success over time measured by their return on investment
- The data set that could be helpful for interpreting financial performance will be further explored
  - The Numbers
  - IMDB Movie Basics, Persons, Principals, and Movie Ratings

First we will take a look at the genre with the highest frequency. Initially, and very basically, one would assume that the most frequent occurrence would lend itself to the most popular type of movie in terms of public interest and assumed to be financial performance.

It is important to note that movies often times are categorized with multiple genres, so the information represented is not a one to one relationship with movies.

To allow for further exploration of genres in a meaningful way, we will need to turn the genres column within the IMDB Movie Basics dataset into a string and then work to separate the newly created genres_split column by comma. We will use the explode method to represent a movie with multiple genres on a separate row.

In [42]:
```
# turn the genres column into a string which will be used to be separa
IMDB_movie_basics['genres_split'] = IMDB_movie_basics.genres.str.split
```

executed in 137ms, finished 13:45:59 2022-06-16

In [43]:
```
# look at the frist 5 rows of the movie basics dataset to see the new
IMDB_movie_basics.head()
```

executed in 7ms, finished 13:45:59 2022-06-16

Out[43]:

	movie_id	primary_title	start_year	genres	genres_split
0	tt0063540	Sunghursh	2013	Action,Crime,Drama	[Action, Crime, Drama]
1	tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	[Biography, Drama]
2	tt0069049	The Other Side of the Wind	2018	Drama	[Drama]
3	tt0069204	Sabse Bada Sukh	2018	Comedy,Drama	[Comedy, Drama]
4	tt0100275	The Wandering Soap Opera	2017	Comedy,Drama,Fantasy	[Comedy, Drama, Fantasy]

In [44]:
```
# separate the genres_split column by comma to be better analyzed
exploded_IMDB_basics = IMDB_movie_basics.explode('genres_split')
exploded_IMDB_basics.head()
```

executed in 251ms, finished 13:46:00 2022-06-16

Out[44]:

	movie_id	primary_title	start_year	genres	genres_split
0	tt0063540	Sunghursh	2013	Action,Crime,Drama	Action
0	tt0063540	Sunghursh	2013	Action,Crime,Drama	Crime
0	tt0063540	Sunghursh	2013	Action,Crime,Drama	Drama
1	tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Biography
1	tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Drama

In [45]:
```
# find the proportion of each genres representation in the movie basic
genre_count = exploded_IMDB_basics.groupby('genres_split')['genres'].c
genre_sum = sum(genre_count)
genre_count_proportion = round(genre_count / genre_sum * 100, 2)
genre_count_proportion
```

executed in 48ms, finished 13:46:00 2022-06-16

```
Drama            21.73
Family            2.71
Fantasy           1.53
Game-Show         0.00
History           2.71
Horror            4.71
Music             1.88
Musical           0.62
Mystery           2.03
News              0.68
Reality-TV        0.04
Romance           4.08
Sci-Fi            1.47
Short             0.00
Sport             0.97
Talk-Show         0.02
Thriller          5.18
War               0.61
Western           0.20
Name: genres, dtype: float64
```

Earlier in this analysis we created a query to look at the IMDB movie ratings table. The table is listed below as a refresher.

In [46]:
```
# look at the first 5 rows of the IMDB movie ratings dataset
IMDB_movie_ratings.head()
```

executed in 5ms, finished 13:46:00 2022-06-16

Out[46]:

	movie_id	averagerating	numvotes
0	tt10356526	8.3	31
1	tt10384606	8.9	559
2	tt1042974	6.4	20
3	tt1043726	4.2	50352
4	tt1060240	6.5	21

In an effort to understand public interest in movies by genre, we will merge the exploded movie basics table with the movie ratings table to then further analyze and visualize this metric

Merge IMDB Movie Basics with Movie Ratings

In [47]:
```
# set the index column for IMDB movie ratings
IMDB_movie_ratings.set_index('movie_id', inplace=True)
IMDB_movie_ratings.head()
```

executed in 7ms, finished 13:46:02 2022-06-16

Out[47]:

movie_id	averagerating	numvotes
tt10356526	8.3	31
tt10384606	8.9	559
tt1042974	6.4	20
tt1043726	4.2	50352
tt1060240	6.5	21

In [48]:
```
# set the index column for exploded IMDB basics
exploded_IMDB_basics.set_index('movie_id', inplace=True)
exploded_IMDB_basics.head()
```

executed in 33ms, finished 13:46:02 2022-06-16

Out[48]:

movie_id	primary_title	start_year	genres	genres_split
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action
tt0063540	Sunghursh	2013	Action,Crime,Drama	Crime
tt0063540	Sunghursh	2013	Action,Crime,Drama	Drama
tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Biography
tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Drama

In [49]:
```
IMDB_basics_ratings = exploded_IMDB_basics.join(IMDB_movie_ratings, ho
IMDB_basics_ratings.head()
```

executed in 273ms, finished 13:46:02 2022-06-16

Out[49]:

movie_id	primary_title	start_year	genres	genres_split	averagerating	nu
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Crime	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Drama	7.0	
tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Biography	7.2	
tt0066787	One Day Before the Rainy Season	2019	Biography,Drama	Drama	7.2	

Later on in this analysis we will also benefit from having information from the principals and persons datasets. We will join them now with the ratings and basics datasets to have one comprehensive list from the IMDB Dataset that we will continue to work from throughout this analysis

In [50]:
```
# set the index column for principals
IMDB_principals.set_index('person_id', inplace=True)
IMDB_principals.head()
```

executed in 56ms, finished 13:46:03 2022-06-16

Out[50]:

person_id	movie_id	ordering	category
nm0246005	tt0111414	1	actor
nm0398271	tt0111414	2	director
nm3739909	tt0111414	3	producer
nm0059247	tt0323808	10	editor
nm3579312	tt0323808	1	actress

In [51]:
```
# set the index column for persons
IMDB_persons.set_index('person_id', inplace=True)
IMDB_persons.head()
```

executed in 38ms, finished 13:46:03 2022-06-16

Out[51]:

person_id	primary_name	death_year
nm0061671	Mary Ellen Bauder	NaN
nm0061865	Joseph Bauer	NaN
nm0062070	Bruce Baum	NaN
nm0062195	Axel Baumann	NaN
nm0062798	Pete Baxter	NaN

In [52]:
```
# join principals and persons
principals_persons = IMDB_principals.join(IMDB_persons, how = 'left')
principals_persons.head()
```

executed in 2.35s, finished 13:46:06 2022-06-16

Out[52]:

person_id	movie_id	ordering	category	primary_name	death_year
nm0000002	tt1626811	4	self	Lauren Bacall	2014.0
nm0000002	tt0858500	2	actress	Lauren Bacall	2014.0
nm0000002	tt1368858	1	actress	Lauren Bacall	2014.0
nm0000002	tt2053352	4	archive_footage	Lauren Bacall	2014.0
nm0000003	tt2004245	1	archive_footage	Brigitte Bardot	NaN

In [53]:
```
# set the index column for pricipals_persons
principals_persons.set_index('movie_id', inplace=True)
principals_persons.head()
```

executed in 42ms, finished 13:46:06 2022-06-16

Out[53]:

movie_id	ordering	category	primary_name	death_year
tt1626811	4	self	Lauren Bacall	2014.0
tt0858500	2	actress	Lauren Bacall	2014.0
tt1368858	1	actress	Lauren Bacall	2014.0
tt2053352	4	archive_footage	Lauren Bacall	2014.0
tt2004245	1	archive_footage	Brigitte Bardot	NaN

In [54]:
```
IMDB_full_details = IMDB_basics_ratings.join(principals_persons, how =
IMDB_full_details.head()
```

executed in 1.02s, finished 13:46:08 2022-06-16

Out[54]:

movie_id	primary_title	start_year	genres	genres_split	averagerating	nu
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	
tt0063540	Sunghursh	2013	Action,Crime,Drama	Action	7.0	

Analyze Average Movie Ratings by Genre

In [55]:
```
# find the average rating of each genres representation in the movie b
ratings_mean = round(IMDB_full_details.groupby('genres_split')['averag
ratings_mean
```

executed in 84ms, finished 13:46:08 2022-06-16

Out[55]:
```
genres_split
Action         5.78
Adult          3.40
Adventure      6.11
Animation      6.25
Biography      7.10
Comedy         5.97
Crime          6.08
Documentary    7.33
Drama          6.38
Family         6.31
Fantasy        5.88
Game-Show      7.30
History        6.96
Horror         5.00
Music          7.03
Musical        6.46
Mystery        5.91
News           7.28
Reality-TV     6.10
Romance        6.13
Sci-Fi         5.46
Short          8.80
Sport          6.90
Talk-Show       NaN
Thriller       5.64
War            6.49
Western        5.84
Name: averagerating, dtype: float64
```

In [56]:
```
# import the necessary packages to complete data visualizations
import matplotlib.pyplot as plt
%matplotlib inline
```

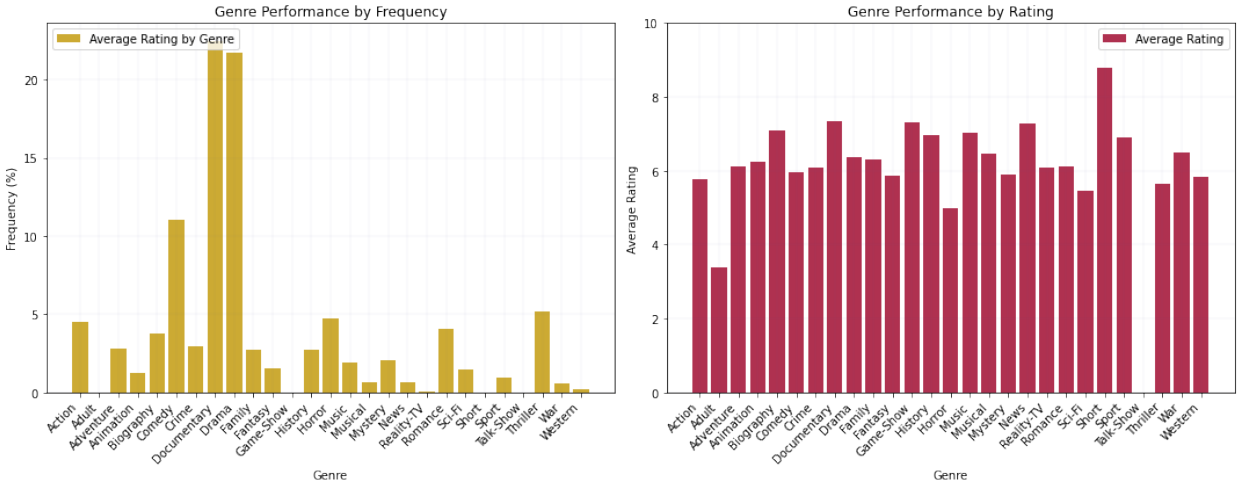
executed in 241ms, finished 13:46:08 2022-06-16

Below we will plot a histogram to visualize movie average ratings and frequency of occurrence based on genre.

In [57]:

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph measuring f
ax1.bar(genre_count_proportion.index, genre_count_proportion.values,
       color = '#CCAA33',
       align='center',
       label = 'Average Rating by Genre')
ax1.set_ylabel('Frequency (%)')
ax1.set_xlabel('Genre')
ax1.set_title('Genre Performance by Frequency')
plt.setp(ax1.get_xticklabels(), rotation=45, ha='right') # cite: https
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph measuring
ax2.bar(ratings_mean.index, ratings_mean.values,
       color = '#AF3150',
       align='center',
       label = 'Average Rating')
ax2.set_ylabel('Average Rating')
ax2.set_xlabel('Genre')
ax2.set_title('Genre Performance by Rating')
plt.setp(ax2.get_xticklabels(), rotation=45, ha='right') # cite: https
major_ticks = np.arange(0, 12, 2)
ax2.set_yticks(major_ticks)
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 426ms, finished 13:46:09 2022-06-16



Based on the interpretation of the visualizations above we would assume the following:

- Documentary (22% of films) and Drama (21% of films) films were the most frequently created films. One could make an assumption that this is a result of public interest and reception of these types of films, which would then likely lead to profitability of this type of genre of film
- Short films have the highest average rating on a ten point scale. This would allow one to believe that since short films (8.8 rating out of 10) have a high rating they would likely be well received and potentially profitable.

We will continue to explore these assumptions throughout this analysis.

Merge IMDB Movie Basics to The Numbers Movie Budget Dataset

To further analyze movie performance financially we will connect the IMDB full details list with the Numbers Movie Budget Dataset. The numbers dataset will allow us to compare their listing of production budget and gross sales to better understand the profitability of genres more broadly.

In [58]:
```
# merge the IMDB full details list with the Number Movie Budget data
# merging with the Numbers data on the left since this dataset has few
# we will be limited to their available information
full_movie_details = TN_movie_budget.merge(IMDB_full_details,
                     left_on='movie', right_on='primary_title')
full_movie_details.head()
```

executed in 184ms, finished 13:46:10 2022-06-16

Out[58]:

worldwide_gross	primary_title	start_year	genres	genres_split	averagerating	numv
$2,776,345,279	Avatar	2011	Horror	Horror	6.1	
$2,776,345,279	Avatar	2011	Horror	Horror	6.1	
$2,776,345,279	Avatar	2011	Horror	Horror	6.1	
$2,776,345,279	Avatar	2011	Horror	Horror	6.1	
$2,776,345,279	Avatar	2011	Horror	Horror	6.1	

In [59]:
```
# look at the merged dataset
full_movie_details.info()
```

executed in 40ms, finished 13:46:10 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69673 entries, 0 to 69672
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 69673 non-null  int64  
 1   release_date       69673 non-null  object 
 2   movie              69673 non-null  object 
 3   production_budget  69673 non-null  object 
 4   domestic_gross     69673 non-null  object 
 5   worldwide_gross    69673 non-null  object 
 6   primary_title      69673 non-null  object 
 7   start_year         69673 non-null  int64  
 8   genres             69673 non-null  object 
 9   genres_split       69673 non-null  object 
 10  averagerating      61871 non-null  float64
 11  numvotes           61871 non-null  float64
 12  ordering           69649 non-null  float64
 13  category           69649 non-null  object 
 14  primary_name       69644 non-null  object 
 15  death_year         1421 non-null   float64
dtypes: float64(4), int64(2), object(10)
memory usage: 9.0+ MB
```

In [60]:
```
# drop unused columns and columns with missing values
full_movie_details.drop(columns = ['primary_title', 'start_year', 'ave
```

executed in 11ms, finished 13:46:11 2022-06-16

In [61]:
```
# drop rows with missing values in category since it cannot be normali
# we will not remove null entries in death year as we want to preserve
# things further with actively alive writers and directors
full_movie_details.dropna(subset=['category', 'primary_name'], inplace
```

executed in 41ms, finished 13:46:11 2022-06-16

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

executed in 43ms, finished 13:46:12 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69644 entries, 0 to 69672
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 69644 non-null  int64  
 1   release_date       69644 non-null  object 
 2   movie              69644 non-null  object 
 3   production_budget  69644 non-null  object 
 4   domestic_gross     69644 non-null  object 
 5   worldwide_gross    69644 non-null  object 
 6   genres             69644 non-null  object 
 7   genres_split       69644 non-null  object 
 8   category           69644 non-null  object 
 9   primary_name       69644 non-null  object 
 10  death_year         1421 non-null   float64
dtypes: float64(1), int64(1), object(9)
memory usage: 6.4+ MB
```

In [63]:
```
# given that movies are represented on multiple lines since they have
# associated with a single film, we will not look to understand how ma
# films are in this dataset
distinct = len(pd.unique(full_movie_details['movie']))
print("Number of unique movies that we will analyze further:", distinc
```

executed in 19ms, finished 13:46:13 2022-06-16

```
Number of unique movies that we will analyze further: 2298
```

Above we can see that the production budget, domestic gross, and worldwide gross are an object type, though we would like for them to be represented as an integer. Also release date is an object, we will want to convert this to a data format and then separate into a month a year column to be used in further analysis. We want them to convert to an integer. We will address this below.

In [64]:
```
# convert production budget from object to integer
full_movie_details['production_budget'] = full_movie_details['producti
```

executed in 87ms, finished 13:46:15 2022-06-16

In [65]:
```
# convert domestic gross from object to integer
full_movie_details['domestic_gross'] = full_movie_details['domestic_gr
```

executed in 67ms, finished 13:46:17 2022-06-16

In [66]:
```
# convert worldwide gross from object to integer
full_movie_details['worldwide_gross'] = full_movie_details['worldwide_
```
executed in 66ms, finished 13:46:19 2022-06-16

In [67]:
```
# convert release date from object to date format
full_movie_details['release_date'] = pd.to_datetime(full_movie_details
```
executed in 94ms, finished 13:46:21 2022-06-16

In [68]:
```
# add a new column that only includes month from release date
full_movie_details['month'] = pd.DatetimeIndex(full_movie_details['rel
```
executed in 6ms, finished 13:46:21 2022-06-16

In [69]:
```
# add a new column that only includes the day from release date
full_movie_details['day'] = pd.DatetimeIndex(full_movie_details['relea
```
executed in 14ms, finished 13:46:22 2022-06-16

In [70]:
```
# check the full movie details dataset for no null entries, the adjust
# new columns
full_movie_details.info()
```
executed in 44ms, finished 13:46:24 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69644 entries, 0 to 69672
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 69644 non-null  int64         
 1   release_date       69644 non-null  datetime64[ns]
 2   movie              69644 non-null  object        
 3   production_budget  69644 non-null  int64         
 4   domestic_gross     69644 non-null  int64         
 5   worldwide_gross    69644 non-null  int64         
 6   genres             69644 non-null  object        
 7   genres_split       69644 non-null  object        
 8   category           69644 non-null  object        
 9   primary_name       69644 non-null  object        
 10  death_year         1421 non-null   float64       
 11  month              69644 non-null  int64         
 12  day                69644 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(6), object(5)
memory usage: 7.4+ MB
```

In [71]:
```
full_movie_details.head()
```

executed in 25ms, finished 13:46:26 2022-06-16

Out[71]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	2009-12-18	Avatar	425000000	760507625	2776345279
1	1	2009-12-18	Avatar	425000000	760507625	2776345279
2	1	2009-12-18	Avatar	425000000	760507625	2776345279
3	1	2009-12-18	Avatar	425000000	760507625	2776345279
4	1	2009-12-18	Avatar	425000000	760507625	2776345279

Compare Production Budget against Gross Sales

In [72]:
```
# find the average production budget of each genres representation
# in the merged full_movie_details dataset
production_mean_genres = round(full_movie_details.groupby('genres_spli
                                mean(), 0)
production_mean_genres
```

executed in 27ms, finished 13:46:30 2022-06-16

```
Animation      83588220.0
Biography      25277735.0
Comedy         33513922.0
Crime          27600020.0
Documentary    23185315.0
Drama          23464711.0
Family         49684181.0
Fantasy        67400869.0
History        35001607.0
Horror         18725366.0
Music          15748796.0
Musical        38792835.0
Mystery        22416109.0
News           26733333.0
Reality-TV      1000000.0
Romance        20457060.0
Sci-Fi         67628715.0
Sport          24647971.0
Thriller       27217754.0
War            24132955.0
```

In [73]:
```
# find the average domestic production of each genres representation
# in the merged full_movie_details dataset
domestic_gross_mean_genres = round(full_movie_details.groupby('genres_
                                   mean(), 0)
domestic_gross_mean_genres
```

executed in 27ms, finished 13:46:46 2022-06-16

```
Biography        33818533.0
Comedy           47019171.0
Crime            30188175.0
Documentary      28296439.0
Drama            28225497.0
Family           71085513.0
Fantasy          81538728.0
History          34907317.0
Horror           28550294.0
Music            30047837.0
Musical         100341883.0
Mystery          31586043.0
News             14858212.0
Reality-TV             0.0
Romance          28987680.0
Sci-Fi           87473618.0
Sport            34740147.0
Thriller         32265561.0
War              22717227.0
Western          34718831.0
```

In [74]:
```
# find the average worldwide production of each genres representation
# in the merged full_movie_details dataset
global_gross_mean_genres = round(full_movie_details.groupby('genres_sp
                                                mean(), 0)
global_gross_mean_genres
```

executed in 31ms, finished 13:48:27 2022-06-16

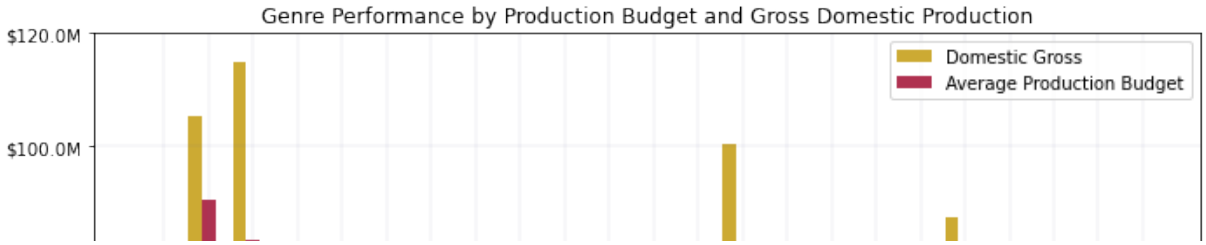
```
Comedy          104112928.0
Crime            65045955.0
Documentary      57422911.0
Drama            59338170.0
Family          168226437.0
Fantasy         218614818.0
History          74866117.0
Horror           64122869.0
Music            64019916.0
Musical         221427300.0
Mystery          69596847.0
News             52699432.0
Reality-TV             0.0
Romance          61369778.0
Sci-Fi          234937933.0
Sport            68181053.0
Thriller         78533865.0
War              51062352.0
Western          66656409.0
Name: worldwide_gross, dtype: float64
```

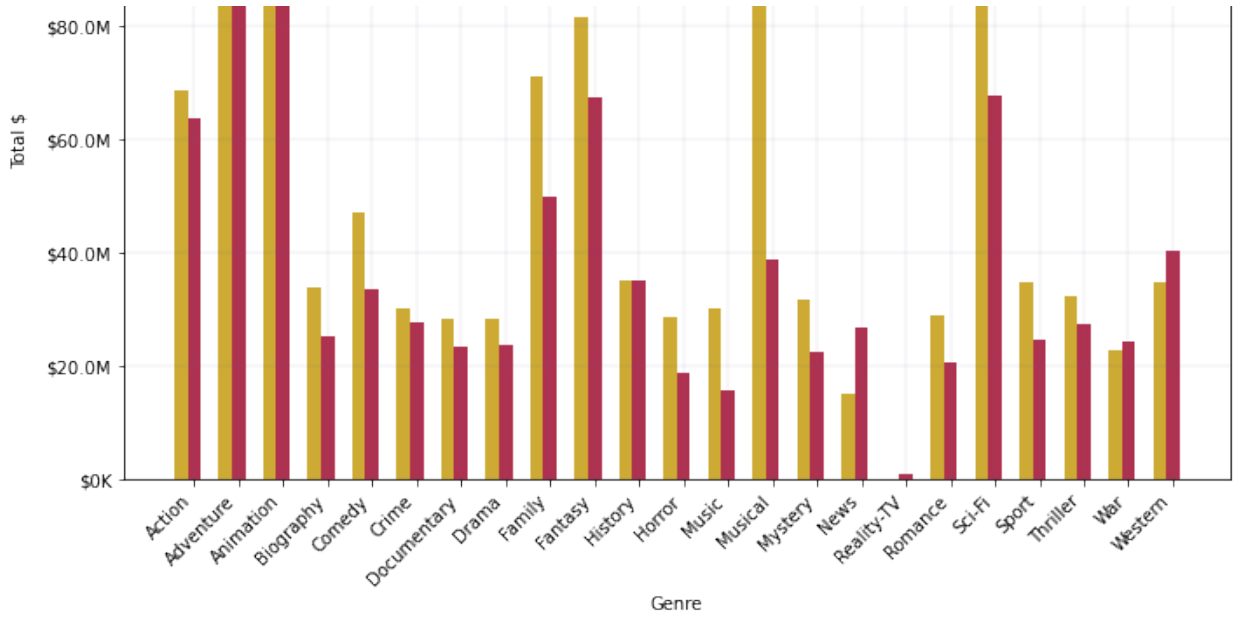
plot a histogram to visualize movie budget and the domestic gross production based on genre

In [75]:

```
#plot a histogram to visualize movie budget and the domestic gross pro

# this function is used for formatting the y axis without scientific n
# cite: https://matplotlib.org/stable/tutorials/introductory/lifecycle
def currency(x, pos):
    if x >= 1e6:
        s = '${:1.1f}M'.format(x * 1e-6)
    else:
        s = '${:1.0f}K'.format(x * 1e-3)
    return s

# establish x values for a multi dataset visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = domestic_gross_mean_genres.index
x = np.arange(len(labels))

# set the plot, format, and labels
fig, ax = plt.subplots(figsize=(10,7))
bar_width = 0.3
ax1 = ax.bar(x - bar_width / 2,
             domestic_gross_mean_genres.values,
             width = 0.3,
             color = '#CCAA33',
             label='Domestic Gross')
ax2 = ax.bar(x + bar_width / 2,
             production_mean_genres.values,
             width = 0.3,
             color = '#AF3150',
             label='Average Production Budget')
ax.yaxis.set_major_formatter(currency)
plt.ylim(0,120000000)
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(domestic_gross_mean_genres.index.unique())
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Genre')
ax.set_ylabel('Total $')

# Add title and legends
ax.set_title('Genre Performance by Production Budget and Gross Domesti
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 259ms, finished 13:48:45 2022-06-16

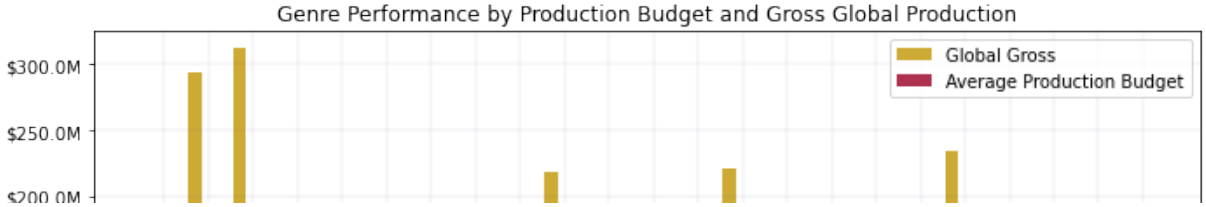


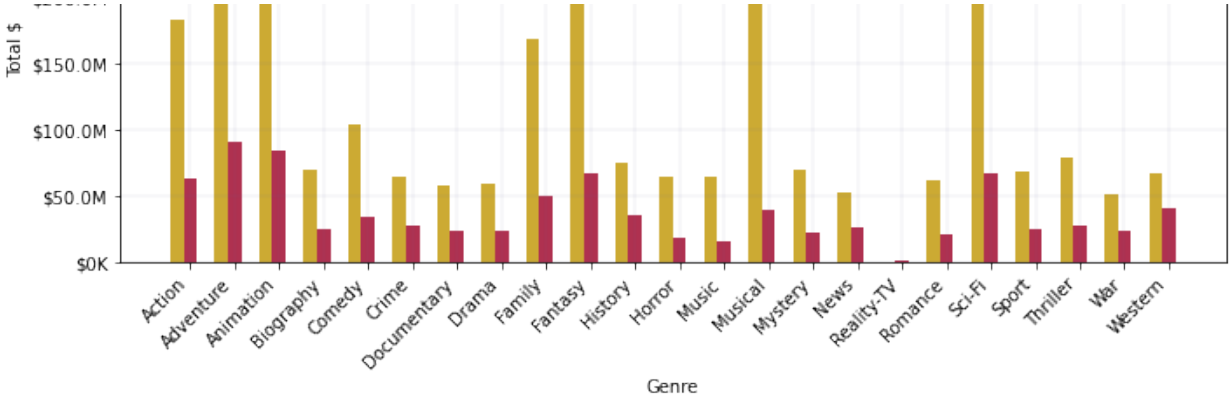

$80.0M
$60.0M
$40.0M
$20.0M
$0K
Total $
Action
Adventure
Animation
Biography
Comedy
Crime
Documentary
Drama
Family
Fantasy
History
Horror
Music
Musical
Mystery
News
Reality-TV
Romance
Sci-Fi
Sport
Thriller
War
Western
Genre

In [76]:

```
#plot a histogram to visualize movie budget and the global gross produ

# this function is used for formatting the y axis without scientific n
# cite: https://matplotlib.org/stable/tutorials/introductory/lifecycle
def currency(x, pos):
    if x >= 1e6:
        s = '${:1.1f}M'.format(x * 1e-6)
    else:
        s = '${:1.0f}K'.format(x * 1e-3)
    return s

# establish x values for a multi dataset visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = global_gross_mean_genres.index
x = np.arange(len(labels))

# set the plot, format, and labels
fig, ax = plt.subplots(figsize=(10,5))
bar_width = 0.3
ax1 = ax.bar(x - bar_width / 2,
             global_gross_mean_genres.values,
             width = 0.3,
             color = '#CCAA33',
             label='Global Gross')
ax2 = ax.bar(x + bar_width / 2,
             production_mean_genres.values,
             width = 0.3,
             color = '#AF3150',
             label='Average Production Budget')
ax.yaxis.set_major_formatter(currency)
plt.ylim(0,325000000)
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(global_gross_mean_genres.index.unique())
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Genre')
ax.set_ylabel('Total $')


# Add title and legends
ax.set_title('Genre Performance by Production Budget and Gross Global
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 221ms, finished 13:48:51 2022-06-16





Above we can see that a higher budget lends itself to a higher sales production, particularly when you look at global production. Global animation films on average have over $300M in sales. Though this does not tell us a full story - we need to evaluate return on investment to understand the full scope of profitability.

Calculate Return on Investment

Return on investment is a % measurement to interpret how much profit was gained from the about of money initially invested to create the movies. It is calculated as follows:

ROI = ((amount gained - amount spent)/amount spent) * 100

Below we will explore ROI as it relates to genres to better understand how Microsoft might be able to use this information to their advantage as they enter the movie industry.

In [77]:
```
# calculate return on investment by genre (domestic)
ROI_domestic = round(((domestic_gross_mean_genres - production_mean_ge
                      production_mean_genres) * 100, 2)
ROI_domestic
```

executed in 16ms, finished 13:50:22 2022-06-16

Out[77]:

```
genres_split
Action            7.48
Adventure        16.39
Animation        37.60
Biography        33.79
Comedy           40.30
Crime             9.38
Documentary      22.04
Drama            20.29
Family           43.07
Fantasy          20.98
History          -0.27
Horror           52.47
Music            90.79
Musical         158.66
Mystery          40.91
News            -44.42
Reality-TV     -100.00
Romance          41.70
Sci-Fi           29.34
Sport            40.95
Thriller         18.55
War              -5.87
Western         -13.57
dtype: float64
```

In [78]:
```
# calculate return on investment by genre (global)
ROI_global = round(((global_gross_mean_genres - production_mean_genres
                     production_mean_genres) * 100, 2)
ROI_global
```

executed in 12ms, finished 13:50:26 2022-06-16

```
Comedy          210.05
Crime           135.67
Documentary     147.67
Drama           152.88
Family          238.59
Fantasy         224.35
History         113.89
Horror          242.44
Music           306.51
Musical         470.79
Mystery         210.48
News             97.13
Reality-TV    -100.00
Romance         199.99
Sci-Fi          247.39
Sport           176.62
Thriller        188.54
War             111.59
Western          65.94
dtype: float64
```

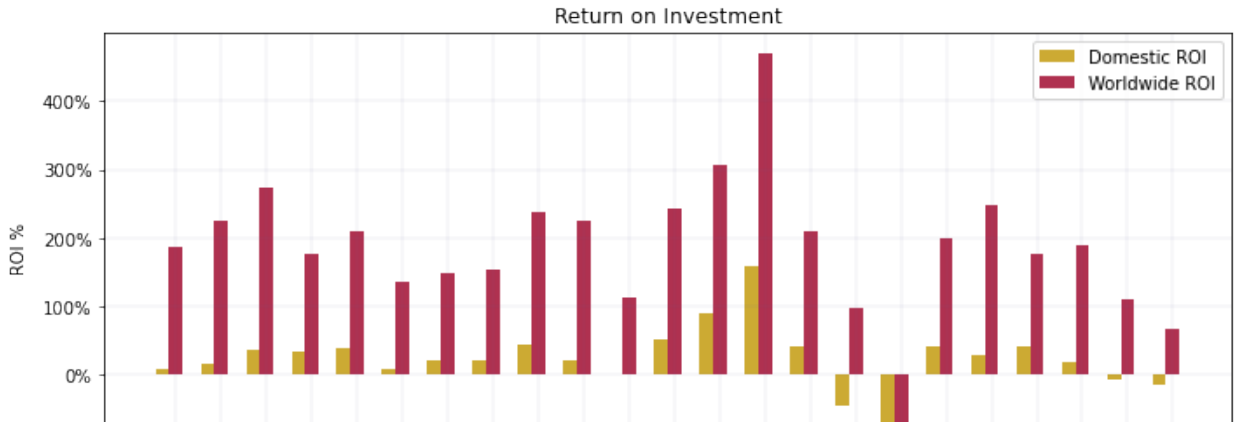
In [79]:
```
# import the necessary support packages for the visualization below
import matplotlib.ticker as mtick
```

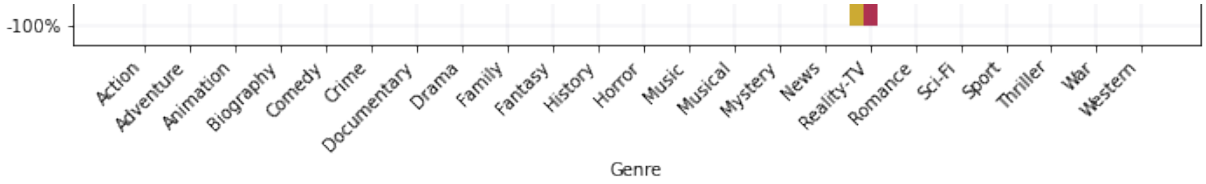
executed in 5ms, finished 13:50:31 2022-06-16

In [80]:

```
# establish x values for a multi dataset visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_domestic.index
x = np.arange(len(labels))

# set the plot, format, and labels
fig, ax = plt.subplots(figsize=(10,5))
bar_width = 0.3
ax1 = ax.bar(x - bar_width / 2,
             ROI_domestic.values,
             width = 0.3,
             color = '#CCAA33',
             label ='Domestic ROI')

ax2 = ax.bar(x + bar_width / 2,
             ROI_global.values,
             width = 0.3,
             color = '#AF3150',
             label='Worldwide ROI')

# Add a legend to the plot
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Genre')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(ROI_global.index.unique())
ax.set_ylabel('ROI %')

# Add title and legends
ax.legend()
ax.set_title('Return on Investment')

# format the layout and display the visualization
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 280ms, finished 13:50:36 2022-06-16





Recommendation

This analysis worked to explore the profitability of genres within the movie industry. As Microsoft looks to enter the movie industry, pursuing films that have proven their profitability over time offers the potential to pursue profitability more readily. In the above image we can see that the most profitable movie genres, in terms of average return on investment, are Musical, Music, and Animation films.

Top Genre ROI:

- Musical 471%
- Music 307%
- Animation 274%

As another layer of pursuing proven paths toward successful movie industry entry we will explore writers and directors who have proven profitability.

We will continue to explore these genres further in this analysis.

Business Recommendation 2: Align with Prolific Director and Writer Teams

In this part of the analysis we will explore the profitability of directors - those who have proven themselves to consistently perform at an elevated level. Given that Microsoft is an established and trusted organization with capital to spend on strong partnerships, aligning with directors and writers that can create impact will be critical.

Below we will explore who those directors and writers are.

Directors

In [81]:
```
# look at the full movie details dataset as a refresh of what informat
full_movie_details.info()
```

executed in 116ms, finished 13:52:16 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69644 entries, 0 to 69672
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 69644 non-null  int64         
 1   release_date       69644 non-null  datetime64[ns]
 2   movie              69644 non-null  object        
 3   production_budget  69644 non-null  int64         
 4   domestic_gross     69644 non-null  int64         
 5   worldwide_gross    69644 non-null  int64         
 6   genres             69644 non-null  object        
 7   genres_split       69644 non-null  object        
 8   category           69644 non-null  object        
 9   primary_name       69644 non-null  object        
 10  death_year         1421 non-null   float64       
 11  month              69644 non-null  int64         
 12  day                69644 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(6), object(5)
memory usage: 7.4+ MB
```

As we look at directors and writers, it is important, if we are truly going to develop a strategy for future actions, we should not take into consideration individuals who have passed away. We will work to remove those individuals from the data below.

In [82]:
```
# remove directors that have passed away
alive_directors = full_movie_details.loc[full_movie_details['death_yea
alive_directors.head()
```

executed in 41ms, finished 13:54:08 2022-06-16

Out[82]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	2009-12-18	Avatar	425000000	760507625	2776345279
1	1	2009-12-18	Avatar	425000000	760507625	2776345279
2	1	2009-12-18	Avatar	425000000	760507625	2776345279
3	1	2009-12-18	Avatar	425000000	760507625	2776345279
4	1	2009-12-18	Avatar	425000000	760507625	2776345279

In [83]:
```
# look at the dataset information to ensure deceased individuals have
alive_directors.info()
```

executed in 44ms, finished 13:54:51 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68223 entries, 0 to 69672
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 68223 non-null  int64         
 1   release_date       68223 non-null  datetime64[ns]
 2   movie              68223 non-null  object        
 3   production_budget  68223 non-null  int64         
 4   domestic_gross     68223 non-null  int64         
 5   worldwide_gross    68223 non-null  int64         
 6   genres             68223 non-null  object        
 7   genres_split       68223 non-null  object        
 8   category           68223 non-null  object        
 9   primary_name       68223 non-null  object        
 10  death_year         0 non-null      float64       
 11  month              68223 non-null  int64         
 12  day                68223 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(6), object(5)
memory usage: 7.3+ MB
```

Below we will work to capture the data that only relates to directors.

In [84]:
```
# from the full movie details dataset sort by directors
category_type_director = ['director']

directors_groupby = alive_directors[alive_directors['category'].isin(c
directors_groupby.head()
```

executed in 34ms, finished 13:55:40 2022-06-16

Out[84]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
7	1	2009-12-18	Avatar	425000000	760507625	277634527
15	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387
25	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387
35	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387
48	3	2019-06-07	Dark Phoenix	350000000	42762350	14976235

We will now work towards understanding ROI by director (domestically and globally)

In [85]:
```
# find the average budget by director
directors_production_mean = round(directors_groupby.groupby('primary_n
                                  mean(), 0)
```

executed in 19ms, finished 13:55:53 2022-06-16

In [86]:
```
# find the average domestic gross sales by director
directors_domestic_gross_mean = round(directors_groupby.groupby('prima
                                      .mean(), 0)
```

executed in 17ms, finished 13:55:54 2022-06-16

In [87]:
```
# find the average global gross sales by director
directors_global_mean = round(directors_groupby.groupby('primary_name'
                                    mean(), 0)
```

executed in 18ms, finished 13:55:55 2022-06-16

In [88]:
```
# calculate the ROI by director (domestic), only display the top 25
ROI_domestic_directors = round(((directors_domestic_gross_mean - direc
                                    directors_production_mean) * 100, 2).
```

executed in 15ms, finished 13:56:44 2022-06-16

In [89]:
```
# calculate the ROI by director (global), only display the top 25
ROI_global_directors = round(((directors_global_mean - directors_produ
                                  directors_production_mean) * 100, 2).so
```

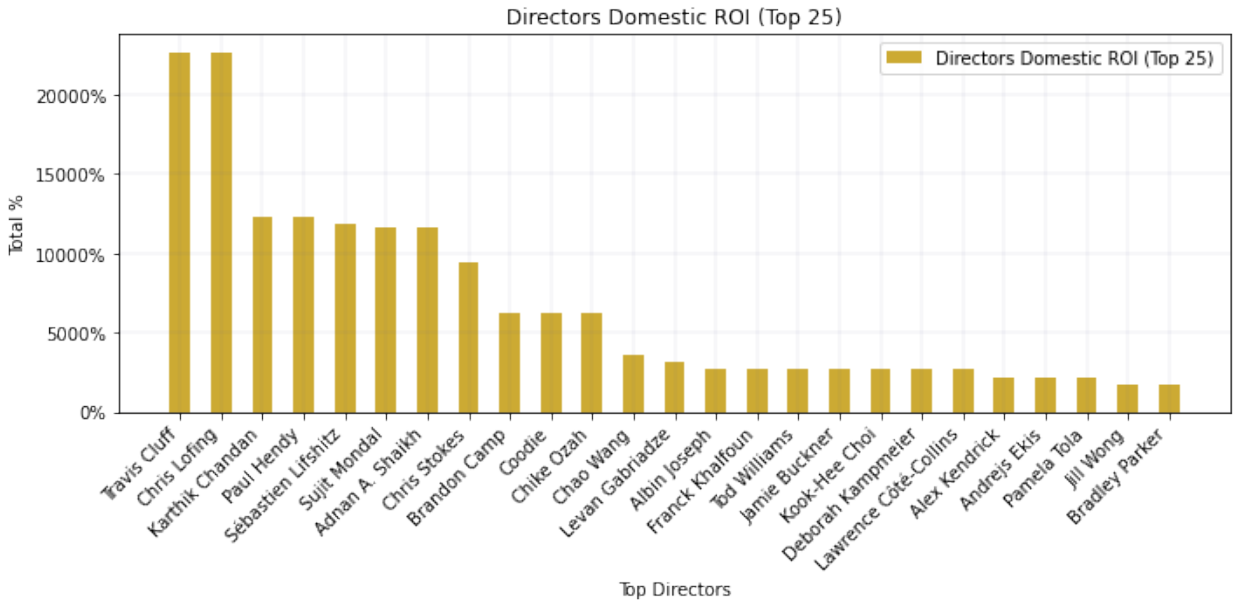
executed in 10ms, finished 13:56:48 2022-06-16

Visualize the ROI of top directors (domestic)

In [90]:

```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_domestic_directors.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_domestic_directors.values,
             width = 0.5,
             color = '#CCAA33',
             label='Directors Domestic ROI (Top 25)')

ax.set_xticks(x)
ax.set_xticklabels(labels)
#ax.yaxis.set_major_formatter(plt.ticker.StrMethodFormatter('{x:,.0f}'
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Directors')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Directors Domestic ROI (Top 25)')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

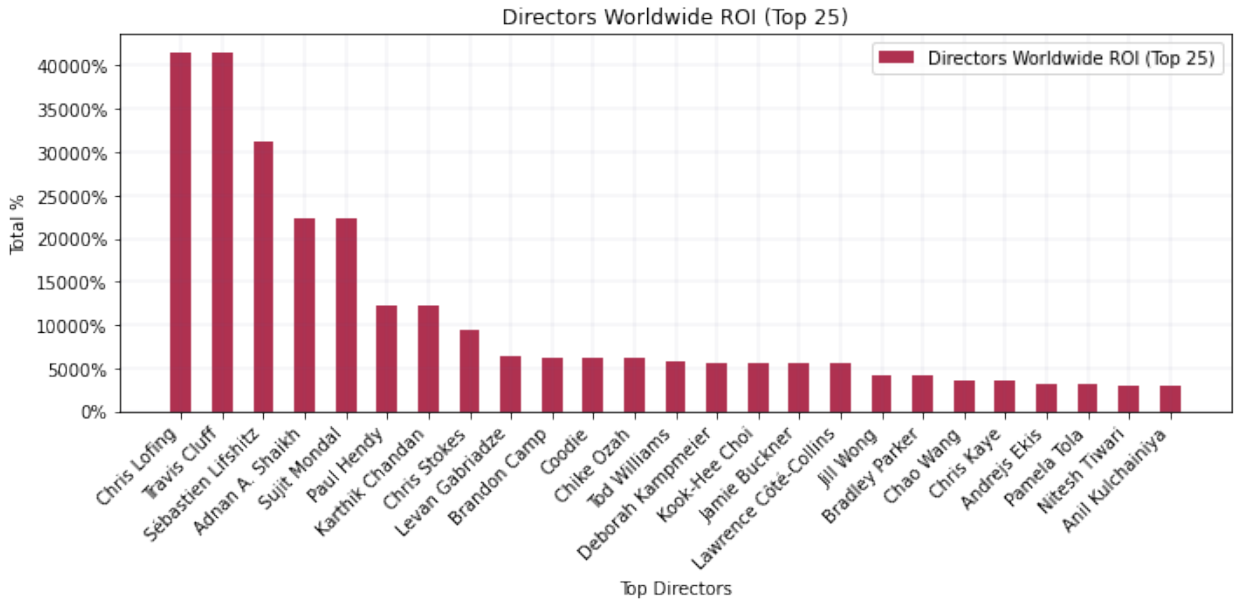
executed in 220ms, finished 13:57:19 2022-06-16



Visualize the ROI of top directors (global)

In [91]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_global_directors.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_global_directors.values,
             width = 0.5,
             color = '#AF3150',
             label='Directors Worldwide ROI (Top 25)')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Directors')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Directors Worldwide ROI (Top 25)')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 204ms, finished 13:57:25 2022-06-16



Above we were able to identify a list of directors who, regardless of genre, have proven themselves to be profitable. Note that this approach does not take into account the number of films a director has been involved with. Travis Cluff has only 9 director credits to his name, though his films have returned well. (cite: https://www.imdb.com/name/nm4000389/?ref_=fn_al_nm_1 (https://www.imdb.com/name/nm4000389/?ref_=fn_al_nm_1))

Below we will work to find the genres with the highest ROI when taking directors into account.

In [92]:
```
# find the average budget by genre when taking into account directors
directors_genre_production_mean = round(directors_groupby.groupby('ger
                                        mean(), 0)
```
executed in 14ms, finished 13:57:49 2022-06-16

In [93]:
```
# find the average domestic gross sales by genre when taking into acco
directors_genre_domestic_gross_mean = round(directors_groupby.groupby(
                                          .mean(), 0)
```
executed in 12ms, finished 13:57:50 2022-06-16

In [94]:
```
# find the average global gross sales by genre when taking into accoun
directors_genre_global_gross_mean = round(directors_groupby.groupby('g
                                       mean(), 0)
```
executed in 11ms, finished 13:57:50 2022-06-16

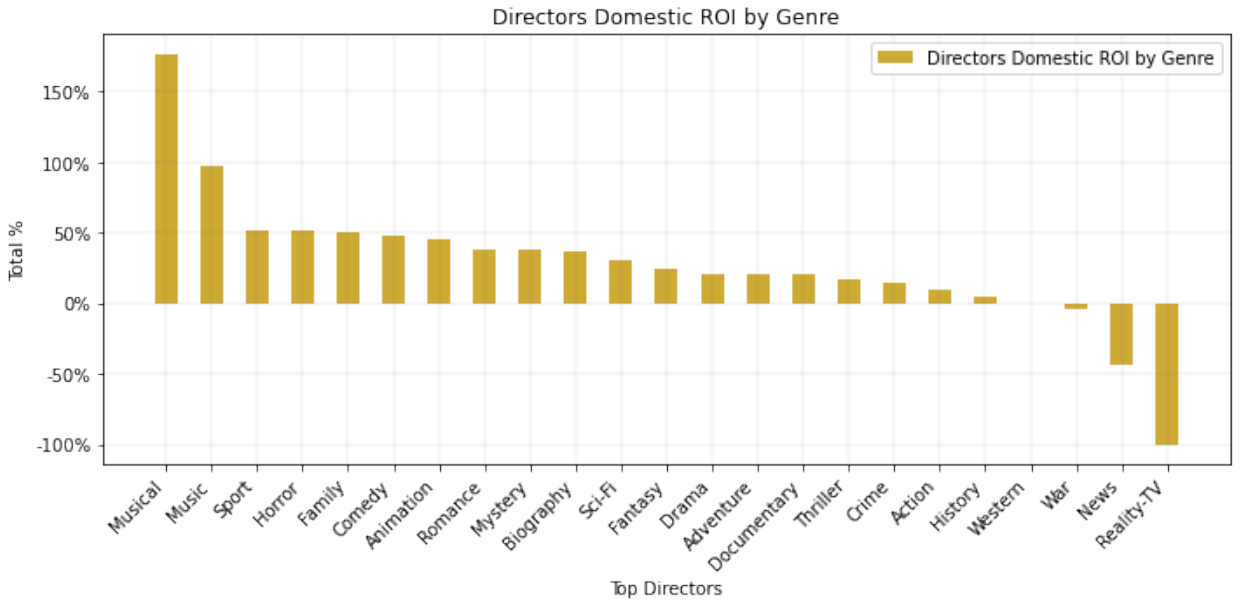
In [95]:
```
# calculate the ROI by genre (domestic) when taking into account direc
ROI_domestic_directors_genre = round(((directors_genre_domestic_gross_
                                       directors_genre_production_mean) * 10
```
executed in 10ms, finished 13:57:51 2022-06-16

In [96]:
```
# calculate the ROI by genre (global) when taking into account directo
ROI_global_directors_genre = round(((directors_genre_global_gross_mean
                                     directors_genre_production_mean)
```
executed in 8ms, finished 13:57:52 2022-06-16

Visualize the ROI of genres when taking into account directors (domestic)

In [97]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_domestic_directors_genre.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_domestic_directors_genre.values,
             width = 0.5,
             color = '#CCAA33',
             label='Directors Domestic ROI by Genre')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Directors')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Directors Domestic ROI by Genre')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

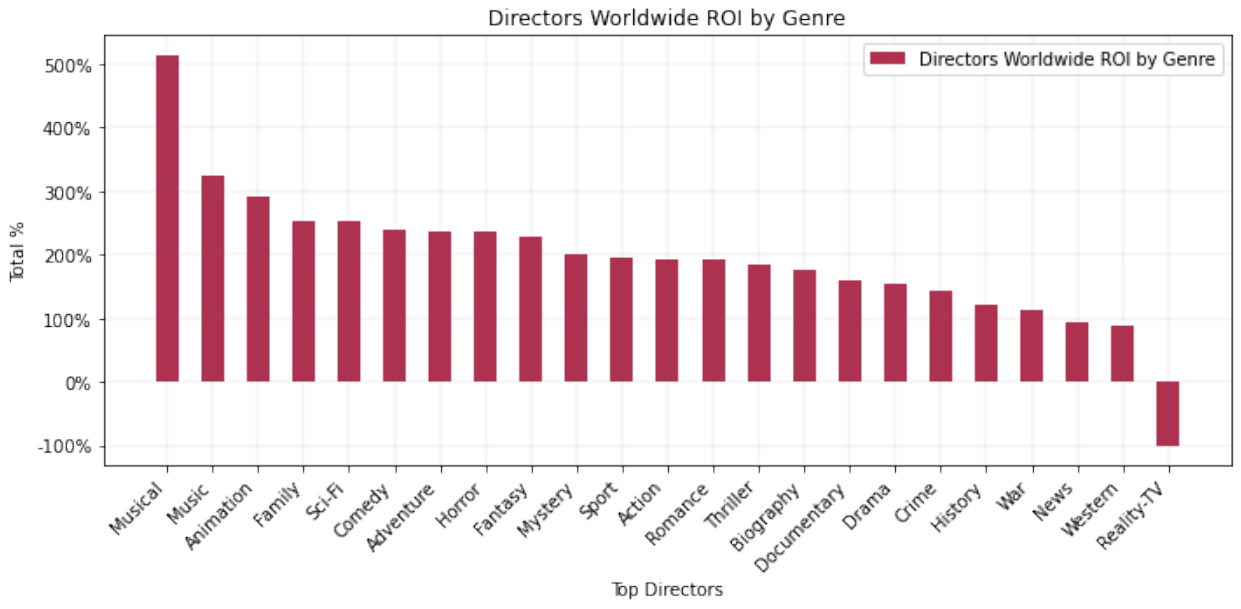
executed in 193ms, finished 13:58:03 2022-06-16



Visualize the ROI of genres when taking into account directors (global)

In [98]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_global_directors_genre.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_global_directors_genre.values,
             width = 0.5,
             color = '#AF3150',
             label='Directors Worldwide ROI by Genre')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Directors')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Directors Worldwide ROI by Genre')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 202ms, finished 13:58:09 2022-06-16



From the above visualizations we can see that when we take into account directors and look at the profitability of movies by genre Musical and Music films prevail with the highest ROI. Domestically, Family movies ranks third, globally, Animation movies rank third. For the duration of this analysis we will continue to look at Musical, Music, and Animation type films.

Top Genre ROI:

- Musical 471%
- Music 307%
- Animation 274%

We will explore the top five directors, by ROI, for the top three genres below. First we will look at the musical genre.

In [99]:
```
# from the directors list, group a list by only directors who have wor
category_type_musical = ['Musical']
directors_musical = directors_groupby[directors_groupby['genres_split'
```
executed in 13ms, finished 13:59:37 2022-06-16

In [100]:
```
# find the average budget for musical films
directors_musical_production_mean = round(directors_musical.groupby('p
                                    mean(), 0)
```
executed in 9ms, finished 13:59:38 2022-06-16

In [101]:
```
# find the average domestic gross sales for musical films
directors_musical_domestic_gross_mean = round(directors_musical.groupb
                                        .mean(), 0)
```
executed in 10ms, finished 13:59:39 2022-06-16

In [102]:
```
# find the average global gross sales for musical films
directors_musical_global_gross_mean = round(directors_musical.groupby(
                                      .mean(), 0)
```
executed in 7ms, finished 13:59:43 2022-06-16

In [103]:
```
# calculate the ROI for musical films by director, only display the to
ROI_domestic_directors_musical = round(((directors_musical_domestic_gr
                                 directors_musical_production_mean) *
```
executed in 8ms, finished 13:59:45 2022-06-16

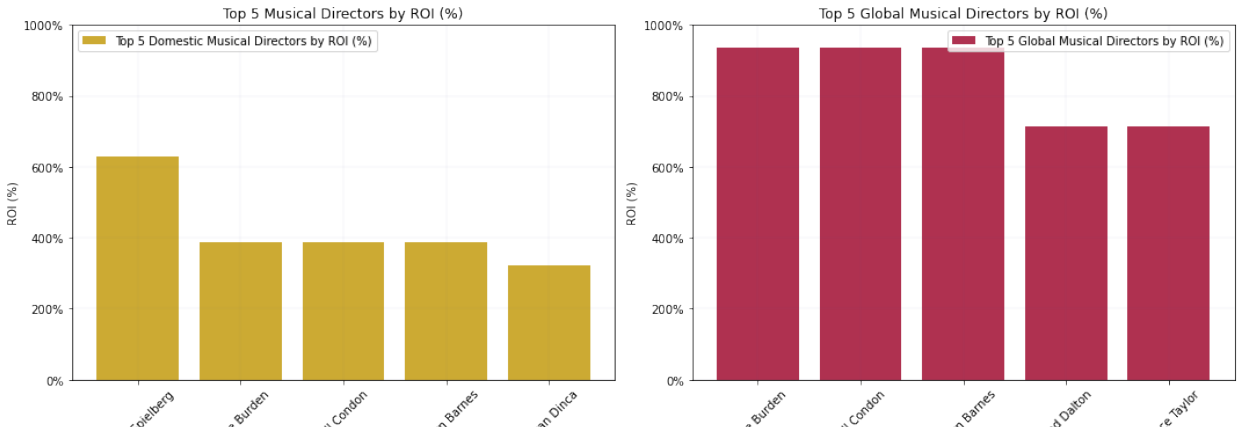
In [104]:
```
# calculate the ROI for musical films by director, only display the to
ROI_global_directors_musical = round(((directors_musical_global_gross_
                                    directors_musical_production_mean
```
executed in 9ms, finished 13:59:48 2022-06-16

Plot a histogram to visualize directors with high ROI in the musical genre

In [105]:

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_directors_musical.index, ROI_domestic_directors_m
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Musical Directors by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Director')
ax1.set_title('Top 5 Musical Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1200, 200)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_directors_musical.index, ROI_global_directors_music
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Musical Directors by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Director')
ax2.set_title('Top 5 Global Musical Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1200, 200)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 205ms, finished 14:01:02 2022-06-16



Now we will explore the top five directors for music films

In [106]:
```
# from the directors list, group a list by only directors who have wor
category_type_music = ['Music']
directors_music = directors_groupby[directors_groupby['genres_split'].
```
executed in 10ms, finished 14:01:21 2022-06-16

In [107]:
```
# find the average budget for music films
directors_music_production_mean = round(directors_music.groupby('prima
                                        mean(), 0)
```
executed in 3ms, finished 14:01:22 2022-06-16

In [108]:
```
# find the average domestic gross sales for music films
directors_music_domestic_gross_mean = round(directors_music.groupby('p
                                            .mean(), 0)
```
executed in 3ms, finished 14:01:22 2022-06-16

In [109]:
```
# find the average global gross sales for music films
directors_music_global_gross_mean = round(directors_music.groupby('pri
                                            .mean(), 0)
```
executed in 10ms, finished 14:01:23 2022-06-16

In [110]:
```
# calculate the ROI for music films by director, only display the top
ROI_domestic_directors_music = round(((directors_music_domestic_gross_
                                      directors_music_production_mean) * 10
```
executed in 11ms, finished 14:01:23 2022-06-16

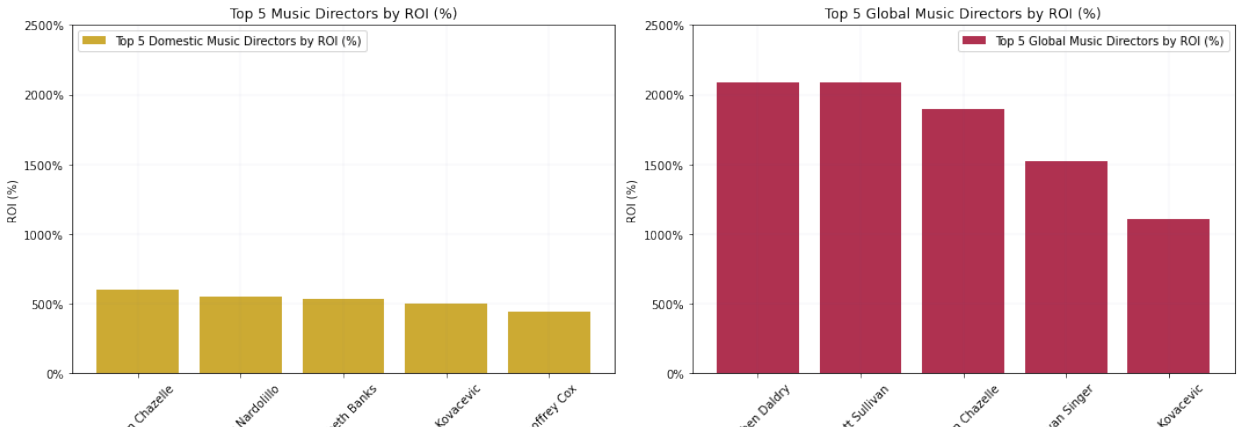
In [111]:
```
 calculate the ROI for music films by director, only display the top 5
OI_global_directors_music = round(((directors_music_global_gross_mean -
                                    directors_music_production_mean) *
```
executed in 10ms, finished 14:01:26 2022-06-16

Plot a histogram to visualize directors with high ROI in the music genre

In [112]:
```

```

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_directors_music.index, ROI_domestic_directors_mus
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Music Directors by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Director')
ax1.set_title('Top 5 Music Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 3000, 500)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_directors_music.index, ROI_global_directors_music.v
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Music Directors by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Director')
ax2.set_title('Top 5 Global Music Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 3000, 500)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 184ms, finished 14:01:56 2022-06-16



Now we will explore the top five directors for animation films

In [113]:
```
# from the directors list, group a list by only directors who have wor
category_type_animation= ['Animation']
directors_animation = directors_groupby[directors_groupby['genres_spli
```
executed in 13ms, finished 14:02:03 2022-06-16

In [114]:
```
# find the average budget for animation films
directors_animation_production_mean = round(directors_animation.groupb
                                    mean(), 0)
```
executed in 3ms, finished 14:02:03 2022-06-16

In [115]:
```
# find the average domestic gross sales for animation films
directors_animation_domestic_gross_mean = round(directors_animation.gr
                                        .mean(), 0)
```
executed in 3ms, finished 14:02:04 2022-06-16

In [116]:
```
# find the average global gross sales for animation films
directors_animation_global_gross_mean = round(directors_animation.grou
                                        .mean(), 0)
```
executed in 3ms, finished 14:02:04 2022-06-16

In [117]:
```
# calculate the ROI for animation films by director, only display the
ROI_domestic_directors_animation = round(((directors_animation_domesti
                                   directors_animation_production_mean)
```
executed in 3ms, finished 14:02:05 2022-06-16

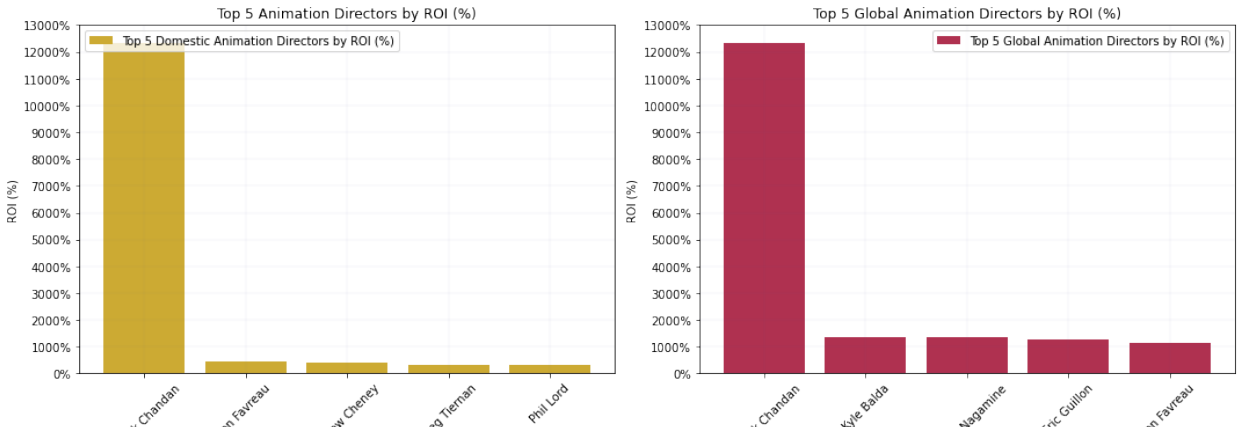
In [118]:
```
# calculate the ROI for animation films by director, only display the
ROI_global_directors_animation = round(((directors_animation_global_gr
                                   directors_animation_production_mean)
```
executed in 8ms, finished 14:02:06 2022-06-16

Plot a histogram to visualize directors with high ROI in the animation genre

In [119]:

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_directors_animation.index, ROI_domestic_directors
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Animation Directors by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Director')
ax1.set_title('Top 5 Animation Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 14000, 1000)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_directors_animation.index, ROI_global_directors_ani
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Animation Directors by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Director')
ax2.set_title('Top 5 Global Animation Directors by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 14000, 1000)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 226ms, finished 14:02:26 2022-06-16

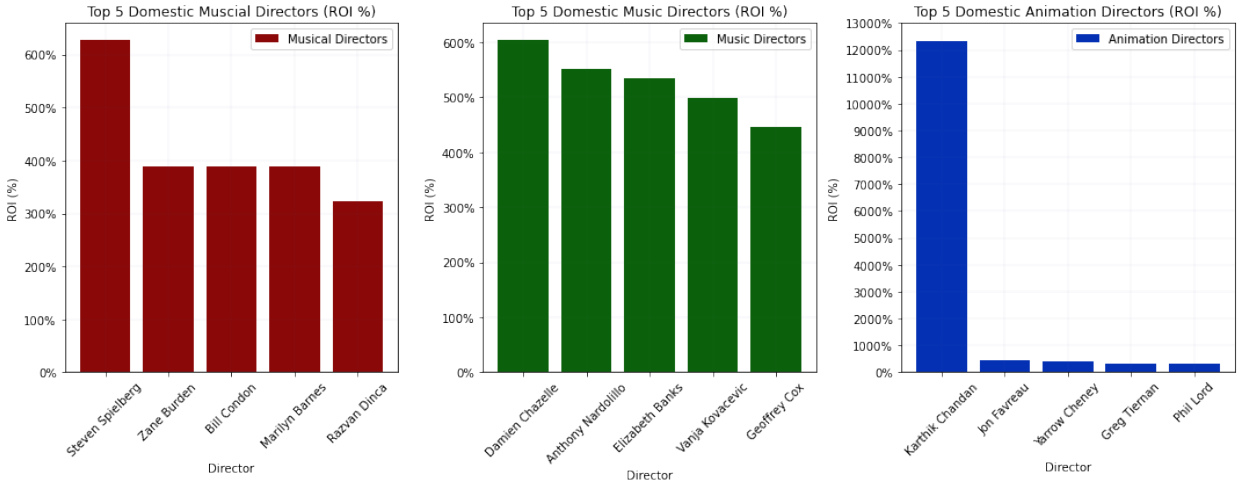


For ease of viewing, we will plot a comprehensive histogram to visualize directors with high ROI by genre (domestic)

In [120]:
```
# create the plot with 2 figures side by side
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(15, 6), ncols = 3)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_directors_musical.index, ROI_domestic_directors_m
       color = '#8A0808',
       align='center',
       label = 'Musical Directors')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Director')
ax1.set_title('Top 5 Domestic Muscial Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 700, 100)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper right')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_domestic_directors_music.index, ROI_domestic_directors_mus
       color = '#0B610B',
       align='center',
       label = 'Music Directors')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Director')
ax2.set_title('Top 5 Domestic Music Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 700, 100)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# set the plot, format, and labels for the second bar graph
ax3.bar(ROI_domestic_directors_animation.index, ROI_domestic_directors
       color = '#0431B4',
       align='center',
       label = 'Animation Directors')
ax3.set_ylabel('ROI (%)')
ax3.set_xlabel('Director')
ax3.set_title('Top 5 Domestic Animation Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax3.yaxis.set_major_formatter(xticks)
```

```
ax3.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 14000, 1000)
ax3.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax3.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax3.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 381ms, finished 14:03:58 2022-06-16

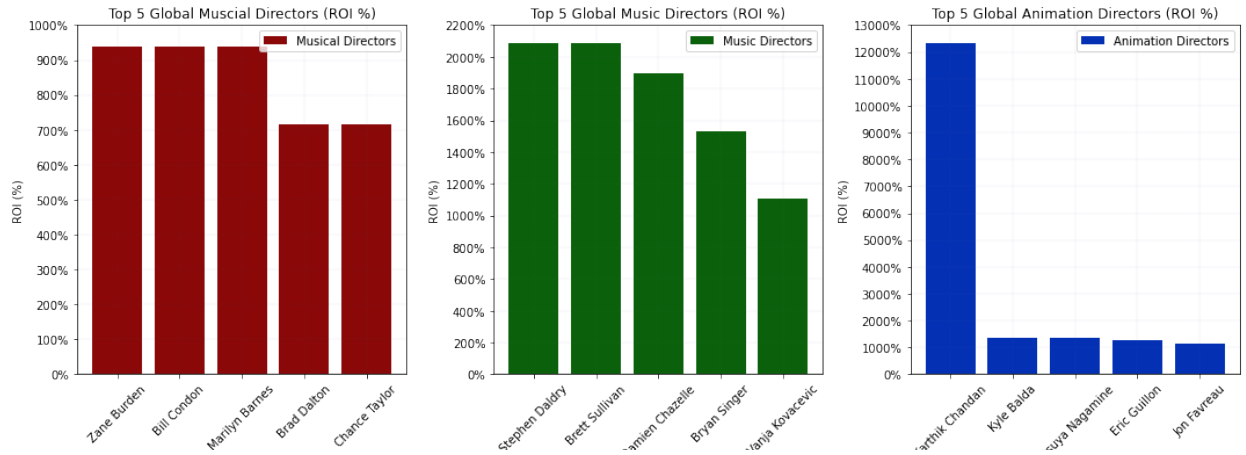


For ease of viewing, we will plot a comprehensive histogram to visualize directors with high ROI by genre (global)

In [121]:
```
# create the plot with 2 figures side by side
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(15, 6), ncols = 3)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_global_directors_musical.index, ROI_global_directors_music
       color = '#8A0808',
       align='center',
       label = 'Musical Directors')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Director')
ax1.set_title('Top 5 Global Muscial Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1100, 100)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper right')
```

```
# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_directors_music.index, ROI_global_directors_music.v
       color = '#0B610B',
       align='center',
       label = 'Music Directors')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Director')
ax2.set_title('Top 5 Global Music Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 2400, 200)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# set the plot, format, and labels for the second bar graph
ax3.bar(ROI_global_directors_animation.index, ROI_global_directors_ani
       color = '#0431B4',
       align='center',
       label = 'Animation Directors')
ax3.set_ylabel('ROI (%)')
ax3.set_xlabel('Director')
ax3.set_title('Top 5 Global Animation Directors (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax3.yaxis.set_major_formatter(xticks)
ax3.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 14000, 1000)
ax3.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax3.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax3.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 468ms, finished 14:04:03 2022-06-16



The above 2 visualizations provide insight into the top directors for the Musical, Music, and Animation genres based on domestic and global ROI. These individuals have proven to be quite prolific in their field and would be beneficial to align with for future projects.

In particular, Zane Burden, Bill Condon, and Marilyn Barnes have over a 900% ROI for Musical films, Stephen Daldry and Brett Sullivan have over a 2000% ROI for music films, and Kathik Candan has over a 12000% ROI for Animation films. Partnership with these directors could be quite beneficial to explore.

Writers

We will replicate the process used above to analyze directors as we look to better understand which writers to partner with.

In [122]:
```
# remove writers that have passed away
alive_writers = full_movie_details.loc[full_movie_details['death_year'
alive_writers.head()
```

executed in 81ms, finished 14:04:48 2022-06-16

Out[122]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	2009-12-18	Avatar	425000000	760507625	2776345279
1	1	2009-12-18	Avatar	425000000	760507625	2776345279
2	1	2009-12-18	Avatar	425000000	760507625	2776345279
3	1	2009-12-18	Avatar	425000000	760507625	2776345279
4	1	2009-12-18	Avatar	425000000	760507625	2776345279

In [123]:
```
# from the full movie details dataset sort by writers
category_type_writer = ['writer']

writers_groupby =  alive_writers[alive_writers['category'].isin(categc
writers_groupby.head()
```

executed in 34ms, finished 14:04:55 2022-06-16

Out[123]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1	1	2009-12-18	Avatar	425000000	760507625	277634527
9	1	2009-12-18	Avatar	425000000	760507625	277634527
13	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387
14	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387
17	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	104566387

In [124]:
```
# find the average budget by writer
writers_production_mean = round(writers_groupby.groupby('primary_name'
                                    mean(), 0)
```

executed in 18ms, finished 14:05:03 2022-06-16

In [125]:
```
# find the average domestic gross sales by writer
writers_domestic_gross_mean = round(writers_groupby.groupby('primary_n
                                      .mean(), 0)
```

executed in 17ms, finished 14:05:04 2022-06-16

In [126]:
```
# find the average global gross sales by writer
writers_global_mean = round(writers_groupby.groupby('primary_name')['w
                                    mean(), 0)
```

executed in 6ms, finished 14:05:04 2022-06-16

In [127]:
```
# calculate the ROI by writer (domestic), only display the top 25
ROI_domestic_writers = round(((writers_domestic_gross_mean - writers_p
                                writers_production_mean) * 100, 2).so
```

executed in 12ms, finished 14:05:07 2022-06-16

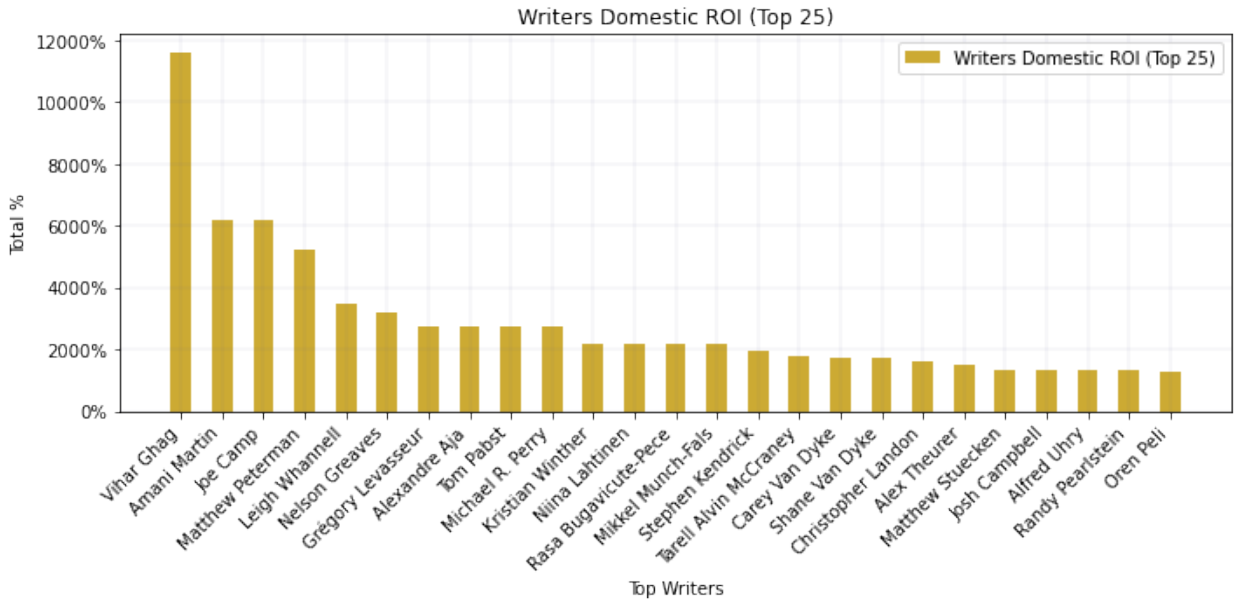
In [128]:
```
# calculate the ROI by writer (global), only display the top 25
ROI_global_writers = round(((writers_global_mean - writers_production_
                              writers_production_mean) * 100, 2).sort
```

executed in 14ms, finished 14:05:11 2022-06-16

Visualize the ROI of top writers (domestic)

In [129]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_domestic_writers.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_domestic_writers.values,
             width = 0.5,
             color = '#CCAA33',
             label='Writers Domestic ROI (Top 25)')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Writers')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Writers Domestic ROI (Top 25)')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

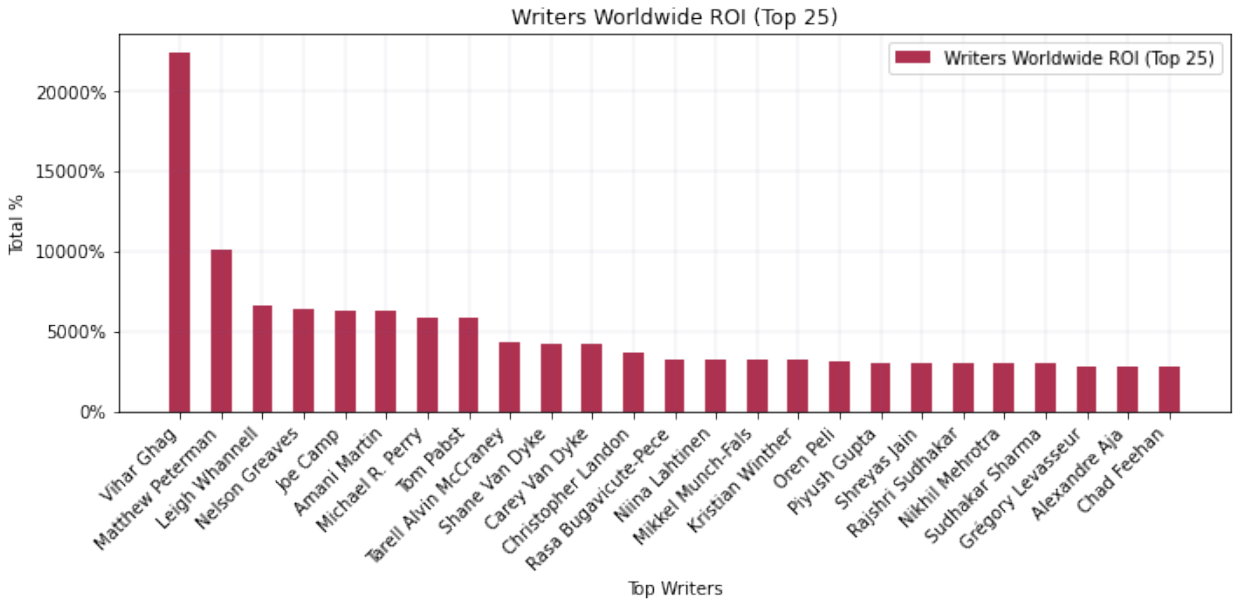
executed in 206ms, finished 14:05:12 2022-06-16



Visualize the ROI of top writers (global)

In [130]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_global_writers.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_global_writers.values,
             width = 0.5,
             color = '#AF3150',
             label='Writers Worldwide ROI (Top 25)')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Writers')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Writers Worldwide ROI (Top 25)')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 206ms, finished 14:05:17 2022-06-16



Above we were able to identify a list of Writers who, regardless of genre, have proven themselves to be profitable. Note that this approach does not take into account the number of films a writer has been involved with. Vihar Ghag has only 1 writer credit to his name, though his film has returned well. (cite: https://www.imdb.com/name/nm3358805/ (https://www.imdb.com/name/nm3358805/))

Below we will work to find the genres with the highest ROI when taking writer into account.

In [131]:
```
# find the average budget by genre when taking into account writers
writers_genre_production_mean = round(writers_groupby.groupby('genres_
                                      mean(), 0)
```
executed in 12ms, finished 14:05:28 2022-06-16

In [132]:
```
# find the average domestic gross sales by genre when taking into acco
writers_genre_domestic_gross_mean = round(writers_groupby.groupby('ger
                                        .mean(), 0)
```
executed in 11ms, finished 14:05:29 2022-06-16

In [133]:
```
# find the average global gross sales by genre when taking into accoun
writers_genre_global_gross_mean = round(writers_groupby.groupby('genre
                                    mean(), 0)
```
executed in 4ms, finished 14:05:30 2022-06-16

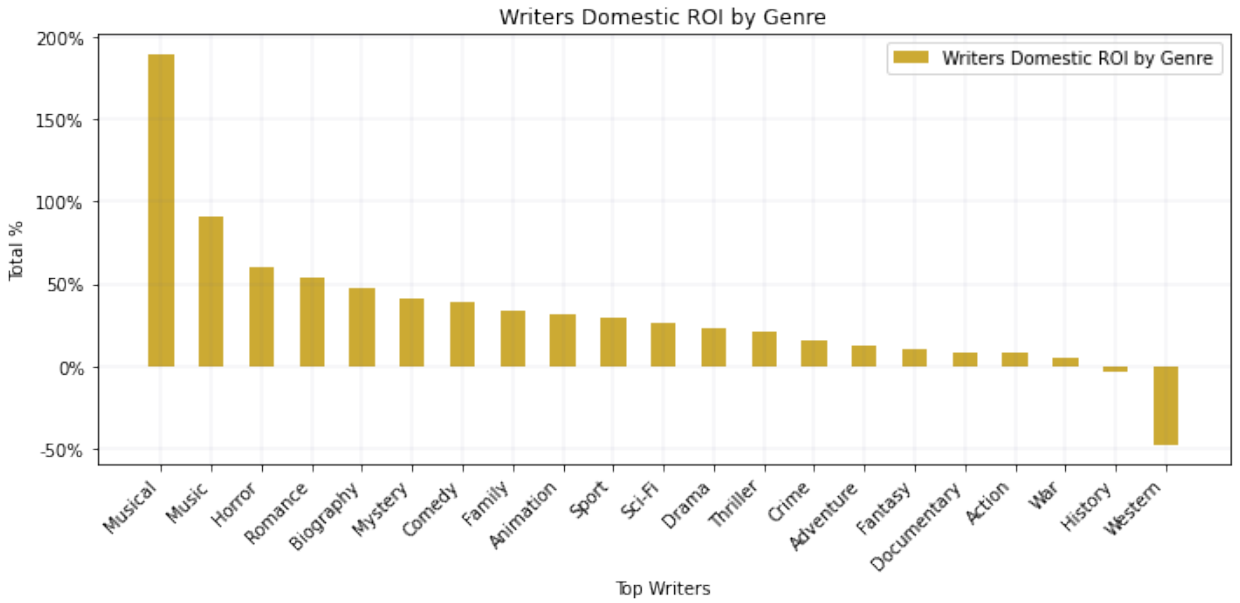
In [134]:
```
# calculate the ROI by genre (domestic) when taking into account write
ROI_domestic_writers_genre = round(((writers_genre_domestic_gross_mean
                                    writers_genre_production_mean) * 100,
```
executed in 4ms, finished 14:05:31 2022-06-16

In [135]:
```
# calculate the ROI by genre (global) when taking into account writers
ROI_global_writers_genre = round(((writers_genre_global_gross_mean - w
                                      writers_genre_production_mean) *
```
executed in 8ms, finished 14:05:37 2022-06-16

Visualize the ROI of genres when taking into account writers (domestic)

In [136]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_domestic_writers_genre.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_domestic_writers_genre.values,
             width = 0.5,
             color = '#CCAA33',
             label='Writers Domestic ROI by Genre')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Writers')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Writers Domestic ROI by Genre')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

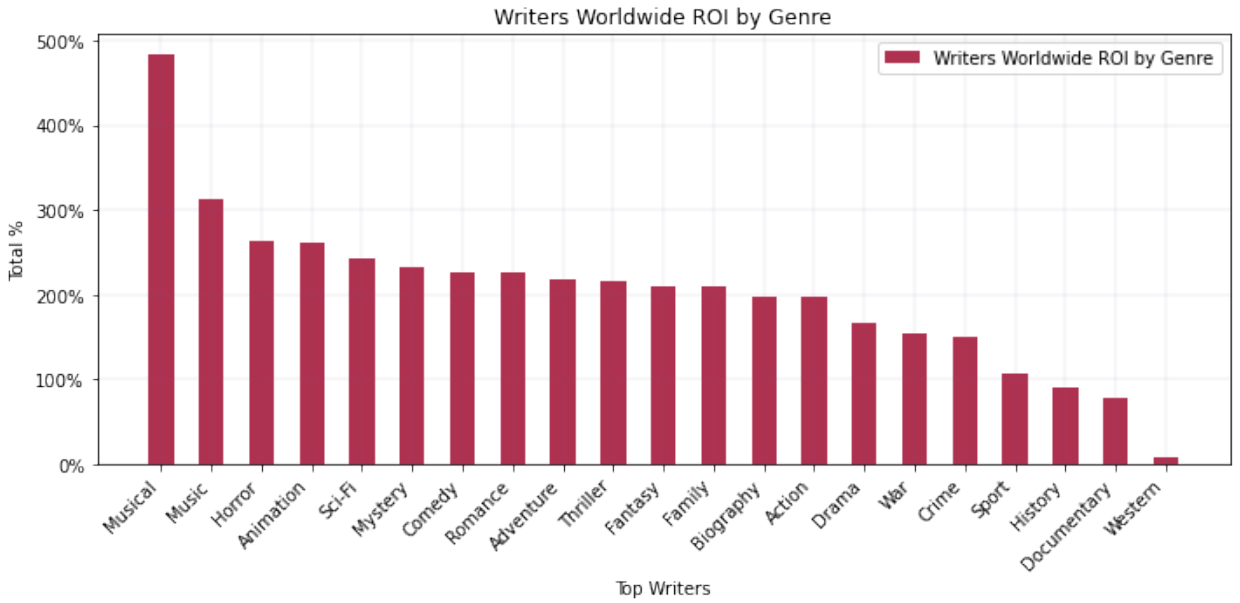
executed in 186ms, finished 14:05:42 2022-06-16



Visualize the ROI of genres when taking into account writers (global)

In [137]:
```
# establish x values for the visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = ROI_global_writers_genre.index
x = np.arange(len(labels))

# set the plot, format, and labels
bar_width = 0.3
fig, ax = plt.subplots(figsize=(10,5))
ax1 = ax.bar(x,
             ROI_global_writers_genre.values,
             width = 0.5,
             color = '#AF3150',
             label='Writers Worldwide ROI by Genre')

ax.set_xticks(x)
ax.set_xticklabels(labels)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Top Writers')
ax.set_ylabel('Total %')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)

# Add title and legends
ax.set_title('Writers Worldwide ROI by Genre')
ax.legend()

# format the layout and display the visualization
fig.tight_layout()
plt.show()
```

executed in 178ms, finished 14:05:45 2022-06-16



From the above visualizations we can see that when we take into account writers and look at the profitability of movies by genre Musical and Music films prevail with the highest ROI. Domestically, Horror movies ranks third, globally, Animation and horror movies rank third. For the duration of this analysis we will continue to look at Musical, Music, and Animation type films.

We will explore the top five writers for the top three genres below. First we will look at musical.

In [138]:
```
# from the writers list, group a list by only writers who have worked
category_type_musical = ['Musical']
writers_musical = writers_groupby[writers_groupby['genres_split'].isin
```
executed in 10ms, finished 14:06:51 2022-06-16

In [139]:
```
# find the average budget for musical films
writers_musical_production_mean = round(writers_musical.groupby('prima
                                        mean(), 0)
```
executed in 4ms, finished 14:06:51 2022-06-16

In [140]:
```
# find the average domestic gross sales for musical films
writers_musical_domestic_gross_mean = round(writers_musical.groupby('p
                                            .mean(), 0)
```
executed in 3ms, finished 14:06:52 2022-06-16

In [141]:
```
# find the average global gross sales for musical films
writers_musical_global_gross_mean = round(writers_musical.groupby('pri
                                            .mean(), 0)
```
executed in 3ms, finished 14:06:52 2022-06-16

In [142]:
```
# calculate the ROI for musical films by writer, only display the top
ROI_domestic_writers_musical = round(((writers_musical_domestic_gross_
                                      writers_musical_production_mean) * 10
```
executed in 4ms, finished 14:06:53 2022-06-16

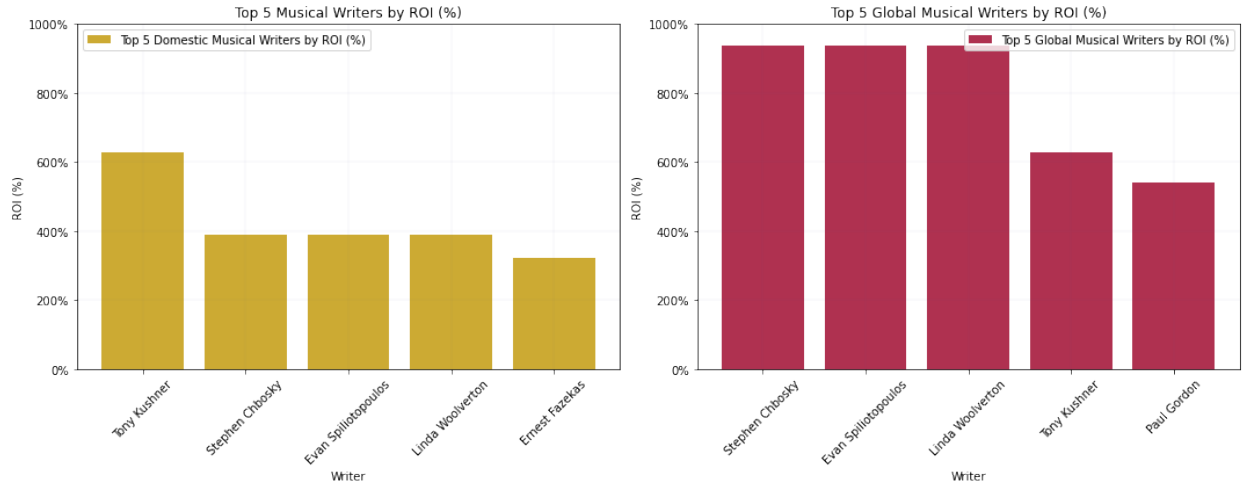
In [143]:
```
# calculate the ROI for musical films by writer, only display the top
ROI_global_writers_musical = round(((writers_musical_global_gross_mean
                                        writers_musical_production_mean)
```
executed in 7ms, finished 14:06:54 2022-06-16

plot a histogram to visualize writers with high ROI in the musical genre

In [144]:
```
```

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_writers_musical.index, ROI_domestic_writers_music
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Musical Writers by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Writer')
ax1.set_title('Top 5 Musical Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
major_ticks = np.arange(0, 1200, 200)
ax1.set_yticks(major_ticks)
ax1.tick_params(axis='x', labelrotation=45)
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_writers_musical.index, ROI_global_writers_musical.v
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Musical Writers by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Writer')
ax2.set_title('Top 5 Global Musical Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
major_ticks = np.arange(0, 1200, 200)
ax2.set_yticks(major_ticks)
ax2.tick_params(axis='x', labelrotation=45)
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 192ms, finished 14:07:13 2022-06-16



Now we will explore the top five Writers for music films

In [145]:
```
# from the writers list, group a list by only writers who have worked
category_type_music = ['Music']
writers_music = writers_groupby[writers_groupby['genres_split'].isin(c
```
executed in 10ms, finished 14:07:16 2022-06-16

In [146]:
```
# find the average budget for music films
writers_music_production_mean = round(writers_music.groupby('primary_n
                                      mean(), 0)
```
executed in 3ms, finished 14:07:17 2022-06-16

In [147]:
```
# find the average domestic gross sales for music films
writers_music_domestic_gross_mean = round(writers_music.groupby('prima
                                          .mean(), 0)
```
executed in 3ms, finished 14:07:17 2022-06-16

In [148]:
```
# find the average global gross sales for music films
writers_music_global_gross_mean = round(writers_music.groupby('primary
                                        .mean(), 0)
```
executed in 3ms, finished 14:07:18 2022-06-16

In [149]:
```
# calculate the ROI for music films by writer, only display the top 5
ROI_domestic_writers_music = round(((writers_music_domestic_gross_mean
                                    writers_music_production_mean) * 100,
```
executed in 3ms, finished 14:07:18 2022-06-16

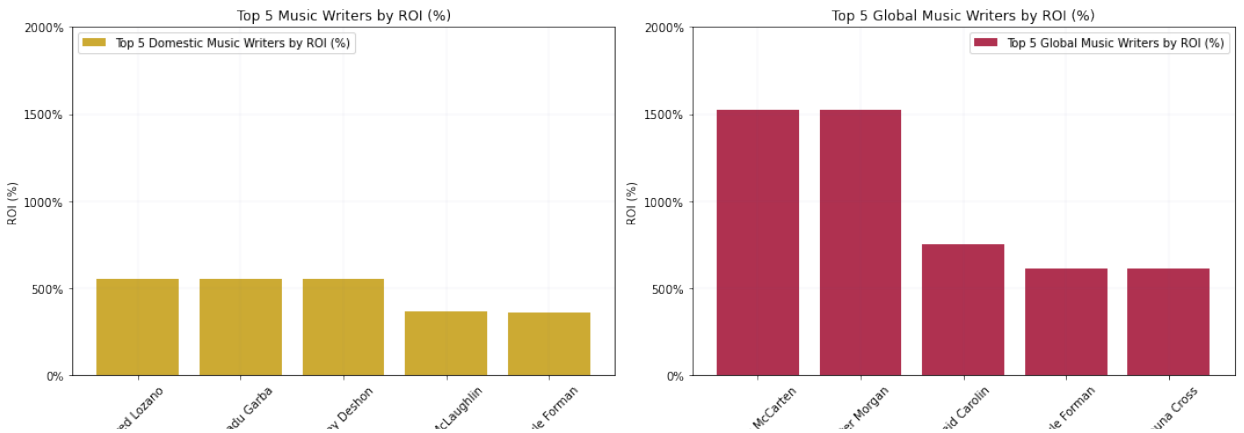
In [150]:
```
# calculate the ROI for music films by writer, only display the top 5
ROI_global_writers_music = round(((writers_music_global_gross_mean - w
                                  writers_music_production_mean) *
```
executed in 10ms, finished 14:07:19 2022-06-16

Plot a histogram to visualize writers with high ROI in the music genre

In [151]:
```

```

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_writers_music.index, ROI_domestic_writers_music.v
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Music Writers by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Writer')
ax1.set_title('Top 5 Music Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 2500, 500)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_writers_music.index, ROI_global_writers_music.value
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Music Writers by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Writer')
ax2.set_title('Top 5 Global Music Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 2500, 500)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 191ms, finished 14:07:20 2022-06-16



Now we will explore the top five directors for animation films

In [152]:
```
# from the writers list, group a list by only writers who have worked
category_type_animation= ['Animation']
writers_animation = writers_groupby[writers_groupby['genres_split'].is
```
executed in 10ms, finished 14:07:50 2022-06-16

In [153]:
```
# find the average budget for animation films
writers_animation_production_mean = round(writers_animation.groupby('p
                                    mean(), 0)
```
executed in 3ms, finished 14:07:50 2022-06-16

In [154]:
```
# find the average domestic gross sales for animation films
writers_animation_domestic_gross_mean = round(writers_animation.groupb
                                        .mean(), 0)
```
executed in 10ms, finished 14:07:51 2022-06-16

In [155]:
```
# find the average global gross sales for animation films
writers_animation_global_gross_mean = round(writers_animation.groupby(
                                        .mean(), 0)
```
executed in 3ms, finished 14:07:52 2022-06-16

In [156]:
```
# calculate the ROI for animation films by writer, only display the to
ROI_domestic_writers_animation = round(((writers_animation_domestic_gr
                                 writers_animation_production_mean) * 1
```
executed in 3ms, finished 14:07:52 2022-06-16

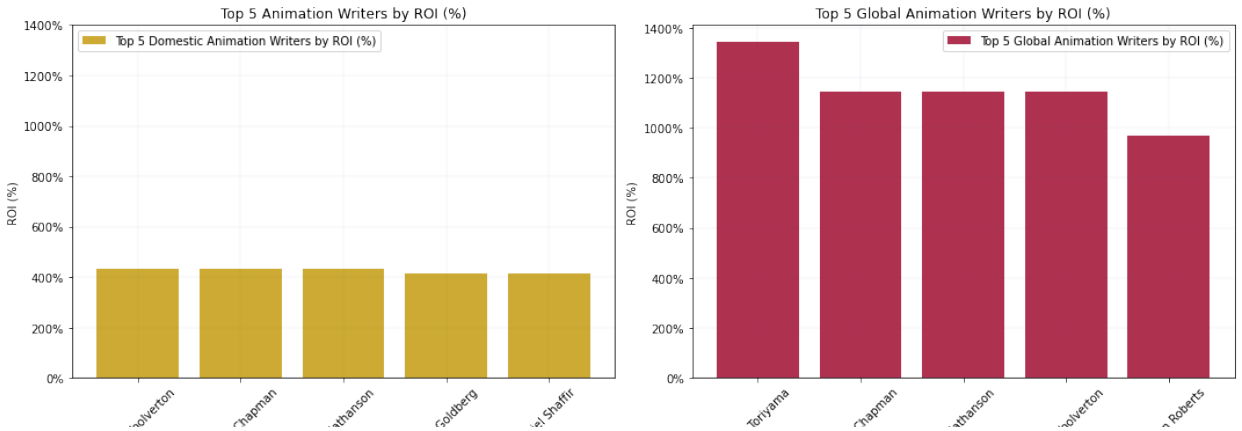
In [157]:
```
# calculate the ROI for animation films by writer, only display the to
ROI_global_writers_animation = round(((writers_animation_global_gross_
                                  writers_animation_production_mean) *
```
executed in 11ms, finished 14:07:57 2022-06-16

Plot a histogram to visualize writers with high ROI in the animation genre

In [158]:
```

```

```
# create the plot with 2 figures side by side
fig, (ax1, ax2) = plt.subplots(figsize=(15, 6), ncols = 2)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_writers_animation.index, ROI_domestic_writers_ani
       color = '#CCAA33',
       align='center',
       label = 'Top 5 Domestic Animation Writers by ROI (%)')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Writer')
ax1.set_title('Top 5 Animation Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1600, 200)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper left')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_writers_animation.index, ROI_global_writers_animati
       color = '#AF3150',
       align='center',
       label = 'Top 5 Global Animation Writers by ROI (%)')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Writer')
ax2.set_title('Top 5 Global Animation Writers by ROI (%)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1600, 200)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()
```

executed in 215ms, finished 14:08:14 2022-06-16

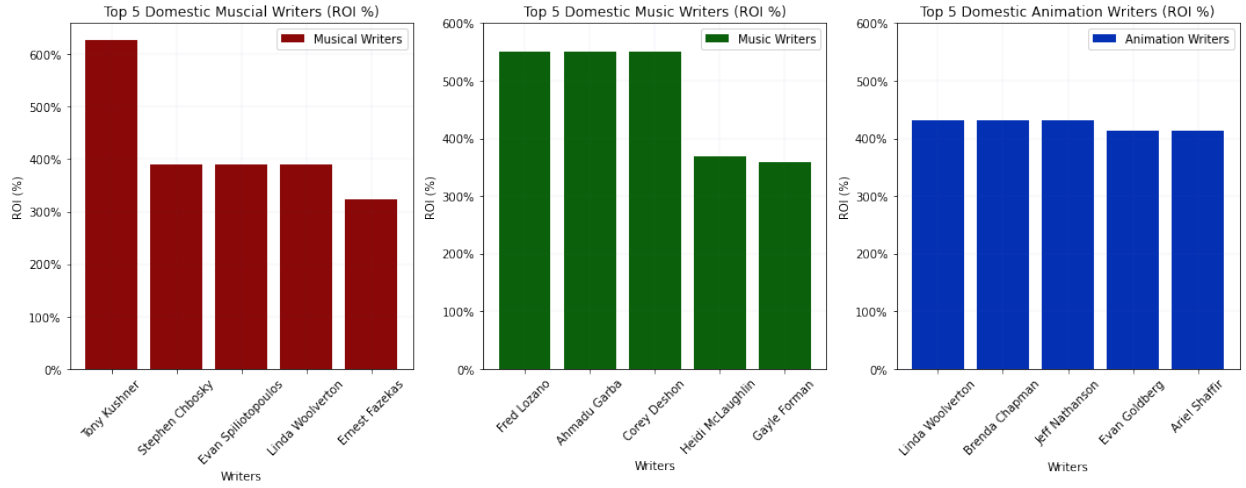


Below is a visualization that provides the top writers for the Musical, Music, and Animation genres based on domestic ROI.

In [159]:
```
# plot a histogram to visualize writers with high ROI by genre (domest

# create the plot with 2 figures side by side
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(15, 6), ncols = 3)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_domestic_writers_musical.index, ROI_domestic_writers_music
       color = '#8A0808',
       align='center',
       label = 'Musical Writers')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Writers')
ax1.set_title('Top 5 Domestic Muscial Writers (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 700, 100)
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper right')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_domestic_writers_music.index, ROI_domestic_writers_music.v
       color = '#0B610B',
       align='center',
       label = 'Music Writers')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Writers')
ax2.set_title('Top 5 Domestic Music Writers (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 700, 100)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# set the plot, format, and labels for the second bar graph
ax3.bar(ROI_domestic_writers_animation.index, ROI_domestic_writers_ani
       color = '#0431B4',
       align='center',
       label = 'Animation Writers')
ax3.set_ylabel('ROI (%)')
ax3.set_xlabel('Writers')
ax3.set_title('Top 5 Domestic Animation Writers (ROI %)')
```

```
xticks = mtick.FormatStrFormatter('%.0f%%')
ax3.yaxis.set_major_formatter(xticks)
ax3.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 700, 100)
ax3.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax3.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax3.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 371ms, finished 14:08:27 2022-06-16

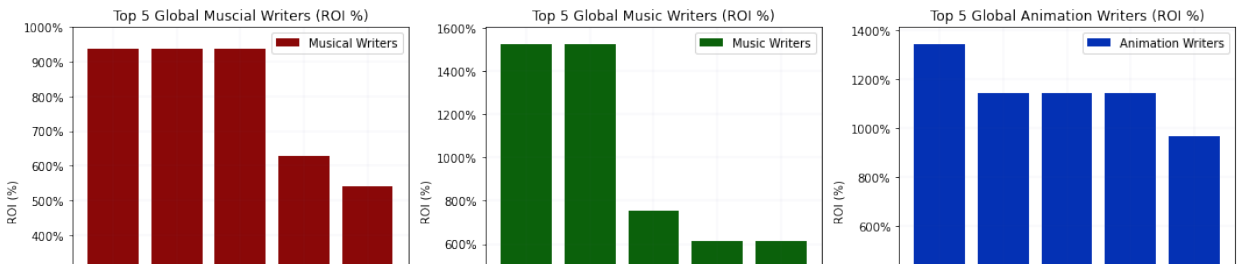


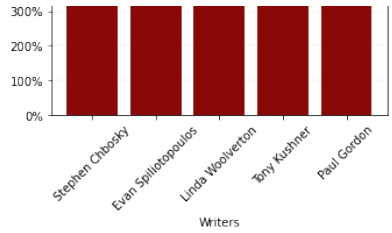
Below is a visualization that provides the top writers for the Musical, Music, and Animation genre based on global ROI. This is the information that is used to further provide recommendations.

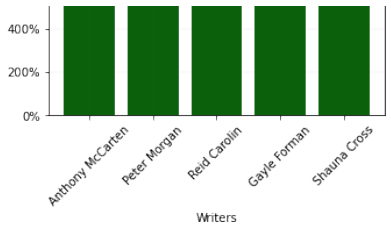
In [160]:
```
# plot a histogram to visualize writers with high ROI by genre (global

# create the plot with 2 figures side by side
fig, (ax1, ax2, ax3) = plt.subplots(figsize=(15, 6), ncols = 3)

# set the plot, format, and labels for the first bar graph
ax1.bar(ROI_global_writers_musical.index, ROI_global_writers_musical.v
       color = '#8A0808',
       align='center',
       label = 'Musical Writers')
ax1.set_ylabel('ROI (%)')
ax1.set_xlabel('Writers')
ax1.set_title('Top 5 Global Muscial Writers (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax1.yaxis.set_major_formatter(xticks)
ax1.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1100, 100)
```

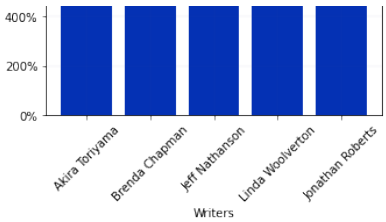
```
ax1.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax1.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax1.legend(loc='upper right')

# set the plot, format, and labels for the second bar graph
ax2.bar(ROI_global_writers_music.index, ROI_global_writers_music.value
       color = '#0B610B',
       align='center',
       label = 'Music Writers')
ax2.set_ylabel('ROI (%)')
ax2.set_xlabel('Writers')
ax2.set_title('Top 5 Global Music Writers (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax2.yaxis.set_major_formatter(xticks)
ax2.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1700, 200)
ax2.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax2.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax2.legend(loc = 'upper right')

# set the plot, format, and labels for the second bar graph
ax3.bar(ROI_global_writers_animation.index, ROI_global_writers_animati
       color = '#0431B4',
       align='center',
       label = 'Animation Writers')
ax3.set_ylabel('ROI (%)')
ax3.set_xlabel('Writers')
ax3.set_title('Top 5 Global Animation Writers (ROI %)')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax3.yaxis.set_major_formatter(xticks)
ax3.tick_params(axis='x', labelrotation=45)
major_ticks = np.arange(0, 1600, 200)
ax3.set_yticks(major_ticks)
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
ax3.grid(color='#666699', alpha = .5, linestyle='-', linewidth=0.1)
ax3.legend(loc = 'upper right')

# format the layout and display the graphs
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 372ms, finished 14:08:35 2022-06-16









The above 2 visualizations provide insight into the top writers for the Musical, Music, and Animation genres based on domestic and global ROI. These individuals have proven to be quite prolific in their field and would be beneficial to align with for future projects.

In particular, Stephen Chbosky, Evan Spiliotopoulos, and Linda Wollverton have over a 900% ROI for Musical films, Anthony McCarten and Peter Morgan have over a 1500% ROI for music films, and Akira Toriyama has over a 13000% ROI for Animation films. Partnership with these Writers could be quite beneficial to explore.

Recommendation

In an effort to mitigate risk, and possibly increase the likelihood of future profitability through return on investment, this analysis looked into which writers and directors in the most profitable genres had a portfolio that was also quite profitable. As you can see in the graphical representations above within the Musical, Music, and Animation genres, the top writers and directors have return on investments of incredible measure. Aligning with these partners who have a proven track record for success could lend itself to great market entry.

As Microsoft explores their approach towards entering the movie industry they should consider working with the following directors.

- Musical
  - Zane Burden
  - Bill Condon
  - Marilyn Barnes
- Music
  - Stephen Daldry
  - Brett Sullivan
  - Damien Chazelle
- Animation
  - Karthik Chandan
  - Kyle Balda
  - Tatsuya Nagamine

As Microsoft explores their approach towards entering the movie industry they should consider working with the following writers.

- Musical
  - Stephen Chbosky
  - Evan Spiliotopoulos
  - Linda Woolverton
- Music
  - Anthony McCarten
  - Peter Morgan
  - Reid Carolin
- Animation
  - Akira Toriyama
  - Brenda Chapman
  - Jeff Nathanson

Business Recommendation 3: Strategic Planning for Profitable Release Date

It is important to intimately understand the profitability of movies based on the timing of their release. In the below analysis we will look at movie release dates by month to see what time of year is the most profitable.

In [161]:
```
# look at the full movie details dataset to remind what information is
full_movie_details.info()
```

executed in 57ms, finished 14:09:04 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69644 entries, 0 to 69672
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 69644 non-null  int64         
 1   release_date       69644 non-null  datetime64[ns]
 2   movie              69644 non-null  object        
 3   production_budget  69644 non-null  int64         
 4   domestic_gross     69644 non-null  int64         
 5   worldwide_gross    69644 non-null  int64         
 6   genres             69644 non-null  object        
 7   genres_split       69644 non-null  object        
 8   category           69644 non-null  object        
 9   primary_name       69644 non-null  object        
 10  death_year         1421 non-null   float64       
 11  month              69644 non-null  int64         
 12  day                69644 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(6), object(5)
memory usage: 7.4+ MB
```

In [162]:
```
# calculate return on investment by month (domestic)
full_movie_details['domestic_roi'] = ((full_movie_details['domestic_gr
                                      - full_movie_details['productior
```

executed in 11ms, finished 14:09:13 2022-06-16

In [163]:
```
# calculate return on investment by month (global)
full_movie_details['global_roi'] = ((full_movie_details['worldwide_gro
                                    - full_movie_details['productior
```

executed in 8ms, finished 14:09:15 2022-06-16

In [164]:
```
# check to ensure the domestic and global ROI columns have been added
full_movie_details.head()
```

executed in 27ms, finished 14:09:22 2022-06-16

Out[164]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	2009-12-18	Avatar	425000000	760507625	2776345279
1	1	2009-12-18	Avatar	425000000	760507625	2776345279
2	1	2009-12-18	Avatar	425000000	760507625	2776345279
3	1	2009-12-18	Avatar	425000000	760507625	2776345279
4	1	2009-12-18	Avatar	425000000	760507625	2776345279

In [165]:
```
# drop unused columns for easier sorting
movie_details_temp = full_movie_details.drop(columns = ['genres', 'ger
                     'primary_name'], inplace=True )
```

executed in 19ms, finished 14:09:34 2022-06-16

Exploring Profitability by Month

In [166]:
```
# look at the fulll movie details list to ensure those columns have be
full_movie_details.info()
```

executed in 26ms, finished 14:10:10 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 69644 entries, 0 to 69672
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 69644 non-null  int64
 1   release_date       69644 non-null  datetime64[ns]
 2   movie              69644 non-null  object
 3   production_budget  69644 non-null  int64
 4   domestic_gross     69644 non-null  int64
 5   worldwide_gross    69644 non-null  int64
 6   death_year         1421 non-null   float64
 7   month              69644 non-null  int64
 8   day                69644 non-null  int64
 9   domestic_roi       69644 non-null  float64
 10  global_roi         69644 non-null  float64
dtypes: datetime64[ns](1), float64(3), int64(6), object(1)
memory usage: 6.4+ MB
```

In [167]:
```
# drop duplicates in full_movie_details, since genre_split can have mo
full_movie_details_unique = full_movie_details.drop_duplicates()
```

executed in 36ms, finished 14:10:29 2022-06-16

In [168]:
```
# look at the fulll movie details list to ensure those rows have been
full_movie_details_unique.info()
```

executed in 17ms, finished 14:10:32 2022-06-16

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2910 entries, 0 to 69658
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   id                 2910 non-null   int64         
 1   release_date       2910 non-null   datetime64[ns]
 2   movie              2910 non-null   object        
 3   production_budget  2910 non-null   int64         
 4   domestic_gross     2910 non-null   int64         
 5   worldwide_gross    2910 non-null   int64         
 6   death_year         548 non-null    float64       
 7   month              2910 non-null   int64         
 8   day                2910 non-null   int64         
 9   domestic_roi       2910 non-null   float64       
 10  global_roi         2910 non-null   float64       
dtypes: datetime64[ns](1), float64(3), int64(6), object(1)
memory usage: 272.8+ KB
```

In [169]:
```
# find the average ROI by month (domestic)
month_ROI_domestic = full_movie_details_unique.groupby('month')['domes
month_ROI_domestic
```

executed in 11ms, finished 14:10:35 2022-06-16

Out[169]:
```
month
1     152.717233
2     201.157949
3      37.308967
4      80.753734
5      86.415330
6      79.994468
7     185.562611
8     108.834335
9      27.331331
10    227.305671
11    155.105782
12     44.739185
Name: domestic_roi, dtype: float64
```

In [170]:
```
# find the average ROI by month (global)
month_ROI_global = full_movie_details_unique.groupby('month')['global_
month_ROI_global
```

executed in 13ms, finished 14:10:45 2022-06-16

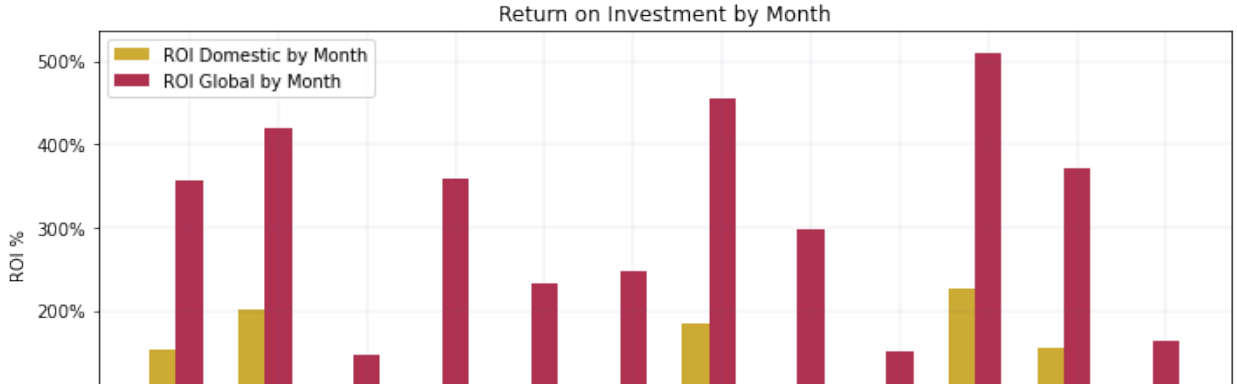
Out[170]:
```
month
1     357.001752
2     420.289962
3     147.577041
4     358.119700
5     233.775751
6     246.701981
7     455.389762
8     298.559062
9     151.981236
10    510.736383
11    370.732280
12    163.846499
Name: global_roi, dtype: float64
```

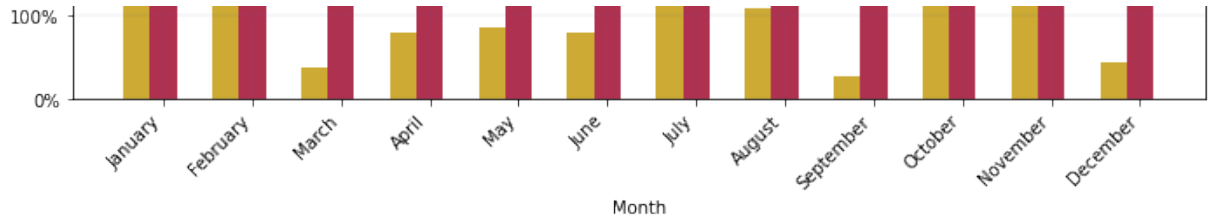
Visualize the ROI by month (domestic)

In [171]:

```
# establish x values for a multi dataset visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = 'January', 'February', 'March', 'April', 'May', 'June', 'July
x = np.arange(len(labels))

# set the plot, format, and labels
fig, ax = plt.subplots(figsize=(10,5))
bar_width = 0.3
ax1 = ax.bar(x - bar_width / 2,
             month_ROI_domestic.values,
             width = 0.3,
             color = '#CCAA33',
             label ='ROI Domestic by Month')

ax2 = ax.bar(x + bar_width / 2,
             month_ROI_global.values,
             width = 0.3,
             color = '#AF3150',
             label='ROI Global by Month')

# Add a legend to the plot
ax.tick_params(axis='x')
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Month')
plt.setp(ax.get_xticklabels(), rotation=45, ha='right') # cite: https:
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(labels)
ax.set_ylabel('ROI %')

# Add title and legends
ax.legend()
ax.set_title('Return on Investment by Month')

# format the layout and display the visualization
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 216ms, finished 14:10:57 2022-06-16





The visualization above helps us understand with greater awareness that October proves to have the highest potential for return on investment. In total, October boasts an ROI of 511%.

Exploring Profitability by Days in October

As we can see from the analysis above, that October has prevailed as the most profitable month to release a film domestically and globally. We will explore the specific dates within October to better understand details about the month of October's profitability.

In [172]:
```
# from the full movie details dataset sort by October
category_type_october = ['10']

october = full_movie_details_unique[full_movie_details_unique['month']
october.head()
```

executed in 25ms, finished 14:11:23 2022-06-16

Out[172]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
1986	80	2017-10-06	Blade Runner 2049	185000000	92054159	2593574(
1987	80	2017-10-06	Blade Runner 2049	185000000	92054159	2593574(
4425	93	2015-10-09	Pan	150000000	35088320	15152597
4426	93	2015-10-09	Pan	150000000	35088320	15152597
5984	46	2011-10-28	Puss in Boots	130000000	149260504	55498747

In [173]:
```
# find the average ROI for each day of the month of October (domestic)
october_by_day_domestic = october.groupby('day')['domestic_roi'].mean(
october_by_day_domestic
```

executed in 14ms, finished 14:11:27 2022-06-16

```
14     -10.839401
15     127.728465
16      -8.272882
17    2048.548101
18     418.317694
19     522.286615
20     404.157395
21     285.271733
22     177.708750
23     -62.710916
24      94.767867
25     -84.354986
26     -46.502395
27     107.277796
28     221.932015
29     -55.803575
30     102.688961
31     113.352617
Name: domestic_roi, dtype: float64
```

In [174]:
```
# find the average ROI for each day of the month of October (global)
october_by_day_global = october.groupby('day')['global_roi'].mean()
october_by_day_global
```

executed in 15ms, finished 14:11:29 2022-06-16

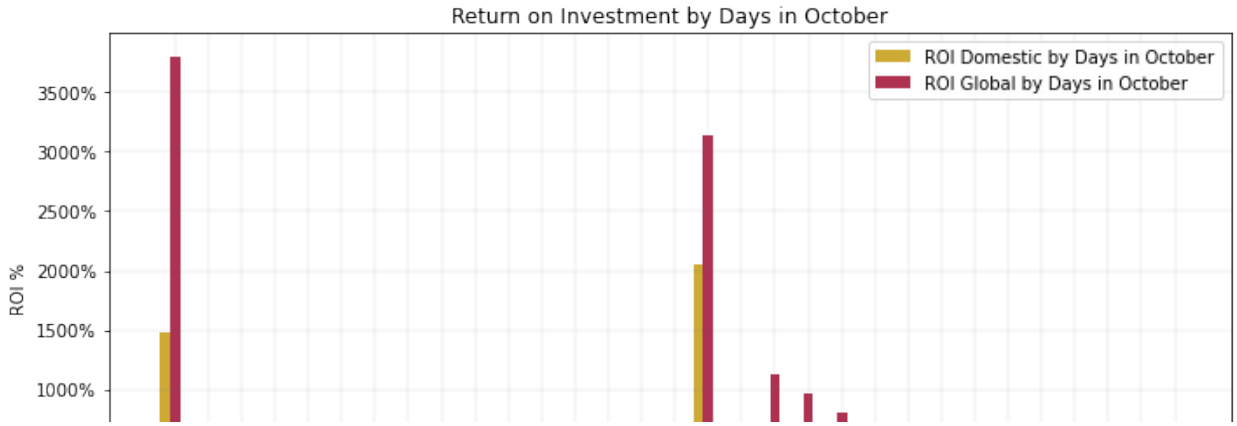
Out[174]:
```
day
1     3803.751868
2       96.104987
3      304.011031
4       69.159219
5      196.750610
6      117.044771
7      155.896810
8       35.419224
9        0.052388
10     178.288660
11     118.235038
12     573.959139
13     460.615907
14      18.877220
15     241.901944
16      80.136377
17    3145.389788
18     677.477485
```

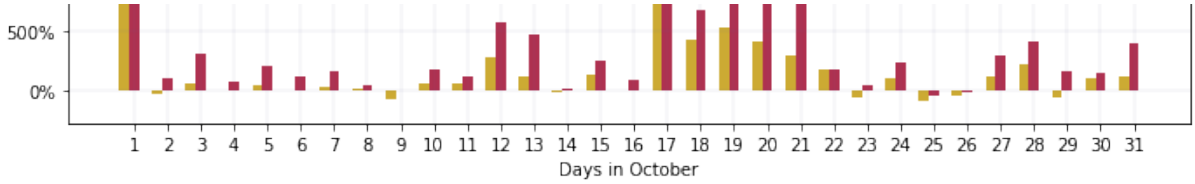
Visualize the ROI by day in October

In [175]:

```
# establish x values for a multi dataset visualization
# cite: https://matplotlib.org/stable/gallery/lines_bars_and_markers/b
labels = october_by_day_domestic.index
x = np.arange(len(labels))

# set the plot, format, and labels
fig, ax = plt.subplots(figsize=(10,5))
bar_width = 0.3
ax1 = ax.bar(x - bar_width / 2,
             october_by_day_domestic.values,
             width = 0.3,
             color = '#CCAA33',
             label ='ROI Domestic by Days in October')

ax2 = ax.bar(x + bar_width / 2,
             october_by_day_global.values,
             width = 0.3,
             color = '#AF3150',
             label='ROI Global by Days in October')

# Add a legend to the plot
ax.tick_params(axis='x')
ax.grid(color='#666699', linestyle='-', linewidth=0.1)
ax.set_xlabel('Days in October')
xticks = mtick.FormatStrFormatter('%.0f%%')
ax.yaxis.set_major_formatter(xticks)
ax.set_xticks(x + bar_width / 2)
ax.set_xticklabels(labels)
ax.set_ylabel('ROI %')

# Add title and legends
ax.legend()
ax.set_title('Return on Investment by Days in October')

# format the layout and display the visualization
fig.tight_layout()
plt.show()

# save figure to files
fig.savefig('/Users/laurenbrown/Documents/Flatiron/Phase_1/Project/Pha
```

executed in 348ms, finished 14:11:59 2022-06-16





Recommendation

Another aspect of the successful launch of Microsoft's entry into the movie industry is to understand, with confidence, when to release a film. As can be expected, there are seasonal impacts to a film's success. This analysis worked to understand what time of year is more ideal than others to release a film. As represented above, we now understand that the most profitable month that a film can be released, in terms of average return on investment, is in October. In total, October boasts an ROI of 511%.

As we learned from further analysis, of all days in October, the 1st day of the month has proved to be the most profitable day to release a film. In total, October 1st boasts an ROI of 3801%.

Conclusion

As shared in the detailed analysis above, there are 3 strategic recommendations for potentially profitable entry into the movie industry:

1. Pursue creating movies that fall into the genres of greatest profitability: Musical, Music, and Animation
2. Collaborate with highly profitable writers and directors within the profitable genres outlined in recommendation 1.
3. Release films during the most profitable time of year, which is October. In particular October 1 is the most profitable day to release a film.