1 Final Project Submission: Microsoft Movie Studios Viability Analysis

(Phase 1)

- Student Name: Tenicka Norwood
- Program Pace: self paced
- Scheduled Project Review time: 11/1/2022
- Instructor name: Joe Comeaux
- Blog post Url: https://medium.com/@t.terell.norwood/why-did-i-choose-data-science-3783379cc338 (https://medium.com/@t.terell.norwood/why-did-i-choose-data-science-3783379cc338)

Photo by: Crownlab on Canva (http://www.canva.com)

2 Business Understanding

Analysts at Business Wire estimate that the global film and video market will reach $410.6 billion by 2030. Microsoft is uniquely situated to leverage its existing technology holidings to redefine the film industry by crafting a one stop shop platform which manages the entire process from preproduction to filming to distribution. Microsoft's executives are in search of actionable ways to ensure successful movies are produced as they launch a new movie studio that is well supported from its onset.

As a new recruit for the newly formed performance business analysis team at Microsoft, I have been tasked with crafting a plan to make the company's decision to diversify their holdings to include creating original video content at their new movie studio that is both well supported with data and profitable. The guiding question that I have been tasked to answer through data analysis is: Which types of films historically are the most successful at the box office?

How do we prevent headlines like:

> ***'Fantastic Beasts' turns 'The Secrets of Dumbledore' into too much of a snore*** - Review by Brian Lowry, CNN April 15, 2022

From negatively impacting our bottom line?

With this in mind, I am working on the following questions:

- When is the best time of year to release a movie?
- Which director makes the most profitable movies?
- Which genres of movies make the most profit at the box office?

3 Data Understanding

The datasets used in this project are from the following sources:

- [IMDB (https://www.imdb.com/)](https://www.imdb.com/)
- [TheMovieDB (https://www.themoviedb.org/)](https://www.themoviedb.org/)
- [Box Office Mojo (https://www.boxofficemojo.com/)](https://www.boxofficemojo.com/)
- [The Numbers (https://www.the-numbers.com/)](https://www.the-numbers.com/)

There is a variety of information available on the web that can help as I try to identify the tools Microsoft will need to leverage to be a viable contender in the original video content space and compete as a new Movie studio within their brand. This data includes information on movie genres, titles, runtimes, production costs, gross box office revenue both foreign and domestic and release dates. With this data I will drill down for insights related to the impact that time of year, performer and genres have on success of movies at the box office.

4 Method

This project will explore data related to current trends in the movie industry. This exploration will include:

- **Importing Relevant libraries and packages**
  - Access code from different modules
- **Data Preparation**
  - Access databases and dataframes useful to this project
  - Investigate data shape and datatype information
  - Drop or impute null values
  - Reduce complexity (ex. join dataframes where necessary, remove or replace missing values, address duplicates data)
- **Addressing Question 1 through Exploratory Data Analysis (EDA)**
  - Build or extract features from cleaned data
  - Make visualizations
  - Analyze correlations
  - Summarize findings
- **Addression Question 2 through EDA**
  - Build or extract features from cleaned data
  - Make visualizations
  - Analyze correlations
  - Summarize findings
- **Addressing Question 3 through EDA**
  - Build or extract features from cleaned data
  - Make visualizations
  - Analyze correlations
  - Summarize findings
- **Discussion & Recommendations**
  - Discuss findings
  - Identify next steps based on findings

5 Data Preparation

5.1 Import libraries and Visualization Packages

Importing libraries at the beginning allows access to modules and other tools throughout this project that help to make the tasks within this project manageable to implement. The main libraries that will be used within this project include:

- `sqlite3` : a library that provides a SQL interface that allows accessing and manipulating SQL database
- `pandas` : a data analysis and manipulation library which allows for flexible reading, writing, and reshaping of data
- `numpy` : a key library that brings the computationaly power of languages like C to Python
- `matplotlib` : a comprehensive visualization library
- `seaborn` : a data visualization library based on matplotlib

In [1]:

```
# Import libraries and visualization packages
import sqlite3
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
# Allow plots to display and be stored inline within a notebook
%matplotlib inline

# Set display option to readable format
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

In [2]:

```
# Create a connection to the imdb database
conn = sqlite3.connect('zippedData/im.db/im.db')
```

5.2 Accessing databases and dataframes

Review data shape and statistics. IMDB data is in a database and has multiple tables which have a column called movie_id that allows them to be combined. These tables include genres, primary titles, and runtimes, but do not contain financial information. Data from The Numbers and Box Office Mojo contain financial information.

In [3]:

```
# Read in the data
# Read Data from Box office Mojo
bom_df = pd.read_csv('zippedData/bom.movie_gross.csv.gz')

# Select All from Movie Basics
movie_basics = pd.read_sql("SELECT * FROM movie_basics;", conn)

#Explore data from Movie Ratings Table by Selecting all fields
movie_ratings = pd.read_sql("SELECT * FROM movie_ratings;", conn)
#movie_ratings.head()

# Select All from Directors
movie_directors = pd.read_sql("SELECT * FROM directors", conn)

# Select ALL from Persons
movie_persons = pd.read_sql("SELECT * FROM persons", conn)

# Read Data from The Movie Databases
tmdb_movies_df = pd.read_csv('zippedData/tmdb.movies.csv.gz')

# Read in movie budget data from The Numbers
tn_movie_budgets_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz')
```

In [4]:

```
def preview_dataframe(df):
    '''
    Preview information about the shape, datatypes and size of the dataframe

    Will show the first 5 rows of a dataframe and information about
    the type of each column in the data frame

    -
    Input:

    df: Pandas dataframe
    -

    Output: Pandas dataframe, showing shape, information and statistics
    '''

    preview = df.head()
    preview_info = df.info()
    preview_shape = pd.Series(df.shape)


    return preview
```

In [5]:

```
preview_dataframe(bom_df)
```

```
<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
```

Out[5]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000	2010
3	Inception	WB	292600000.00	535700000	2010
4	Shrek Forever After	P/DW	238700000.00	513900000	2010

In [6]:

```
bom_df.head()
```

Out[6]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000	2010
3	Inception	WB	292600000.00	535700000	2010
4	Shrek Forever After	P/DW	238700000.00	513900000	2010

In [7]:

```
# Preview Box Office Mojo Dataframe
bom_df.head()
```

Out[7]:

	title	studio	domestic_gross	foreign_gross	year
0	Toy Story 3	BV	415000000.00	652000000	2010
1	Alice in Wonderland (2010)	BV	334200000.00	691300000	2010
2	Harry Potter and the Deathly Hallows Part 1	WB	296000000.00	664300000	2010
3	Inception	WB	292600000.00	535700000	2010
4	Shrek Forever After	P/DW	238700000.00	513900000	2010

In [8]:

```
# Information about the shape, datatypes and size of bom_df dataframe
bom_df.info()
```

```
<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
```

In [9]:

```
# Remove Future Projects From IMDB Movie Basics
movie_basics = movie_basics[movie_basics['start_year'] < 2022]
```

In [10]:

```
# Display First Five Rows of the movie_basics table for Movies with Release dates prior to this year
movie_basics.head()
```

Out[10]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.00	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.00	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.00	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	nan	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.00	Comedy,Drama,Fantasy

In [11]:

```
# Information about the shape, datatypes and size of movie_basics dataframe
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146101 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146101 non-null  object 
 1   primary_title    146101 non-null  object 
 2   original_title   146080 non-null  object 
 3   start_year       146101 non-null  int64  
 4   runtime_minutes  114402 non-null  float64
 5   genres           140705 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

In [12]:
```python
# Preview IMDB directors Table
movie_directors.head()
```

Out[12]:

	movie_id	person_id
0	tt0285252	nm0899854
1	tt0462036	nm1940585
2	tt0835418	nm0151540
3	tt0835418	nm0151540
4	tt0878654	nm0089502

In [13]:
```python
# Information about the shape, datatypes and size of movie_directors dataframe
movie_directors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 291174 entries, 0 to 291173
Data columns (total 2 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   movie_id   291174 non-null  object
 1   person_id  291174 non-null  object
dtypes: object(2)
memory usage: 4.4+ MB
```

In [14]:
```python
# Address missing data replacing null values with placeholder
movie_persons.head()
movie_persons["primary_profession"].fillna("Unknown", inplace = True)
movie_persons.info()
```

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

In [15]:
```python
# Drop columns birth_year and death_year
movie_persons.drop(labels =["birth_year", "death_year"], axis = 1, inplace = True)
movie_persons.head()
```

Out[15]:

	person_id	primary_name	primary_profession
0	nm0061671	Mary Ellen Bauder	miscellaneous,production_manager,producer
1	nm0061865	Joseph Bauer	composer,music_department,sound_department
2	nm0062070	Bruce Baum	miscellaneous,actor,writer
3	nm0062195	Axel Baumann	camera_department,cinematographer,art_department
4	nm0062798	Pete Baxter	production_designer,art_department,set_decorator

In [16]:

```
# Merge directors and persons dataframes
director_names = pd.merge(left = movie_directors, right = movie_persons, left_on = ["person_id"],
                          right_on = ["person_id"], how = "inner")
director_names.head()
```

Out[16]:

	movie_id	person_id	primary_name	primary_profession
0	tt0285252	nm0899854	Tony Vitale	producer,director,writer
1	tt0462036	nm1940585	Bill Haley	director,writer,producer
2	tt0835418	nm0151540	Jay Chandrasekhar	director,actor,writer
3	tt0835418	nm0151540	Jay Chandrasekhar	director,actor,writer
4	tt0859635	nm0151540	Jay Chandrasekhar	director,actor,writer

In [17]:

```
# Information about the shape, datatypes and size of movie_basics dataframe
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146101 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146101 non-null  object 
 1   primary_title    146101 non-null  object 
 2   original_title   146080 non-null  object 
 3   start_year       146101 non-null  int64  
 4   runtime_minutes  114402 non-null  float64
 5   genres           140705 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

In [18]:

```
# Information about the shape, datatypes and size of director_name dataframe
director_names.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 291171 entries, 0 to 291170
Data columns (total 4 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   movie_id            291171 non-null  object
 1   person_id           291171 non-null  object
 2   primary_name        291171 non-null  object
 3   primary_profession  291171 non-null  object
dtypes: object(4)
memory usage: 11.1+ MB
```

In [19]:

```python
# Preview The Movie Database Movies Dataframe
tmdb_movies_df.head()
```

Out[19]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	title	vote_average	vote_coun
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.53	2010-11-19	Harry Potter and the Deathly Hallows: Part 1	7.70	1078
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.73	2010-03-26	How to Train Your Dragon	7.70	761
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.52	2010-05-07	Iron Man 2	6.80	1236
3	3	[16, 35, 10751]	862	en	Toy Story	28.00	1995-11-22	Toy Story	7.90	1017
4	4	[28, 878, 12]	27205	en	Inception	27.92	2010-07-16	Inception	8.30	2218

In [20]:

```python
# Information about the shape, datatypes and size of tmdb_movies_df dataframe
tmdb_movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26517 entries, 0 to 26516
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Unnamed: 0         26517 non-null  int64  
 1   genre_ids          26517 non-null  object 
 2   id                 26517 non-null  int64  
 3   original_language  26517 non-null  object 
 4   original_title     26517 non-null  object 
 5   popularity         26517 non-null  float64
 6   release_date       26517 non-null  object 
 7   title              26517 non-null  object 
 8   vote_average       26517 non-null  float64
 9   vote_count         26517 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 2.0+ MB
```

While there are 146101 rows and 6 columns in the movie basics data. I notice that **three of those rows have missing values**:

- `genres`
- `runtime_minutes`
- `original_title`

In [21]:

```python
# Remove Unnamed column
tmdb_movies_df.drop(labels = "Unnamed: 0", axis = 1, inplace = True)
```

In [22]:
```
# Print the number of missing values for each column
print("The number of missing rows in the genres column of the movie_basics dataframe is:",
         len(movie_basics[movie_basics["genres"].isna()]),'.')
print("The number of missing rows in the runtime_minutes column of the movie_basics dataframe is",
         len(movie_basics[movie_basics["runtime_minutes"].isna()]),'.')
print("The number of missing rows in the original_title colum of the movie_basics dataframe is",
         len(movie_basics[movie_basics["original_title"].isna()]),'.')
```

```
The number of missing rows in the genres column of the movie_basics dataframe is: 5396 .
The number of missing rows in the runtime_minutes column of the movie_basics dataframe is 31699 .
The number of missing rows in the original_title colum of the movie_basics dataframe is 21 .
```

Since **there are only 21 (~ 0.01 % of column data) rows of missing data** in the `original_title` column, we can delete these rows without skewing our results.

In [23]:
```
# Address Missing values in movie_basics original_title column
movie_basics.dropna(subset =["original_title"], inplace = True)
```

In [24]:
```
# Check the shape of the data
movie_basics.shape
```

Out[24]:
```
(146080, 6)
```

Now that I deleted the rows that were missing `original_title` data, I will check to see if the features have changed by using .info().

In [25]:
```
# Review movie _basics info to see if overall columns have changed
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146080 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146080 non-null  object 
 1   primary_title    146080 non-null  object 
 2   original_title   146080 non-null  object 
 3   start_year       146080 non-null  int64  
 4   runtime_minutes  114398 non-null  float64
 5   genres           140703 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

There are 31699 rows of missing data in the `runtime_minutes` column which means that **only ~78% of movies have a known run time in minutes** so I will replace the null values with the median value of the runtime_minutes column the dataframe.

In [26]:
```
# Address Missing Row values in runtime_minutes column
movie_basics["runtime_minutes"].fillna(movie_basics["runtime_minutes"].median(), inplace = True)
```

In [27]:
```
# Review movie _basics info to see if overall columns have changed
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146080 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146080 non-null  object 
 1   primary_title    146080 non-null  object 
 2   original_title   146080 non-null  object 
 3   start_year       146080 non-null  int64  
 4   runtime_minutes  146080 non-null  float64
 5   genres           140703 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

Now the movie_basics dataframe has *5 out of 6 columns* without any missing values. This leaves the genres column which has 5408 missing rows of data. While this represents ~3.7% of the data in the column, we do not want to drop this data since it may have an affect on our results.

In [28]:
```
# Get value counts of movie genres from movie_basics dataframe
movie_basics["genres"].value_counts()
movie_basics["genres"].value_counts().head(20)
```

Out[28]:
```
Documentary                      32184
Drama                            21484
Comedy                            9177
Horror                            4371
Comedy,Drama                      3519
Thriller                          3046
Action                            2213
Biography,Documentary             2115
Drama,Romance                     2079
Comedy,Drama,Romance              1558
Documentary,Drama                 1554
Comedy,Romance                    1507
Romance                           1453
Documentary,Music                 1365
Drama,Thriller                    1335
Documentary,History               1289
Horror,Thriller                   1253
Biography,Documentary,History     1230
Biography,Documentary,Drama       1028
Family                             939
Name: genres, dtype: int64
```

In [29]:
```
# Address missing genres values with placeholder called Missing.
movie_basics["genres"].fillna("Missing", inplace = True)
```

In [30]:
```
# Review movie _basics info to see if overall columns have changed
movie_basics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 146080 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   movie_id         146080 non-null  object 
 1   primary_title    146080 non-null  object 
 2   original_title   146080 non-null  object 
 3   start_year       146080 non-null  int64  
 4   runtime_minutes  146080 non-null  float64
 5   genres           146080 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.8+ MB
```

Now the movie_basics dataframe has **6 columns of data** each with 146080 entries and **no missing data.**

There are several movies listed under two or more genres. For example there 3519 movies that are listed under both Comedy and Drama and 1028 movies listed under three genres (Biography, Documentary and Drama) for the same movie, so I will make a list and then create a set of rows for each genre a movie is listed under.

In [31]:

```
# Address movies listed under multiple genres
movie_basics["multi_genre"] = movie_basics["genres"].str.split(",")
movie_basics.head()
```

Out[31]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	multi_genre
0	tt0063540	Sunghursh	Sunghursh	2013	175.00	Action,Crime,Drama	[Action, Crime, Drama]
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.00	Biography,Drama	[Biography, Drama]
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.00	Drama	[Drama]
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	87.00	Comedy,Drama	[Comedy, Drama]
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.00	Comedy,Drama,Fantasy	[Comedy, Drama, Fantasy]

In [32]:

```
# Explode genre list into new rows
exploded_movie_basics = movie_basics.explode("multi_genre")
exploded_movie_basics.head()
```

Out[32]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	multi_genre
0	tt0063540	Sunghursh	Sunghursh	2013	175.00	Action,Crime,Drama	Action
0	tt0063540	Sunghursh	Sunghursh	2013	175.00	Action,Crime,Drama	Crime
0	tt0063540	Sunghursh	Sunghursh	2013	175.00	Action,Crime,Drama	Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.00	Biography,Drama	Biography
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.00	Biography,Drama	Drama

In [33]:

```
# Get the descriptive statistics for averagerating
movie_ratings["averagerating"].describe()
```

Out[33]:

```
count   73856.00
mean        6.33
std         1.47
min         1.00
25%         5.50
50%         6.50
75%         7.40
max        10.00
Name: averagerating, dtype: float64
```

In [34]:
```python
# Get the Runtimes of movies by genre
grouped_movies_by_genre = exploded_movie_basics.groupby("multi_genre")
rm_stats = grouped_movies_by_genre["runtime_minutes"].describe()
rm_stats
```

Out[34]:

multi_genre	count	mean	std	min	25%	50%	75%	max
Action	10321.00	97.23	33.40	2.00	87.00	90.00	108.00	2160.00
Adult	25.00	86.80	11.63	57.00	87.00	87.00	87.00	120.00
Adventure	6456.00	85.92	27.55	1.00	75.00	87.00	97.00	540.00
Animation	2793.00	81.81	24.01	1.00	73.00	87.00	90.00	360.00
Biography	8722.00	74.84	32.35	2.00	56.25	79.00	92.00	761.00
Comedy	25309.00	92.70	48.94	1.00	85.00	89.00	100.00	5460.00
Crime	6751.00	94.37	23.76	2.00	85.00	90.00	105.00	605.00
Documentary	51638.00	74.43	241.31	1.00	56.00	77.00	87.00	51420.00
Drama	49876.00	93.07	60.39	2.00	83.00	89.00	102.00	6000.00
Family	6225.00	83.57	39.98	1.00	72.00	87.00	96.00	2400.00
Fantasy	3509.00	91.12	35.53	4.00	80.00	87.00	100.00	1440.00
Game-Show	4.00	102.00	20.31	87.00	87.00	95.50	110.50	130.00
History	6225.00	79.33	55.54	1.00	59.00	81.00	94.00	2905.00
Horror	10804.00	87.28	17.87	1.00	80.00	87.00	92.00	623.00
Missing	5377.00	85.10	21.55	4.00	85.00	87.00	87.00	600.00
Music	4314.00	82.89	29.42	1.00	67.00	87.00	95.00	383.00
Musical	1430.00	93.72	37.51	4.00	80.00	87.00	105.00	808.00
Mystery	4659.00	92.54	85.41	1.00	82.00	89.00	101.00	5460.00
News	1550.00	66.76	90.74	1.00	50.00	67.50	84.00	3450.00
Reality-TV	98.00	82.86	35.81	7.00	60.25	87.00	87.00	240.00
Romance	9371.00	97.98	22.63	2.00	87.00	93.00	108.00	480.00
Sci-Fi	3363.00	89.88	33.19	4.00	81.50	87.00	96.00	1440.00
Short	11.00	22.82	24.64	1.00	8.50	16.00	24.50	87.00
Sport	2234.00	81.71	50.80	1.00	62.00	87.00	96.00	1669.00
Talk-Show	50.00	86.90	21.22	45.00	87.00	87.00	87.00	190.00
Thriller	11881.00	93.01	20.51	4.00	85.00	89.00	100.00	788.00
War	1405.00	87.12	27.45	1.00	75.00	87.00	102.00	192.00
Western	467.00	98.08	227.50	2.00	80.00	87.00	95.00	4980.00

In [35]:
```python
# Information about the shape, datatypes and size of dataframe
movie_ratings.info()
```

```
<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 [36]:

```
# Get movie_ratings statistics
movie_ratings["averagerating"].describe()
```

Out[36]:

```
count    73856.00
mean         6.33
std          1.47
min          1.00
25%          5.50
50%          6.50
75%          7.40
max         10.00
Name: averagerating, dtype: float64
```

Now I will remove movie ratings that are rated 6 or lower, since the mean and median values are **6.33** and **6.50** respectively.

In [37]:

```
# Filter movie_ratings
filtered_movie_ratings = movie_ratings[movie_ratings["averagerating"] <= 6]
filtered_movie_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27968 entries, 3 to 73853
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   movie_id       27968 non-null  object 
 1   averagerating  27968 non-null  float64
 2   numvotes       27968 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 874.0+ KB
```

While the ratings dataframe has no missing values, we need more information. So I will look at another data source [The Numbers (https://www.the-numbers.com/)](https://www.the-numbers.com/) to find more financial information.

In [38]:

```
# Preview The Numbers movie_budgets Dataframe
tn_movie_budgets_df.head()
```

Out[38]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
0	1	Dec 18, 2009	Avatar	$425,000,000	$760,507,625	$2,776,345,279
1	2	May 20, 2011	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875
2	3	Jun 7, 2019	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350
3	4	May 1, 2015	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963
4	5	Dec 15, 2017	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747

In [39]:

```
# Convert columns to data time
tn_movie_budgets_df['release_date'] = pd.to_datetime(tn_movie_budgets_df['release_date'])
tn_movie_budgets_df["release_year"] = tn_movie_budgets_df['release_date'].dt.year
tn_movie_budgets_df["release_month"] = tn_movie_budgets_df['release_date'].dt.month
tn_movie_budgets_df.drop("release_date", axis = 1, inplace = True)
```

In [40]:

```
# Review the shape and datatype information of The numbers movie_budgets dataframe
tn_movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 5782 non-null   int64
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   object
 3   domestic_gross     5782 non-null   object
 4   worldwide_gross    5782 non-null   object
 5   release_year       5782 non-null   int64
 6   release_month      5782 non-null   int64
dtypes: int64(3), object(4)
memory usage: 316.3+ KB
```

In [41]:

```
# Preview The Numbers movie_budgets Dataframe
tn_movie_budgets_df.head()
```

Out[41]:

	id	movie	production_budget	domestic_gross	worldwide_gross	release_year	release_month
0	1	Avatar	$425,000,000	$760,507,625	$2,776,345,279	2009	12
1	2	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	2011	5
2	3	Dark Phoenix	$350,000,000	$42,762,350	$149,762,350	2019	6
3	4	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	2015	5
4	5	Star Wars Ep. VIII: The Last Jedi	$317,000,000	$620,181,382	$1,316,721,747	2017	12

In [42]:

```
# Remove $ and , from Entire Dataframe
tn_movie_budgets_df = tn_movie_budgets_df.replace({"\$":""}, regex = True)
tn_movie_budgets_df = tn_movie_budgets_df.replace({"\,":""}, regex = True)
```

In [43]:

```
# Inspect first five rows of movie_budgets dataframe
tn_movie_budgets_df.head()
```

Out[43]:

	id	movie	production_budget	domestic_gross	worldwide_gross	release_year	release_month
0	1	Avatar	425000000	760507625	2776345279	2009	12
1	2	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	2011	5
2	3	Dark Phoenix	350000000	42762350	149762350	2019	6
3	4	Avengers: Age of Ultron	330600000	459005868	1403013963	2015	5
4	5	Star Wars Ep. VIII: The Last Jedi	317000000	620181382	1316721747	2017	12

In [44]:

```
# Replace production_budget, domestic_gross and worldwide_gross with float values in Millions of $
tn_movie_budgets_df["production_budget"] = pd.to_numeric(tn_movie_budgets_df["production_budget"])
tn_movie_budgets_df["domestic_gross"] = pd.to_numeric(tn_movie_budgets_df["domestic_gross"])
tn_movie_budgets_df["worldwide_gross"] = pd.to_numeric(tn_movie_budgets_df["worldwide_gross"])
tn_movie_budgets_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5782 entries, 0 to 5781
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   id                 5782 non-null   int64 
 1   movie              5782 non-null   object
 2   production_budget  5782 non-null   int64 
 3   domestic_gross     5782 non-null   int64 
 4   worldwide_gross    5782 non-null   int64 
 5   release_year       5782 non-null   int64 
 6   release_month      5782 non-null   int64 
dtypes: int64(6), object(1)
memory usage: 316.3+ KB
```

In [45]:

```
# Create a new column called worldwide_profit
tn_movie_budgets_df["worldwide_profit"] = (tn_movie_budgets_df["worldwide_gross"]-tn_movie_budgets_df

# Create a new column called ROI
tn_movie_budgets_df["ROI"] = tn_movie_budgets_df["worldwide_profit"]/tn_movie_budgets_df["production_
tn_movie_budgets_df.head()
tn_movie_budgets_df = tn_movie_budgets_df.sort_values("release_month")
tn_movie_budgets_df.head()
```

Out[45]:

	id	movie	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_profit
1414	15	Big Momma's House 2	40000000	70165972	137047376	2006	1	97047376
2812	13	Youth in Revolt	18000000	15285588	19685588	2010	1	1685588
5631	32	Solitude	200000	6260	6260	2005	1	-193740
1324	25	Shadow Conspiracy	45000000	2154540	2154540	1997	1	-42845460
2808	9	Norm of the North	18000000	17062499	30535660	2016	1	12535660

Now that I have budget and basic movie data, I will merge them with the imbd and bom dataframes and choose from the merged data frames `bom_and_tn` and `tn_and_imdb` which makes more sense for exploration once they are generated. I will look at the first five rows to check if merging was successful and replace or remove any null values from my dataframe so I can create visualizations later with the cleanest data source I can generate.

In [46]:

```
# Merge the bom and tn data into single dataframe
bom_and_tn = pd.merge(left = bom_df, right = tn_movie_budgets_df, left_on = ["title", "year"],
                         right_on = ["movie", "release_year"])

bom_and_tn.head()
```

Out[46]:

	title	studio	domestic_gross_x	foreign_gross	year	id	movie	production_budget	domestic_gross_y	worldwide_
0	Toy Story 3	BV	415000000.00	652000000	2010	47	Toy Story 3	200000000	415004880	10688
1	Inception	WB	292600000.00	535700000	2010	38	Inception	160000000	292576195	8355
2	Shrek Forever After	P/DW	238700000.00	513900000	2010	27	Shrek Forever After	165000000	238736787	7562
3	The Twilight Saga: Eclipse	Sum.	300500000.00	398000000	2010	53	The Twilight Saga: Eclipse	68000000	300531751	7061
4	Iron Man 2	Par.	312400000.00	311500000	2010	15	Iron Man 2	170000000	312433331	6211

In [47]:

```
## Merge the imdb movie_basics and movie_ratings dataframes
#imdb_df = pd.merge(left = movie_basics, right = filtered_movie_ratings, how = 'left', left_on = "mov
#                       right_on = "movie_id")
#imdb_df.head()
tn_and_imdb = pd.merge(left = tn_movie_budgets_df, right = movie_basics, left_on = ["movie", "release
                          right_on = ["primary_title", "start_year"], how = "inner")
tn_and_imdb.head()
```

Out[47]:

	id	movie	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_profit	RO
0	9	Norm of the North	18000000	17062499	30535660	2016	1	12535660	69.64
1	42	The Mechanic	42500000	29121498	76347393	2011	1	33847393	79.64
2	59	One for the Money	42000000	26414527	36197221	2012	1	-5802779	-13.8
3	64	Man on a Ledge	42000000	18620000	49621440	2012	1	7621440	18.1
4	95	Ride Along 2	40000000	90862685	124827316	2016	1	84827316	212.0

In [48]:

```python
# Replace null values for runtime_minutes with median value
tn_and_imdb["runtime_minutes"].fillna(tn_and_imdb["runtime_minutes"].median(), inplace = True)
tn_and_imdb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1532 entries, 0 to 1531
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 1532 non-null   int64  
 1   movie              1532 non-null   object 
 2   production_budget  1532 non-null   int64  
 3   domestic_gross     1532 non-null   int64  
 4   worldwide_gross    1532 non-null   int64  
 5   release_year       1532 non-null   int64  
 6   release_month      1532 non-null   int64  
 7   worldwide_profit   1532 non-null   int64  
 8   ROI                1532 non-null   float64
 9   movie_id           1532 non-null   object 
 10  primary_title      1532 non-null   object 
 11  original_title     1532 non-null   object 
 12  start_year         1532 non-null   int64  
 13  runtime_minutes    1532 non-null   float64
 14  genres             1532 non-null   object 
 15  multi_genre        1532 non-null   object 
dtypes: float64(2), int64(8), object(6)
memory usage: 203.5+ KB
```

In [49]:

```python
# Review Statistics of combined dataframe
tn_and_imdb.describe()
```

Out[49]:

	id	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_profit	R
count	1532.00	1532.00	1532.00	1532.00	1532.00	1532.00	1532.00	1532.
mean	50.59	44595784.56	55667237.65	139669856.31	2013.88	7.32	95074071.75	262.
std	28.87	56007044.40	84624290.89	233453199.34	2.57	3.39	192559324.32	1194.
min	1.00	15000.00	0.00	0.00	2010.00	1.00	-200237650.00	-100.
25%	25.75	7500000.00	2446797.50	7259289.50	2012.00	4.00	-963550.00	-36.
50%	51.00	23000000.00	26658273.50	50002036.00	2014.00	8.00	21397120.00	105.
75%	76.00	55000000.00	67268835.00	156855286.00	2016.00	10.00	103070598.75	293.
max	100.00	410600000.00	700059566.00	2048134200.00	2020.00	12.00	1748134200.00	41556.

In [50]:
```
# Sort and filter combined tn and imdb dataframe on ROI
tn_and_imdb.head()
tn_and_imdb.sort_values("worldwide_profit", ascending = False)
filtered_movie_ROI_df = tn_and_imdb[tn_and_imdb["production_budget"] >= 2000000]
filtered_movie_ROI_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1369 entries, 0 to 1531
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 1369 non-null   int64  
 1   movie              1369 non-null   object 
 2   production_budget  1369 non-null   int64  
 3   domestic_gross     1369 non-null   int64  
 4   worldwide_gross    1369 non-null   int64  
 5   release_year       1369 non-null   int64  
 6   release_month      1369 non-null   int64  
 7   worldwide_profit   1369 non-null   int64  
 8   ROI                1369 non-null   float64
 9   movie_id           1369 non-null   object 
 10  primary_title      1369 non-null   object 
 11  original_title     1369 non-null   object 
 12  start_year         1369 non-null   int64  
 13  runtime_minutes    1369 non-null   float64
 14  genres             1369 non-null   object 
 15  multi_genre        1369 non-null   object 
dtypes: float64(2), int64(8), object(6)
memory usage: 181.8+ KB
```

In [51]:
```
# Merge tn and imdb combined data frame and director_names dataframes
tn_and_imdb_full = pd.merge(left = tn_and_imdb, right = director_names, left_on = ["movie_id"],
                          right_on = ["movie_id"], how = "inner")
tn_and_imdb_full.rename(columns = {"primary_name": "director"}, inplace = True)
```

Ok. Now that I have added directors to this dataframe, I want to look at the budgets by month since it will help to answer one of my analysis questions and yield insights that would be helpful when setting up a new movie studio.

In [52]:
```
# Create a new dataframe
budgets_by_month_df = tn_movie_budgets_df.groupby("release_month").mean()

budgets_by_month_df = budgets_by_month_df.reset_index()
budgets_by_month_df = budgets_by_month_df.sort_values("release_month")

month_dict = {1: "Jan", 2: "Feb", 3: "Mar", 4: "Apr", 5: "May", 6: "Jun",
                7: "Jul", 8: "Aug", 9: "Sep", 10: "Oct", 11: "Nov", 12: "Dec"}
budgets_by_month_df["month_name"] = month_dict.values()
budgets_by_month_df["id"] = budgets_by_month_df["id"].astype(int)
```

In [53]:
```
# Find shape of budgets_by_month data frame
budgets_by_month_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12 entries, 0 to 11
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   release_month      12 non-null     int64  
 1   id                 12 non-null     int32  
 2   production_budget  12 non-null     float64
 3   domestic_gross     12 non-null     float64
 4   worldwide_gross    12 non-null     float64
 5   release_year       12 non-null     float64
 6   worldwide_profit   12 non-null     float64
 7   ROI                12 non-null     float64
 8   month_name         12 non-null     object 
dtypes: float64(6), int32(1), int64(1), object(1)
memory usage: 912.0+ bytes
```

In [54]:

```
# Find descriptive statistics of budgets_by_month data frame
budgets_by_month_df.describe()
```

Out[54]:

	release_month	id	production_budget	domestic_gross	worldwide_gross	release_year	worldwide_profit	ROI
count	12.00	12.00	12.00	12.00	12.00	12.00	12.00	12.00
mean	6.50	49.92	31655717.26	41878594.51	91582340.53	2003.99	59926623.27	386.00
std	3.61	1.00	9838690.31	16960048.48	43020339.60	0.95	33240284.76	123.38
min	1.00	48.00	20392657.94	23149886.94	46563824.02	2002.16	24880784.87	226.86
25%	3.75	49.75	23305348.38	26602172.72	57306334.36	2003.51	33834713.59	310.54
50%	6.50	50.00	29414248.88	36993819.41	76088948.47	2003.74	46674699.59	352.57
75%	9.25	50.00	42559631.64	58817889.53	137047123.84	2004.70	94460541.53	435.84
max	12.00	52.00	47135195.57	66697948.59	162268003.97	2005.54	115132808.40	688.97

Now that I have clean dataframes, let's answer the first question which is geared toward determining a general timeline for movie releases to maximize the profit centers of our new movie studio.

6 Exploratory Data Analysis Q1

6.1 When is the best time of year to release a movie?

Looking at the `filtered_movie_ROI_df` dataframe for correlations helped to answer this question. First I looked at a correlation heat map to compare each attribute's correlation and found that `ROI` column has a positve correlation of ~0.77 with `worldwide_gross` column. Similarly, the `production_budget` column has a positive correlation of ~0.65 with the `worldwide_profit` column. I used these two pieces of information along with the `budgets_by_month` dataframe to visualize the correlation between the worldwide profit and the month a film is released.

In [55]:

```
# Find correlations in filtered_movie_ROI dataframe
filtered_movie_ROI_df.corr().style.background_gradient(cmap="Blues")
```

Out[55]:

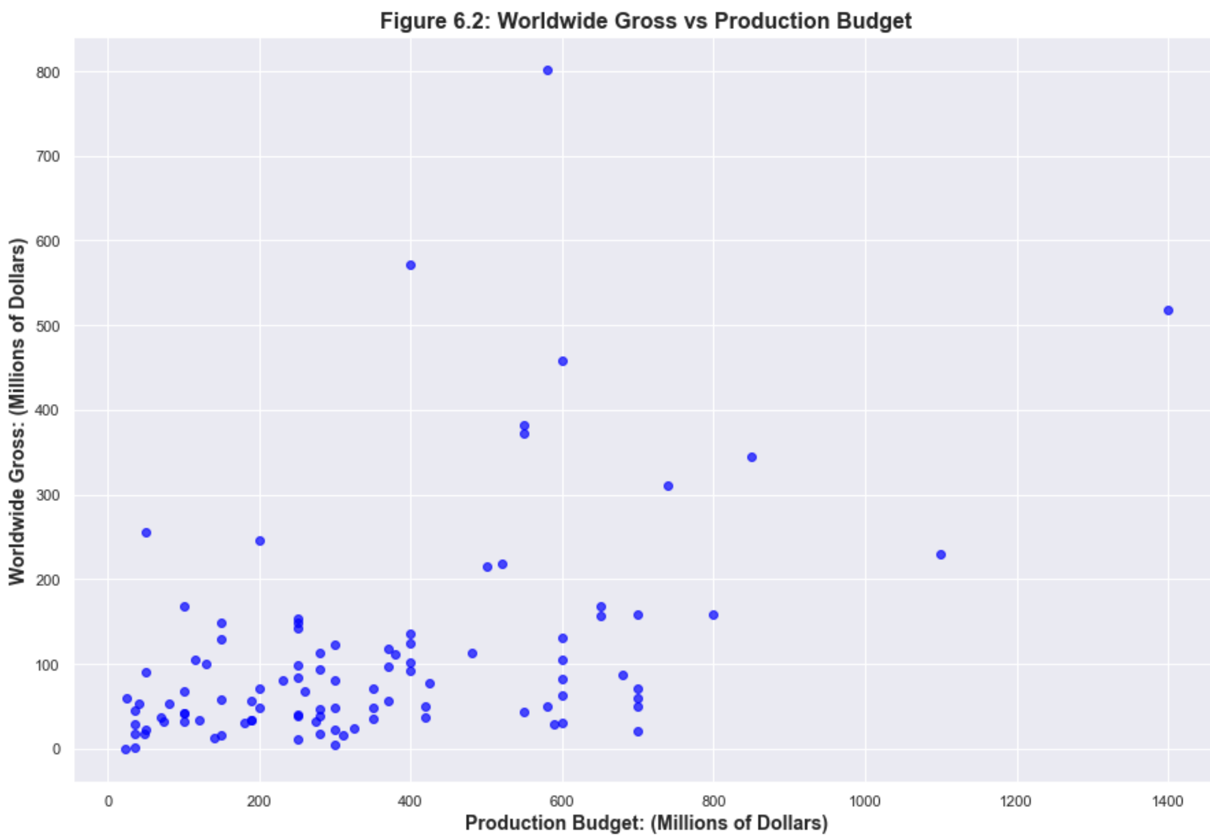
	id	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_
id	1.000000	-0.083084	-0.036307	-0.052061	0.010905	-0.023648	-0.0
production_budget	-0.083084	1.000000	0.697481	0.776006	0.049715	-0.081066	0.6
domestic_gross	-0.036307	0.697481	1.000000	0.943726	0.059030	-0.074364	0.9
worldwide_gross	-0.052061	0.776006	0.943726	1.000000	0.069500	-0.057440	0.9
release_year	0.010905	0.049715	0.059030	0.069500	1.000000	-0.053050	0.0
release_month	-0.023648	-0.081066	-0.074364	-0.057440	-0.053050	1.000000	-0.0
worldwide_profit	-0.039095	0.650468	0.938360	0.983824	0.069573	-0.046147	1.0
ROI	0.074785	-0.038209	0.314164	0.276026	0.064038	-0.029893	0.3
start_year	0.010905	0.049715	0.059030	0.069500	1.000000	-0.053050	0.0
runtime_minutes	-0.003374	0.319625	0.275829	0.291216	0.072257	0.113108	0.2

Now let's visualize the correlation between worldwide profit and a movie's production budget.

In [56]:

```
# Set up theme
sns.set_theme(style="darkgrid", palette="Set2")
data = filtered_movie_ROI_df.head(100)
x_values = np.array(data.production_budget)
y_values = data.worldwide_gross

# Set up plot
fig, ax = plt.subplots(figsize = (15, 10))
ax.scatter(x = x_values, y = "worldwide_gross", data = data, alpha=0.7, color="blue")
ax.set_title("Figure 6.2: Worldwide Gross vs Production Budget", weight = 'bold').set_fontsize('16')
ax.set_xlabel("Production Budget: (Millions of Dollars)", fontsize = 14, weight = 'bold')
ax.set_ylabel("Worldwide Gross: (Millions of Dollars)", fontsize = 14, weight = 'bold');
ax.ticklabel_format(style = "plain")
ax.xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:'{:.4g}'.format(x/100000)))
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:'{:.4g}'.format(x/1000000)))
```



Analyzing this scatterplot, I notice that there is a cluster of values for production values between **200 million and 400 million dollars**. So I drilled down a bit further to see if there was an ideal production budget to maximize revenue generation.

In [57]:

```
# Rough filter of prior correlation investigating dataframe
new_filtered_movie_ROI_df = filtered_movie_ROI_df[filtered_movie_ROI_df["production_budget"] >= 20000
new_filtered_movie_ROI_df.describe()
```

Out[57]:

	id	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_profit	ROI	s
count	43.00	43.00	43.00	43.00	43.00	43.00	43.00	43.00	
mean	31.74	238097674.42	300338741.37	873533771.26	2013.93	6.40	635436096.84	278.77	
std	17.99	47930035.76	172636709.38	405409448.49	2.69	2.87	407822714.21	178.35	
min	2.00	200000000.00	42762350.00	149762350.00	2010.00	2.00	-200237650.00	-57.21	
25%	16.00	200000000.00	195762525.00	627556713.00	2012.00	5.00	373024361.50	166.25	
50%	32.00	220000000.00	255119788.00	879620923.00	2013.00	6.00	617500281.00	271.79	
75%	47.50	255000000.00	408538310.50	1094239087.50	2016.00	7.00	879474467.50	412.75	
max	62.00	410600000.00	700059566.00	2048134200.00	2019.00	12.00	1748134200.00	666.91	

In [58]:

```
# Look at the filtered_movie_ROI dataframe
filtered_movie_ROI_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1369 entries, 0 to 1531
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 1369 non-null   int64
 1   movie              1369 non-null   object
 2   production_budget  1369 non-null   int64
 3   domestic_gross     1369 non-null   int64
 4   worldwide_gross    1369 non-null   int64
 5   release_year       1369 non-null   int64
 6   release_month      1369 non-null   int64
 7   worldwide_profit   1369 non-null   int64
 8   ROI                1369 non-null   float64
 9   movie_id           1369 non-null   object
 10  primary_title      1369 non-null   object
 11  original_title     1369 non-null   object
 12  start_year         1369 non-null   int64
 13  runtime_minutes    1369 non-null   float64
 14  genres             1369 non-null   object
 15  multi_genre        1369 non-null   object
dtypes: float64(2), int64(8), object(6)
memory usage: 181.8+ KB
```

In [59]:

```
# Rough filter of prior correlation investigating dataframe upper limit
new_filtered_movie_ROI_df = new_filtered_movie_ROI_df[new_filtered_movie_ROI_df["production_budget"]
new_filtered_movie_ROI_df.describe()
```

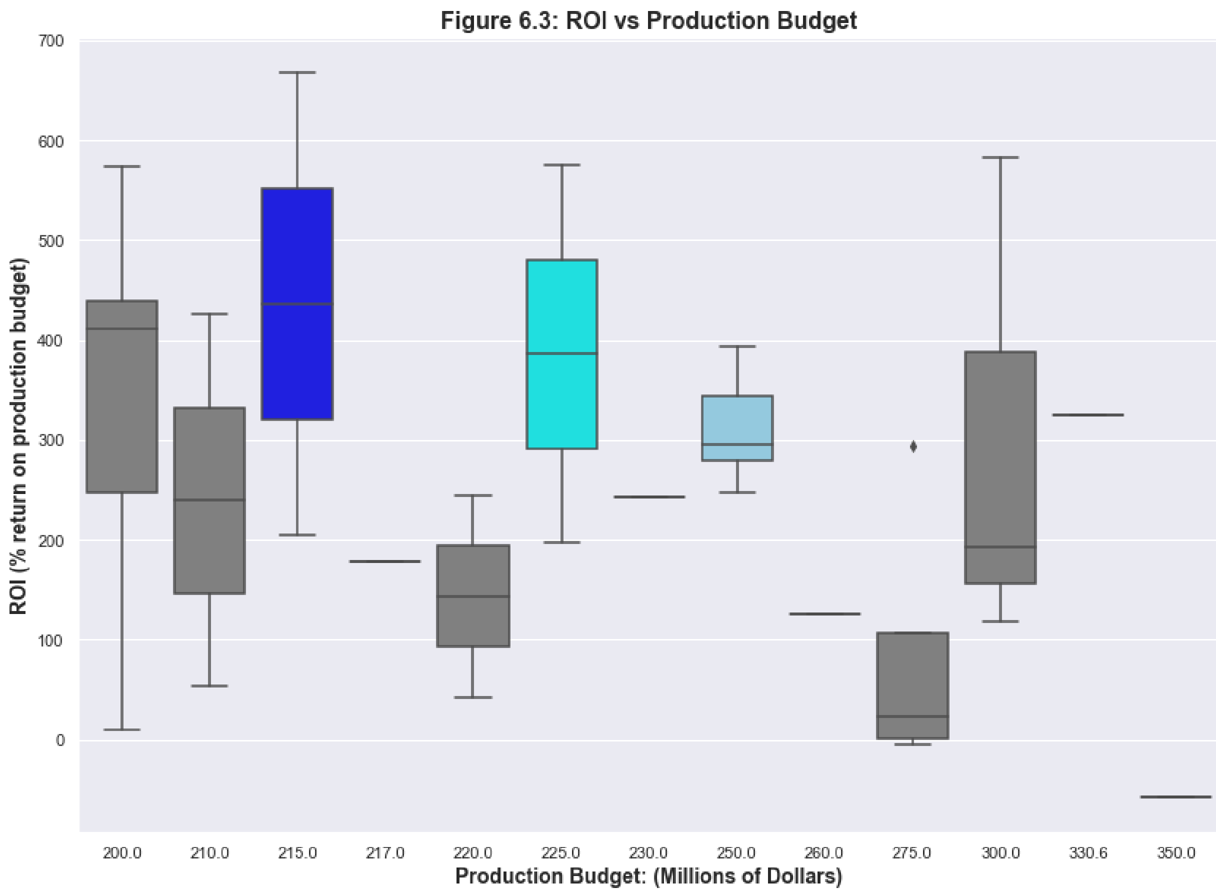
Out[59]:

	id	production_budget	domestic_gross	worldwide_gross	release_year	release_month	worldwide_profit	ROI	s
count	42.00	42.00	42.00	42.00	42.00	42.00	42.00	42.00	
mean	32.45	233990476.19	301750047.71	869435435.45	2014.00	6.43	635444959.26	281.73	
std	17.59	40128013.51	174478109.54	409421133.87	2.69	2.90	412766196.14	179.45	
min	3.00	200000000.00	42762350.00	149762350.00	2010.00	2.00	-200237650.00	-57.21	
25%	17.25	200000000.00	193606700.00	615225026.50	2012.00	5.00	366589872.25	178.39	
50%	32.50	218500000.00	256743321.50	873560602.00	2013.50	6.00	598560602.00	272.86	
75%	47.75	250000000.00	408765291.25	1099139081.75	2016.00	7.00	884771940.25	412.75	
max	62.00	350000000.00	700059566.00	2048134200.00	2019.00	12.00	1748134200.00	666.91	

In [60]:

```
# Plot the relationship between production budget and ROI for films with production budgets between $
# Set up theme
sns.set_theme(style="darkgrid", palette="Set2")

# Create plot variables
data = new_filtered_movie_ROI_df.head(100)
values = np.array(data.ROI)
labels = np.array(data.production_budget)/1000000
colors = ['grey', 'grey', 'blue','grey', 'grey', 'cyan', 'grey', 'skyblue', 'grey', 'grey']
# Set up plot figure size
plt.figure(figsize =(14,10))
ax = sns.boxplot(x = labels, y = values, data=new_filtered_movie_ROI_df.head(100), palette = colors)
ax.set_title("Figure 6.3: ROI vs Production Budget", weight = 'bold').set_fontsize('16')
ax.set_xlabel("Production Budget: (Millions of Dollars)", fontsize = '14', weight = 'bold')
ax.set_ylabel("ROI (% return on production budget)", fontsize = '14', weight = 'bold');
```



In Figure 6.2 I looked at the relationship between production budget and worlwide gross for the top 100 grossing films in my dataset. As a business, we want to get the largest revenue while minimizing production costs. From figure 2.1 I noticed that while the median production cost is around 20 million dollars, **production costs of around 200 million dollars have the most consistent worldwide profit**. In fact more films that had production budgets of over 100 million but less than 250 million dollars saw worldwide gross revenue of 500 million dollars or more than any other movies in the top 100 highest grossing movies in my dataset. While movies over 400 million dollars seem to generate at least the same amount of worldwide revenue as the production costs, the number of movies in this category seem to be outliers.

From Figure 6.3 the most ideal production cost is **215 million dollars**. In this figure ROI is calculated by dividing worlwide profit by the production cost and multiplying by 100.

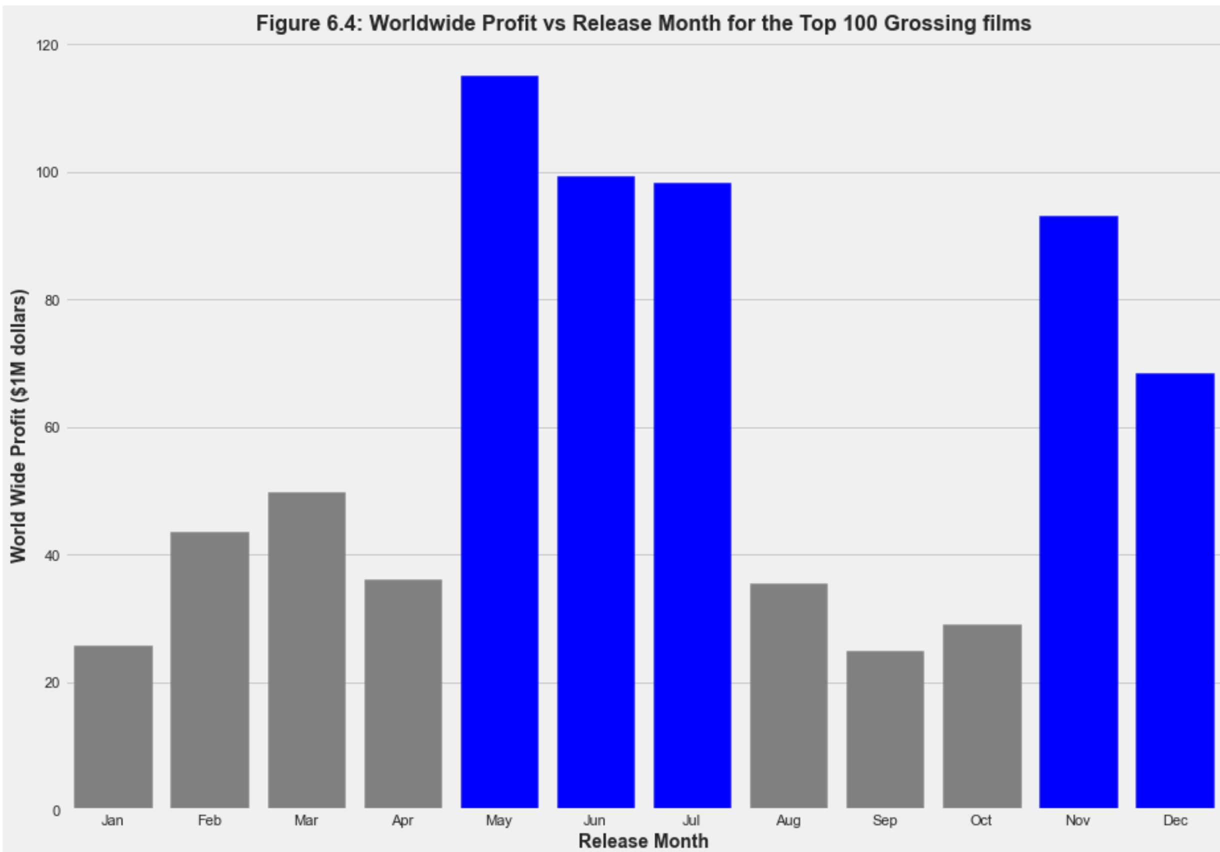
$$ROI = \frac{\text{worldwide profit}}{\text{production budget}} * 100$$

Earlier, I created a dataframe that contains both worldwide profit and the month that a film was released. Let's use the budgets_by_month_df dataframe to visualize which are the top five months to release a movie based on the profit made on those movies worldwide.

In [61]:

```
## Investigate the relationship between release_month and worldwide profit for the top 100 grossing f
# Set up theme
plt.style.use('fivethirtyeight')
# Set up Parameters
# Create plot variables
values = np.array(budgets_by_month_df.month_name)
labels = np.array(budgets_by_month_df.worldwide_profit)/1000000
clrs = ['grey' if (x != max(values)) else 'blue' for x in values]

# Set up plot figure size
plt.figure(figsize =(14,10))
ax = sns.barplot(x = values, y = labels, palette = clrs)
ax.set_title("Figure 6.4: Worldwide Profit vs Release Month for the Top 100 Grossing films", weight =
ax.set_xlabel("Release Month", fontsize = '14', weight = 'bold')
ax.set_ylabel("World Wide Profit ($1M dollars)", fontsize = '14', weight = 'bold');
for bar in ax.patches:
    if bar.get_height() < 60:
        bar.set_color('grey')
    else:
        bar.set_color('blue')
```



From figure 6.4 we can see that while May is the best month to release movies Jun, Jul, November, and December are among the top five profitable months for a movie to be released.

7 Exploratory Data Analysis Q2

7.1 Which director makes the most profitable movies?

Looking at the tn_and_imdb_full dataframe for correlations helped to answer this question. First I looked at a sorted version of a subset of the `tn_and_imdb_full` dataframe called `sorted_ww_profit_directors` . After I ensured there were no duplicates, I generated a barplot where I looked for correlations between directors and the top 100 most profitable movies from my combined dataframe.

In [62]:

```
# Create a new dataframe that has worldwide_profit and directors
tn_and_imdb_full.info()
worldwide_profit_directors_df = tn_and_imdb_full.loc[:, ["director","primary_title", "production_budg
                                        , "release_month", "start_year"]]
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4656 entries, 0 to 4655
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 4656 non-null   int64  
 1   movie              4656 non-null   object 
 2   production_budget  4656 non-null   int64  
 3   domestic_gross     4656 non-null   int64  
 4   worldwide_gross    4656 non-null   int64  
 5   release_year       4656 non-null   int64  
 6   release_month      4656 non-null   int64  
 7   worldwide_profit   4656 non-null   int64  
 8   ROI                4656 non-null   float64
 9   movie_id           4656 non-null   object 
 10  primary_title      4656 non-null   object 
 11  original_title     4656 non-null   object 
 12  start_year         4656 non-null   int64  
 13  runtime_minutes    4656 non-null   float64
 14  genres             4656 non-null   object 
 15  multi_genre        4656 non-null   object 
 16  person_id          4656 non-null   object 
 17  director           4656 non-null   object 
 18  primary_profession 4656 non-null   object 
dtypes: float64(2), int64(8), object(9)
memory usage: 727.5+ KB
```

In [63]:

```
# Create a Dataframe that shows movie titles, directors and worldwide profit
wp_directors_movies = worldwide_profit_directors_df.loc[:, ["primary_title", "director", "worldwide_p
wp_directors_movies = wp_directors_movies.drop_duplicates()
sorted_wp_directors_movies = wp_directors_movies.sort_values("worldwide_profit", ascending = False)
top_100_movies_with_directors_by_wwp_sorted = sorted_wp_directors_movies.head(100)
top_100_movies_with_directors_by_wwp_sorted
```

Out[63]:

	primary_title	director	worldwide_profit
1178	Avengers: Infinity War	Joe Russo	1748134200
1166	Avengers: Infinity War	Anthony Russo	1748134200
1994	Jurassic World	Colin Trevorrow	1433854864
1250	Furious 7	James Wan	1328722794
1527	The Avengers	Joss Whedon	1292935897
...	...	...	...
1142	Fast Five	Justin Lin	505163454
3948	Interstellar	Christopher Nolan	501379375
3770	Life of Pi	Ang Lee	500912003
3991	Thor: The Dark World	Alan Taylor	494602516
2339	Ant-Man and the Wasp	Peyton Reed	493144660

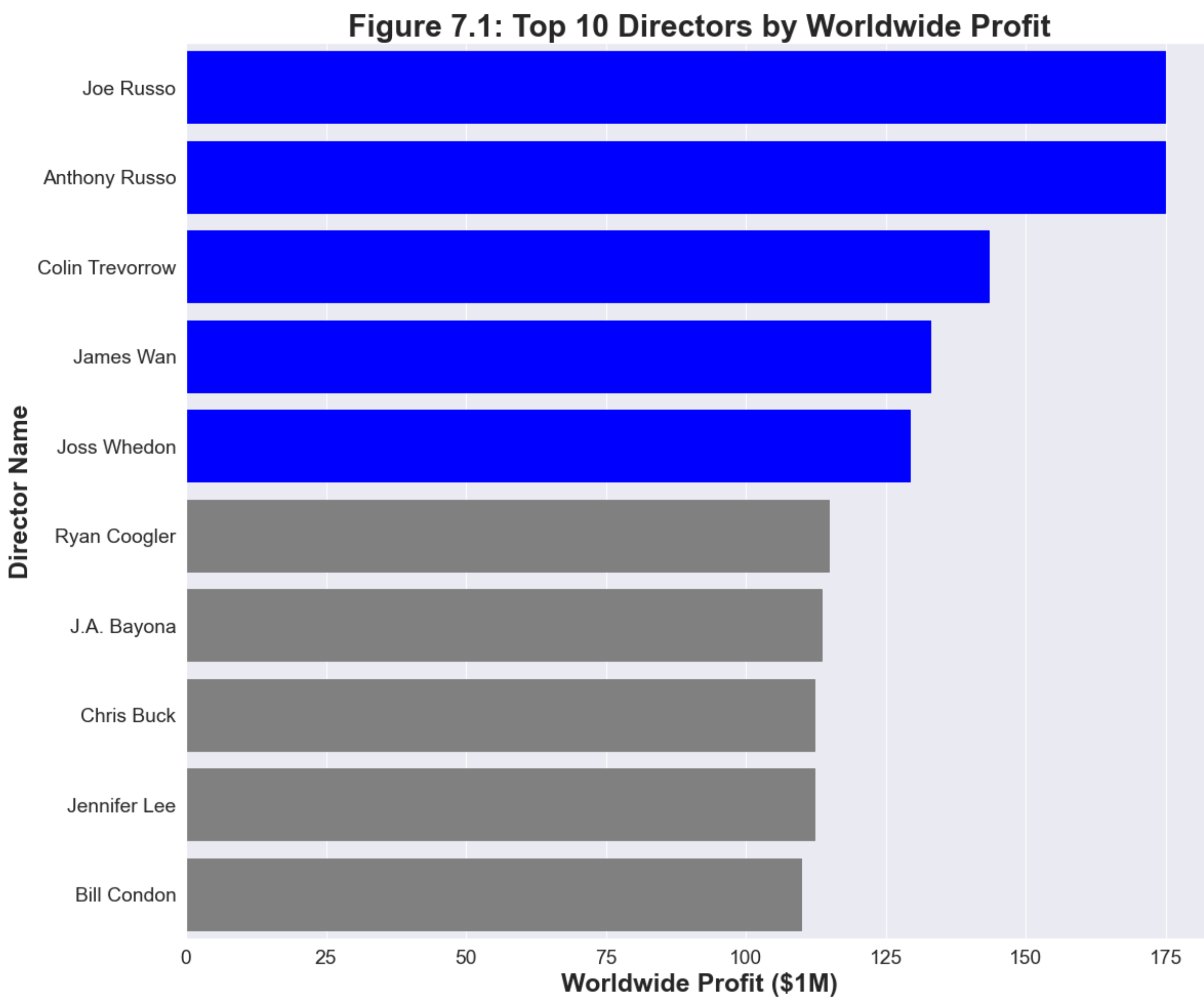
100 rows × 3 columns

In [64]:

```
# Group by director and worldwide_profit
sorted_ww_profit_directors = worldwide_profit_directors_df.sort_values("worldwide_profit", ascending
sorted_ww_profit_directors = sorted_ww_profit_directors.drop_duplicates()
top_100_movies_by_wwprofit = sorted_ww_profit_directors.head(100)
```

In [65]:

```
# Set up theme
# Set up Parameters
# Create plot variables
values = np.array(top_100_movies_by_wwprofit.head(10).director)
labels = np.array(top_100_movies_by_wwprofit.head(10).worldwide_profit)
sns.set(font_scale = 1.75)
clrs = ['grey' if (x != max(values)) else 'blue' for x in values]
# Set up plot figure size
plt.figure(figsize =(16,15))
ax = sns.barplot(y = values, x = labels, palette = clrs)
ax.set_title("Figure 7.1: Top 10 Directors by Worldwide Profit", weight = 'bold').set_fontsize('30')
ax.set_ylabel("Director Name", fontsize = '25', weight = 'bold')
ax.set_xlabel("Worldwide Profit ($1M)", fontsize = '25', weight = 'bold');
ax.xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:'{:.4g}'.format(x/10000000)))
for bar in ax.patches:
    if bar.get_width() < 1150000000:
        bar.set_color('grey')
    else:
        bar.set_color('blue')
```



From figure 7.1 we can see that while Joe and Anthony Russo have directed the most profitable movies out of the top 100 most profitable movies in my combined dataframe `top_100_movies_by_wwprofit` , Colin Treverrow, James Wan, and Joss Whedon are amongst the top 5 directors to choose from based on their movies' worldwide profit values.

8 Exploratory Data Analysis Q3

8.1 Which genres of movies make the most profit at the box office?

Looking at the `genre_budget_df` dataframe for correlations helped to answer this question. First I looked at a a subset of the `tn_and_imdb` dataframe called `genre_budget_df` and the `filtered_movie_ROI_df` for correleations. After I ensured there were no duplicates, I generated a dataframe `grouped_genre_ww_profit_df` where there was one movie genre per row. where I looked for correlations between the month, the genre and the worldwide profit for a movie. Finally, I used the insights from the `month_grouped_genre_median_ww_profit` my initial exploration to create bar graphs for the months of May, June, July, November and December.

In [66]:

```
# Investigating the relationship between Genre and Worldwide Profit
# Create a smaller dataframe to investigate production cost by genre
genre_budget_df = tn_and_imdb.loc[:, ["multi_genre","primary_title", "production_budget",  "worldwide
                                          , "release_month"]]
genre_budget_df.head(20)
```

Out[66]:

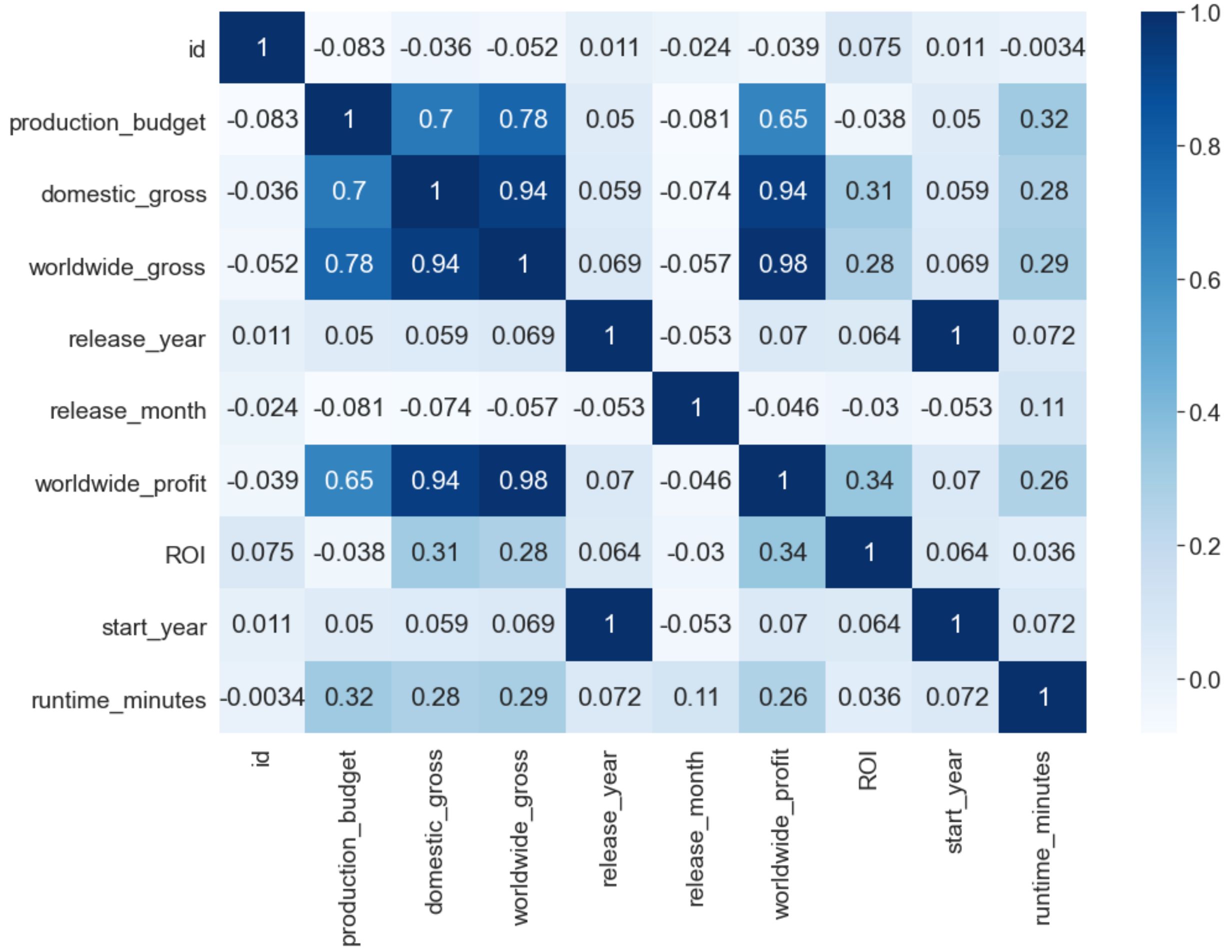
	multi_genre	primary_title	production_budget	worldwide_profit	ROI	release_month
0	[Adventure, Animation, Comedy]	Norm of the North	18000000	12535660	69.64	1
1	[Action, Crime, Thriller]	The Mechanic	42500000	33847393	79.64	1
2	[Action, Comedy, Crime]	One for the Money	42000000	-5802779	-13.82	1
3	[Action, Adventure, Crime]	Man on a Ledge	42000000	7621440	18.15	1
4	[Action, Comedy, Crime]	Ride Along 2	40000000	84827316	212.07	1
5	[Action, Comedy, Crime]	The Green Hornet	110000000	119155503	108.32	1
6	[Comedy]	Dirty Grandpa	11500000	93578449	813.73	1
7	[Comedy, Family, Fantasy]	Tooth Fairy	48000000	64610386	134.60	1
8	[Action, Adventure, Drama]	The 5th Wave	38000000	73336398	192.99	1
9	[Comedy]	Fifty Shades of Black	5000000	17113075	342.26	1
10	[Drama, Horror, Mystery]	The Rite	37000000	60143987	162.55	1

In [67]:

```
# Review correlations on filtered dataset

plt.figure(figsize =(14,10)).suptitle("Figure 8.1: Correlation Between Movie Attributes and ROI")
ax = sns.heatmap(filtered_movie_ROI_df.corr(), annot = True, cmap = "Blues")
```

Figure 8.1: Correlation Between Movie Attributes and ROI



In [68]:

```python
# Explode genres
exploded_genre_budget_df = genre_budget_df.explode("multi_genre")
exploded_genre_budget_df.head(20)
```

Out[68]:

	multi_genre	primary_title	production_budget	worldwide_profit	ROI	release_month
0	Adventure	Norm of the North	18000000	12535660	69.64	1
0	Animation	Norm of the North	18000000	12535660	69.64	1
0	Comedy	Norm of the North	18000000	12535660	69.64	1
1	Action	The Mechanic	42500000	33847393	79.64	1
1	Crime	The Mechanic	42500000	33847393	79.64	1
1	Thriller	The Mechanic	42500000	33847393	79.64	1
2	Action	One for the Money	42000000	-5802779	-13.82	1
2	Comedy	One for the Money	42000000	-5802779	-13.82	1
2	Crime	One for the Money	42000000	-5802779	-13.82	1
3	Action	Man on a Ledge	42000000	7621440	18.15	1
3	Adventure	Man on a Ledge	42000000	7621440	18.15	1
3	Crime	Man on a Ledge	42000000	7621440	18.15	1
4	Action	Ride Along 2	40000000	84827316	212.07	1
4	Comedy	Ride Along 2	40000000	84827316	212.07	1
4	Crime	Ride Along 2	40000000	84827316	212.07	1
5	Action	The Green Hornet	110000000	119155503	108.32	1
5	Comedy	The Green Hornet	110000000	119155503	108.32	1
5	Crime	The Green Hornet	110000000	119155503	108.32	1
6	Comedy	Dirty Grandpa	11500000	93578449	813.73	1
7	Comedy	Tooth Fairy	48000000	64610386	134.60	1

In [69]:

```python
# Group by multi_genre by production budget
grouped_genre_budget = exploded_genre_budget_df.groupby("multi_genre")["production_budget"].median()
grouped_genre_budget.head(20)
```

Out[69]:

```
multi_genre
Action          57750000
Adventure      100000000
Animation       83500000
Biography       20000000
Comedy          25500000
Crime           28000000
Documentary      3000000
Drama           17500000
Family          40000000
Fantasy         60000000
History         25000000
Horror          10000000
Missing         10750000
Music           15000000
Musical         55000000
Mystery         12500000
Reality-TV       1000000
Romance         17000000
Sci-Fi          74000000
Sport           18000000
Name: production_budget, dtype: int64
```

In [70]:

```
# Create a dataframe from the series
grouped_genre_budget_df = grouped_genre_budget.to_frame(name = "production_budget").reset_index()
grouped_genre_budget_df.head(20)
```

Out[70]:

	multi_genre	production_budget
0	Action	57750000
1	Adventure	100000000
2	Animation	83500000
3	Biography	20000000
4	Comedy	25500000
5	Crime	28000000
6	Documentary	3000000
7	Drama	17500000
8	Family	40000000
9	Fantasy	60000000
10	History	25000000
11	Horror	10000000
12	Missing	10750000
13	Music	15000000
14	Musical	55000000
15	Mystery	12500000
16	Reality-TV	1000000
17	Romance	17000000
18	Sci-Fi	74000000
19	Sport	18000000

In [71]:
```python
# Sort by Production Budget
grouped_genre_budget_df.sort_values("production_budget", ascending = False)
```

Out[71]:

	multi_genre	production_budget
1	Adventure	100000000
2	Animation	83500000
18	Sci-Fi	74000000
9	Fantasy	60000000
0	Action	57750000
14	Musical	55000000
8	Family	40000000
22	Western	35000000
5	Crime	28000000
4	Comedy	25500000
10	History	25000000
3	Biography	20000000
20	Thriller	20000000
21	War	20000000
19	Sport	18000000
7	Drama	17500000
17	Romance	17000000
13	Music	15000000
15	Mystery	12500000
12	Missing	10750000
11	Horror	10000000
6	Documentary	3000000
16	Reality-TV	1000000

In [72]:
```python
# Count unique elements in each column
exploded_genre_budget_df.nunique()
```

Out[72]:
```
multi_genre            23
primary_title        1468
production_budget     260
worldwide_profit     1414
ROI                  1351
release_month          12
dtype: int64
```

In [73]:
```python
# Sort the dataframe
exploded_genre_budget_df.sort_values("worldwide_profit", ascending = False)
exploded_genre_budget_df.head()
```

Out[73]:

	multi_genre	primary_title	production_budget	worldwide_profit	ROI	release_month
0	Adventure	Norm of the North	18000000	12535660	69.64	1
0	Animation	Norm of the North	18000000	12535660	69.64	1
0	Comedy	Norm of the North	18000000	12535660	69.64	1
1	Action	The Mechanic	42500000	33847393	79.64	1
1	Crime	The Mechanic	42500000	33847393	79.64	1

In [74]:

```
# Group by multi_genre by production budget
grouped_genre_ww_profit = exploded_genre_budget_df.groupby("multi_genre")["worldwide_profit"].median(
```

In [75]:

```
# Create a dataframe from the series
grouped_genre_ww_profit_df = grouped_genre_ww_profit.to_frame(name = "worldwide_profit").reset_index(
grouped_genre_ww_profit_df.head()
```

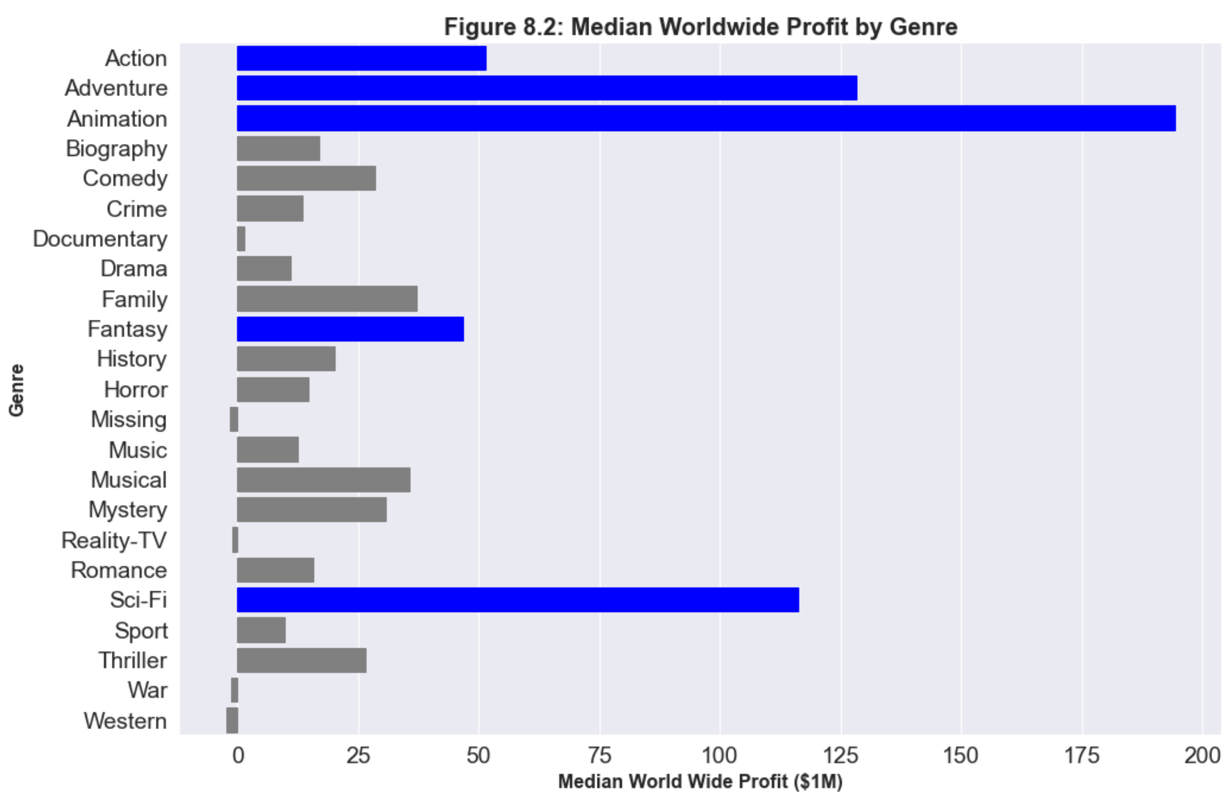
Out[75]:

	multi_genre	worldwide_profit
0	Action	51387479.50
1	Adventure	128314513.00
2	Animation	194139928.50
3	Biography	16983758.50
4	Comedy	28527406.00

In [76]:

```
# Set up theme
#sns.set_theme(style="darkgrid", palette="Set2")
# Set up Parameters
# Create plot variables
values = np.array(grouped_genre_ww_profit_df.multi_genre)
labels = np.array(grouped_genre_ww_profit_df.worldwide_profit)/1000000
clrs = ['grey' if (x != max(values)) else 'blue' for x in values]

# Set up plot figure size
plt.figure(figsize =(14,10))
ax = sns.barplot(y = values, x = labels, palette = clrs)
ax.set_title("Figure 8.2: Median Worldwide Profit by Genre", weight = 'bold').set_fontsize('20')
ax.set_ylabel("Genre", fontsize = '16', weight = 'bold')
ax.set_xlabel("Median World Wide Profit ($1M)", fontsize = '16', weight = 'bold');
for bar in ax.patches:
    if bar.get_width() < 45:
        bar.set_color('grey')
    else:
        bar.set_color('blue')
```



In [77]:

```python
# Group by multi_genre by production budget
month_grouped_genre_median_ww_profit = exploded_genre_budget_df.groupby(["release_month","multi_genre
```

In [78]:

```python
# Look at first five rows of the grouped dataframe
month_grouped_genre_median_ww_profit.head()
```

Out[78]:

release_month	1	2	3	4	5	6	7	
multi_genre								
Action	36118378.00	80378084.00	90808837.00	55275291.00	119825506.00	167194805.00	111115739.50	32295262.
Adventure	51126600.00	101249630.00	134455704.50	222908183.00	299326618.00	265522281.00	168902025.00	61600183.
Animation	92529966.00	237013181.00	190402163.00	131618089.50	279327887.00	506587299.50	457914642.50	19993255.
Biography	-3317305.00	26096200.00	-1534155.00	29158652.00	-2777427.50	-2417804.00	-8567177.00	14371191.
Comedy	51120275.50	49911903.00	19730861.50	33014010.50	51560777.00	35672764.00	72378492.00	27000339.

In [79]:

```python
# Identify best genre for each month
# fill in missing values with zero
# display values as intergers in dataframe
month_grouped_genre_median_ww_profit.fillna(0, inplace = True)
month_grouped_genre_median_ww_profit = month_grouped_genre_median_ww_profit.astype(int)
month_grouped_genre_median_ww_profit.style.highlight_max(color = "royalblue", axis = 0)
```

Out[79]:

release_month	1	2	3	4	5	6	7	8	9	
multi_genre										
Action	36118378	80378084	90808837	55275291	119825506	167194805	111115739	32295262	30335864	480
Adventure	51126600	101249630	134455704	222908183	299326618	265522281	168902025	61600183	52341726	245
Animation	92529966	237013181	190402163	131618089	279327887	506587299	457914642	19993255	123522354	476
Biography	-3317305	26096200	-1534155	29158652	-2777427	-2417804	-8567177	14371191	26492104	17
Comedy	51120275	49911903	19730861	33014010	51560777	35672764	72378492	27000339	17621449	17
Crime	13543388	36742138	21444133	7883237	28041566	3062896	21068890	13836080	23785322	37
Documentary	0	135260488	20667389	-2313312	1495262	-1239832	884276	4856688	-341867	40
Drama	13618920	27362398	12107621	6594052	26721826	420962	11477345	14660323	7846438	43
Family	18752858	-14731320	70279266	4340177	134861276	31777043	57986320	15138912	59068724	81
Fantasy	46387087	25580458	175004422	28095698	123714144	240238093	13281452	53461527	-4774693	24
History	7685569	68826270	-16800490	-1012542	0	-2195774	177592786	174182981	-25801704	200
Horror	53811445	18996911	23204379	22651864	28005871	14720203	15434588	44104225	6542011	5
Missing	0	0	90808837	0	0	-500000	0	-18000000	464792	
Music	8266990	47135679	34228790	0	163134096	5148908	109160597	14660323	-2623879	8
Musical	0	0	962345408	0	42527466	-8968068	-5068194	0	0	-19
Mystery	44485091	45960816	31812062	31291821	41411721	14131551	68345423	44104225	20400229	184
Reality-TV	0	0	0	-1000000	0	0	0	0	0	
Romance	13618920	66050951	26627836	40281179	25168023	3672318	31449135	30858465	9870593	38
Sci-Fi	20909437	72520550	68946715	167916633	369076069	213571084	123617305	41540205	21517819	397
Sport	0	18745772	0	4847480	12042788	28527161	63073118	1874668	-485085	4
Thriller	49401938	40176634	26627836	22651864	497895	41142372	57956618	31795191	12321923	462
War	0	48319835	220780051	-2230701	-31979010	-1973745	0	-11912207	9238343	-8
Western	0	0	2446952	0	0	0	-14997885	0	35142426	-32

In [96]:

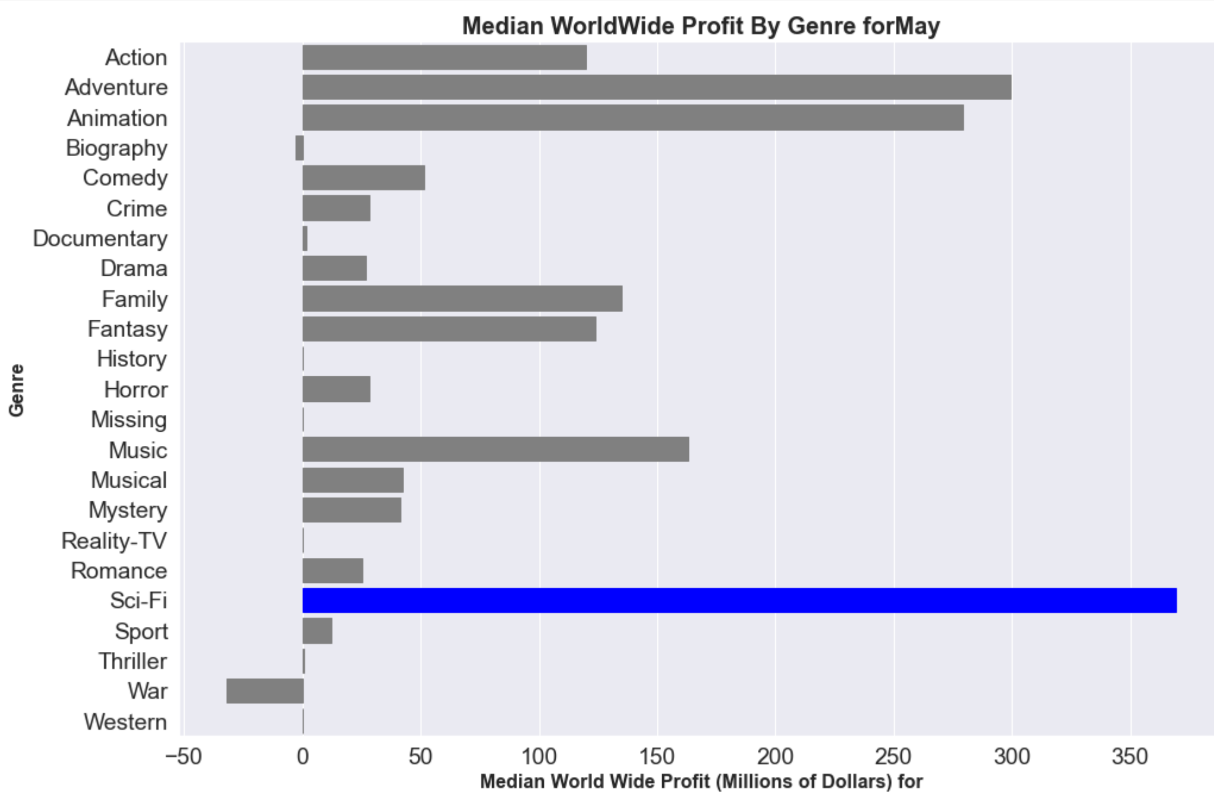
```
def plot_month_genre_bar(df, month_num, month_name, lim):
    """
    Plot bar graph of genre vs ww profit by month

    Inputs:
    -
    df: pandas dataframe
    month_num : integer
    xlabel: independent variable label
    ylabel: dependent variable label

    Output:
    -
    bar plot
    """
    x = df.index
    month_data = df[month_num]/1000000
    xlabel = "Median World Wide Profit (Millions of Dollars) for"
    ylabel =  "Genre"
    title = "Median WorldWide Profit By Genre for " + month_name
    # create a palette that highlights the maximum values in each month in blue
    clrs = ['grey' if (x != max(values)) else 'blue' for x in values]
    # Set up plot figure size
    plt.figure(figsize =(14,10))
    # Plot bar plot using inputs of month, index and palette
    ax = sns.barplot(x = month_data, y = x ,palette = clrs )
    # Setup titles and axes labels
    ax.set_title(title, weight = 'bold').set_fontsize('20')
    ax.set_ylabel(ylabel, fontsize = '16', weight = 'bold')
    ax.set_xlabel(xlabel, fontsize = '16', weight = 'bold');
    for bar in ax.patches:
        if bar.get_width() < lim:
            bar.set_color('grey')
        else:
            bar.set_color('blue')
```

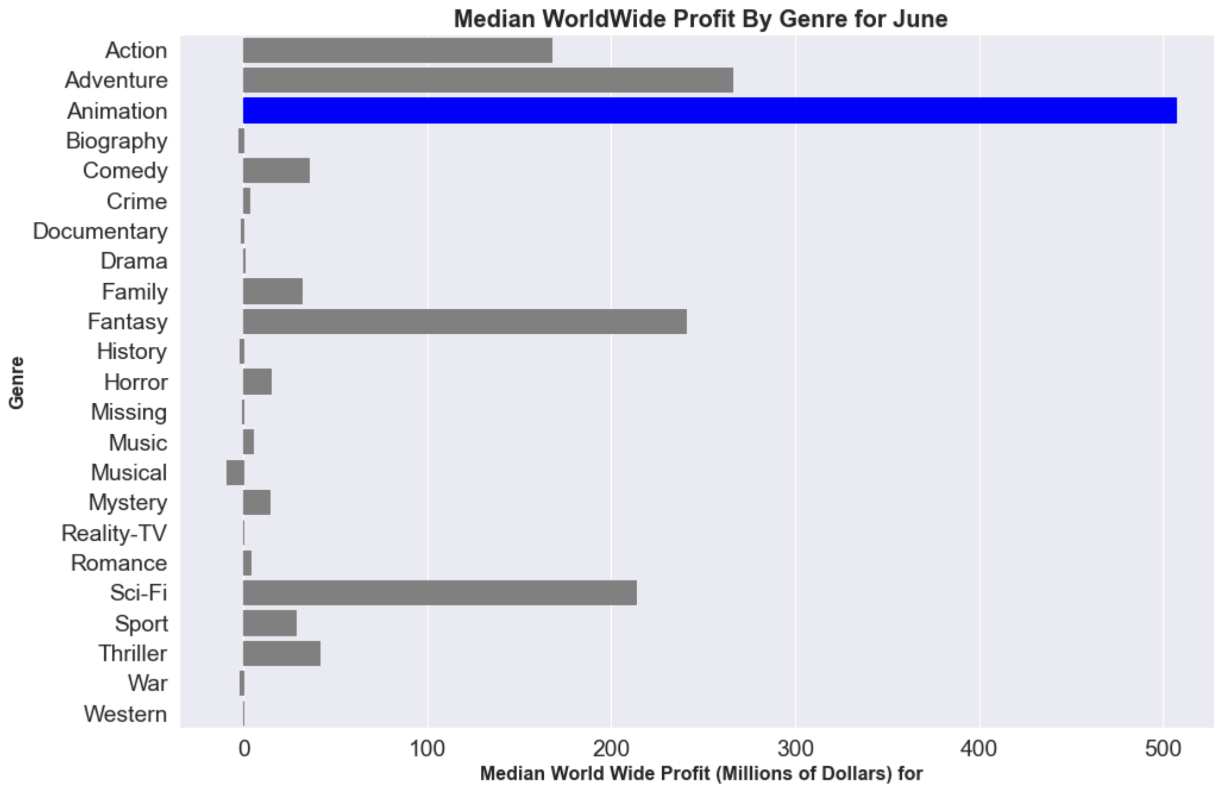
In [95]:

```
plot_month_genre_bar(month_grouped_genre_median_ww_profit, 5, "May", 300 )
```



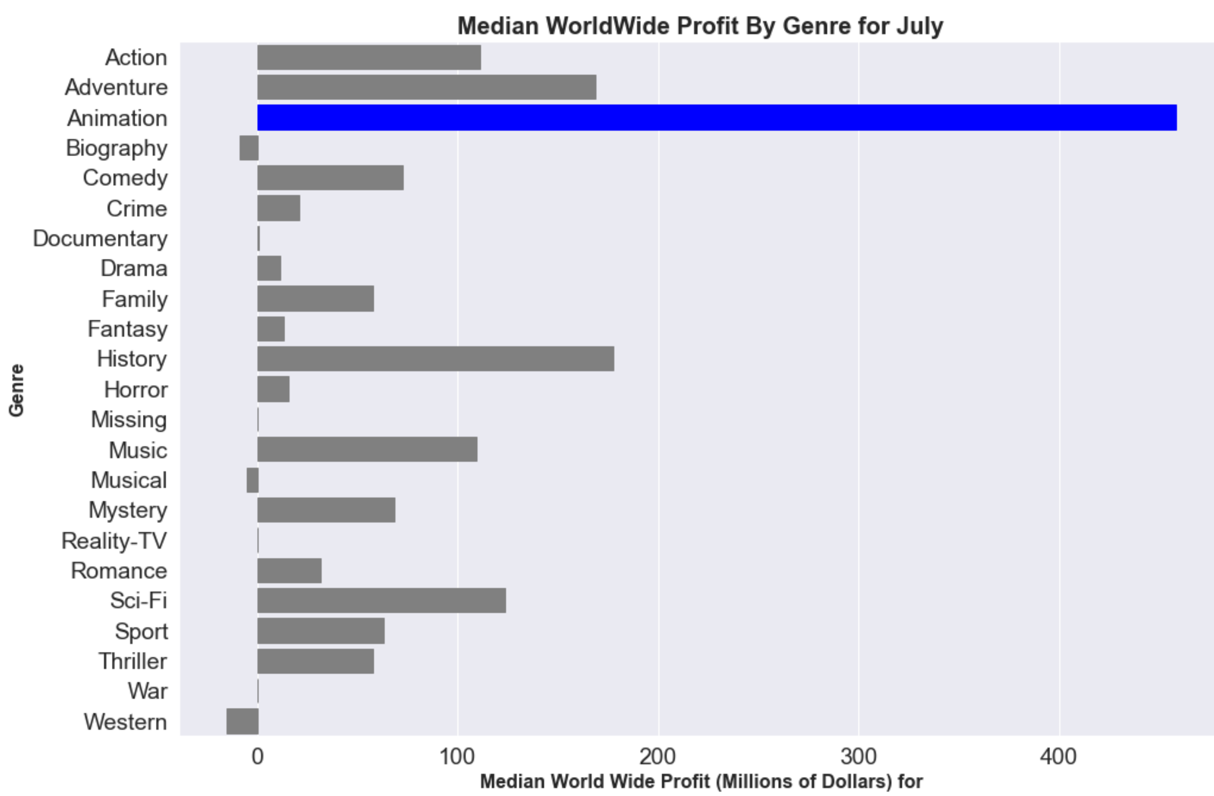
In [97]:

```
plot_month_genre_bar(month_grouped_genre_median_ww_profit, 6, "June", 300 )
```

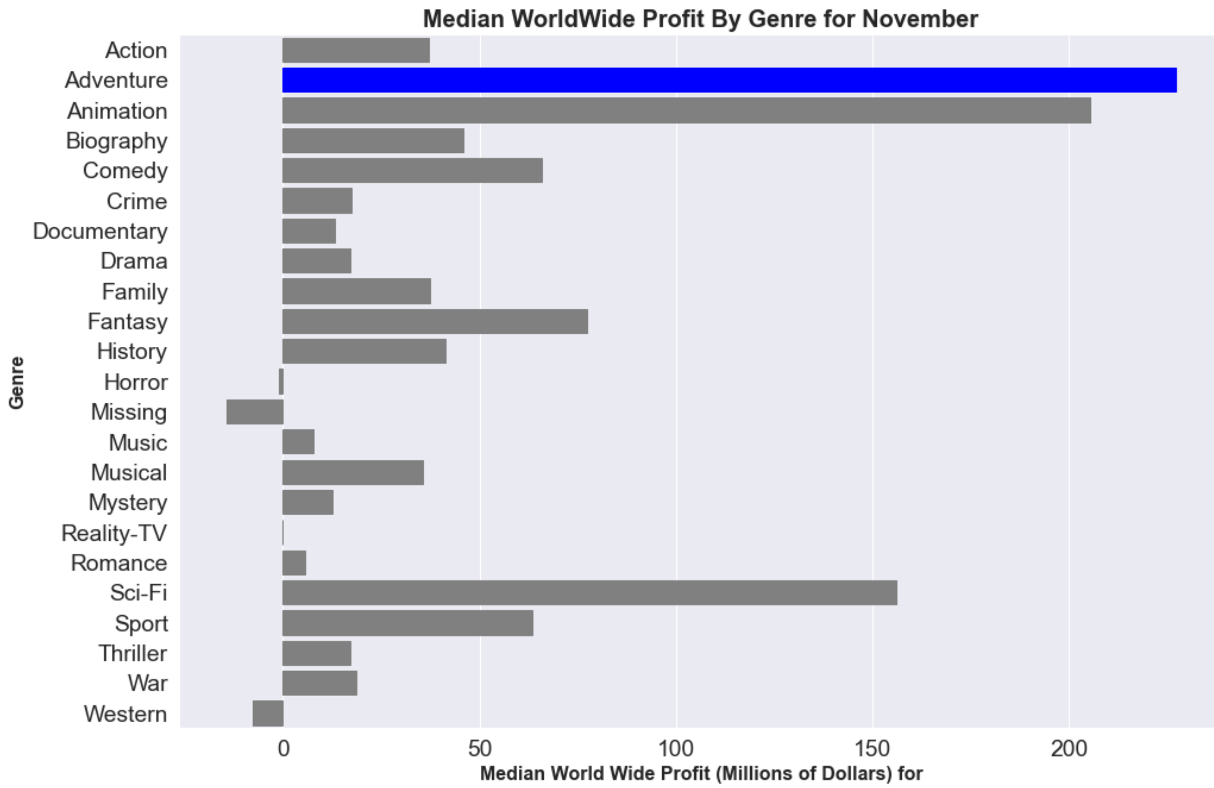


In [98]:

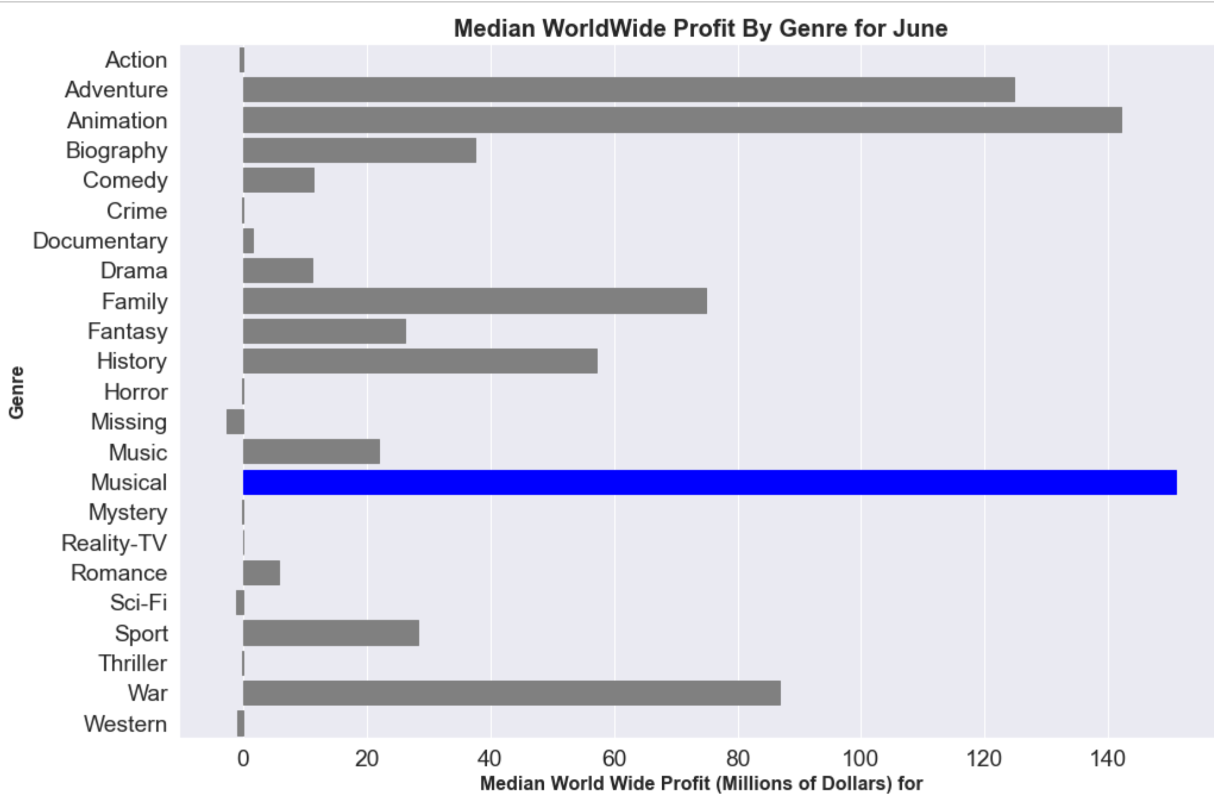
```
plot_month_genre_bar(month_grouped_genre_median_ww_profit, 7, "July", 300 )
```



In [99]:
```
plot_month_genre_bar(month_grouped_genre_median_ww_profit, 11, "November", 220 )
```



In [100]:
```
plot_month_genre_bar(month_grouped_genre_median_ww_profit, 12, "June", 145 )
```



From the figures above, I again noted that there was a positive correlation between both worldwide gross and production budget (correlation ~0.78), along with and worldwide profit and production budget (correlation ~0.65). I went with profit since there are movies within the dataset that generate large amounts of revenue and have equally large associated production costs. From Figure 8.2, I saw that while animation is the most profitable genre to release in theaters, adventure, Sci-Fi, fantasy and action are among the top 5 profitable genres of movies to release. Digging into the data a little further showed that the most profitable genres to during the top 5 release months were:

- May for Sci-Fi movies
- June and July for Animation movies
- November for movies adventure movies
- December for Musical movies

9 Insights

It seems that the best months to release a movie are May, June, July, November and December. With **May**, **June** and **July** (Summer months) yielding the largest profits in the top most profitable films. When viewing the top directors of the top 100 most profitable movies, I found that Joe and Anthony Russo directed the most profitable movies out of the top 100 most profitable movies. While Colin Treverrow, James Wan, and Joss Whedon are amongst the top 5 directors to choose from when choosing directors based on their ability to create profitable movies. Finally, my data suggests that Animation is the best genre of movie to make with Adventure, Action, Sci-Fi and Musicals rounding out the top five best genres to create based on worldwide profit.

10 Recommendations

I have the following recommendations:

Create movies that are **Sci-Fi, Animation, or Adventure films** with a budget of approximately 215 million dollars.

Use effective directors, specially choose a director like Joe Russo, Anthony Russo, Colin Treverrow, James Wan or Joss Whedon who have demonstrated successful direction of profitable movies on the worldwide stage.

Launch Sci-Fi films during May. Launch Animation films during June or July. Launch Adventure fims during November. This will allow the new Microsoft studio to diversify their entry into the large video content space.

Type *Markdown* and LaTeX: α^2