Final Project Submission

Please fill out:

- Student name: Kevin Spring
- Student pace: self paced
- Scheduled project review date/time:
- Instructor name: Morgan Jones
- Blog post URL:

Introduction

Overview

In 2021, digital, theatrical, and physical sales of movies were $136.5 billion globally and \$36.8 billion in the United States. Digital sales account for 80% of revenue, theatrical sales are 12%, and physical media is the remaining 8%.

Microsoft is creating a new movie studio and wants to know what types of movies they should create to tap into this industry. This project will explore what types of movies are currently doing the best at the box office and recommend what types of movies executives at Microsoft's movie studio should create.

Business Problem

Microsoft is creating a new movie studio and wants to know what types of movies to make to be successful. Success can be measured with the amount of profit that the movie makes and the reputation as measured in viewer ranking.

To be successful, Microsoft needs to know what movie genres have the highest ranking and profit, when these movies are released at the box office during the year, and how much should be budgeted for these movies.

Summary of Recommendations

- Movie genre should be **action**, **adventure**, **animation**, or **science fiction**
- The movie should be released in 2nd or 4th quarter of the year
- The movie should have a budget of $60 million to \$100 million

Data

Microsoft's new movie studio wants to know what would be the best types of movies to make. To do this I will need to know the viewer ranking of movies, cost to make the movies, movies gross revenue, and release date.

Data available

- All data located in `./zippedData` folder
- Internet Movie Database (IMDb)
  - `im.db`
  - SQLite database
  - Contains IMDb user generated rankings
- Box Office Mojo (BOM)
  - `bom.movie_gross.csv.gz`
  - domestic revenue ($USD)
  - foreign revenue ($USD)
- The Numbers (CSV)
  - `tn.movie_budgets.csv.gz`
  - production budget ($USD)

In [1]:
```
# Import libraries
import pandas as pd
```

Box Office Mojo (BOM) Data

In [2]:
```
# Import data
bom_df = pd.read_csv('zippedData/bom.movie_gross.csv.gz', header = 0)

# Return top of Box Office Mojo Data
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 [3]:
```
# Return top four rows in bom_df
bom_df.head(4)
```

Out[3]:

	title	studio	domestic_gross	foreign_gross	year

	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

Data Discussion

Box Office Mojo (BOM) has information about gross revenue with 3359 non-null domestic gross revenue and 2037 foreign non-null gross revenue.

`domestic_gross` is listed as a `float` data type but `foreign_gross` is not and likely has `string` data types within this column. I will need to find the `string` data in the `foreign_gross` column and convert it to a numerical data type.

The Numbers Database (TNDB) Data

In [4]:
```
# import data
tndb_df = pd.read_csv('zippedData/tn.movie_budgets.csv.gz', header = 0)

tndb_df.info()
```

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

In [5]:
```
tndb_df.head(4)
```

Out[5]:

	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

Data Discussion

TNDB contains information on movie budget, release date, and domestic and worldwide revenue. The budget is listed as a produciton budget but this does not include the marketing budget. The Numbers database has 5,782 non-null values which indicates it may have placeholder values replacing missing values.

`production_budget`, `domestic_gross`, and `worldwide_gross` all are in `string` data type with dollar signs ($) and commas (,) in the string. I will be using the revenue data from BOM and not TNDB so `domestic_gross` and `worldwide_gross` can be dropped and only the `produciton_budget` data will need to be cleaned. TNDB's gross revenue data may need to be used in the event of missing values in BOM's `domestic_gross` or `foreign_gross` columns.

Internet Movie Database (IMDb) data

This data is contained in a SQLite database with 7 tables. From the ERD I will be using the tables `movie_basics` and `movie_ratings` which contains the title of the movie, ratings, genre, and runtime.

In [6]:
```
# Make connection to SQLite database
import sqlite3

# will join 2 tables movie_basics and movie_ratings
conn = sqlite3.connect('zippedData/imdb.sqlite')
cur = conn.cursor()
q = '''
        SELECT *
        FROM movie_basics
        INNER JOIN movie_ratings
             USING(movie_id);
'''
imdb_df  = pd.read_sql_query(q, conn)
imdb_df.info()

conn.close()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   movie_id         73856 non-null  object 
 1   primary_title    73856 non-null  object 
 2   original_title   73856 non-null  object 
 3   start_year       73856 non-null  int64  
 4   runtime_minutes  66236 non-null  float64
 5   genres           73052 non-null  object 
 6   averagerating    73856 non-null  float64
 7   numvotes         73856 non-null  int64  
dtypes: float64(2), int64(2), object(4)
memory usage: 4.5+ MB
```

In [7]:
```
imdb_df.head(4)
```

Out[7]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	averagerating
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama	7.0
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama	7.2
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama	6.9
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama	6.1

Data Discussion

IMDb contains the average viewer ranking in the `averagerating` column and genre in the `genres` column. There are 804 null values stored in the `genres` column so I will remove the records with these null values before joining the dataframes. This dataframe contains 73,856 records but many of them will be dropped when joining with the `tndb_df` and `bom_df` dataframes as BOM and TNDB have far less records.

Process

In this step of the analysis I will join and clean the raw data of `tndb_df`, `bom_df`, and `imdb_df`.

Processing Steps

1. Discover and remove duplicates
2. Join duplicate free `tndb_df`, `bom_df`, and `imdb_df`
3. Find and correct missing values (NaN)
    - Replace missing `foreign_gross` and `domestic_gross_y` from the BOM data with TNDB's data
4. Check for placeholders of missing values
5. Remove irrelevant columns
6. Calculate worldwide gross revenue
7. Convert single string in `genre` column to a list of genre strings

1. Find and Correct Duplicated Entries

In [8]:

```
# Look for duplicates
# IMDB
print("IMDB")
print("Duplicated records: ", imdb_df.duplicated(['primary_title', 'start_year']).sum()

# BOM
```

```
print("\nBOM")
print("Duplicated records: ", bom_df.duplicated(['title','year']).sum())

# TNDB
print("\nTNDB")
print("Duplicated records: ", tndb_df.duplicated(['movie', 'release_date']).sum())
```

```
IMDB
Duplicated records:  585

BOM
Duplicated records:  0

TNDB
Duplicated records:  0
```

The dataframes `imdb_df`, `bom_df`, and `tndb_df` will be joined based on the movie title and year of release. Dupilicated entries titles and year of release could be a problem if the data between the dataframes are joined on the incorrect records. There are no duplicated entries in `tndb_df` on columns `movie` and `release_date` or in `bom_df` on columns `title` and `year`. `imdb_df` has 585 duplicate records based on columns `primary_title` and `start_year`.

In [9]:
```
# diplay subset of IMDb duplicated records ordered by number of votes
imdb_df[imdb_df.duplicated(['primary_title', 'start_year'])].sort_values(['numvotes'],
```

Out[9]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	a
50177	tt4972582	Split	Split	2016	117.0	Horror,Thriller	
14635	tt1980929	Begin Again	Begin Again	2013	104.0	Comedy,Drama,Music	
36580	tt3488710	The Walk	The Walk	2015	123.0	Adventure,Biography,Drama	
44600	tt4287320	The Circle	The Circle	2017	110.0	Drama,Sci-Fi,Thriller	
31646	tt3064298	Man Up	Man Up	2015	88.0	Comedy,Drama,Romance	
2152	tt1336617	Cyrus	Cyrus	2010	91.0	Comedy,Drama,Romance	
54898	tt5571734	Pink	Pink	2016	136.0	Drama,Thriller	
46025	tt4463816	Terminal	Terminal	2018	95.0	Crime,Drama,Thriller	
45221	tt4359416	Taxi	Taxi	2015	82.0	Comedy,Drama	
40053	tt3802576	Brothers	Brothers	2015	156.0	Action,Drama,Sport	

In [10]:
```
# Focus on the movie 'Cyrus'
imdb_df[(imdb_df['primary_title'] == 'Cyrus') & (imdb_df['start_year'] == 2010)]
```

Out[10]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	avera
2081	tt1327709	Cyrus	Cyrus	2010	87.0	Crime,Horror,Mystery	
2152	tt1336617	Cyrus	Cyrus	2010	91.0	Comedy,Drama,Romance	

In [11]:
```
# Is this movie found in the BOM data?
bom_df[bom_df['title'] == 'Cyrus']
```

Out[11]:

	title	studio	domestic_gross	foreign_gross	year
171	Cyrus	FoxS	7500000.0	2500000	2010

In [12]:
```
# Is this movie found in the TNDB data?
tndb_df[tndb_df['movie'] == 'Cyrus']
```

Out[12]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross
4026	27	Jun 18, 2010	Cyrus	$7,000,000	$7,468,936	$10,062,896

Remove duplicated data

`imdb_df` contains 585 duplicated records based on the `primary_title` and `start_year`. I will be joining the data on these two columns. `bom_df` and `tndb_df` have ten times less records than `imdb_df` so much of the data from IMDB will not be included in this analysis. One way to remove duplicated entries is to keep the record with the highest number of votes as recorded in `numvotes` column as the less known movies with the same name and release year will likely not be represented in the BOM and TNDB data.

The IMDb data without duplicate movie title and year of released will be saved as `imdb_df_filtered`.

In [13]:
```
# Create a new dataframe that filters out the duplicated records
imdb_df_filtered = imdb_df.sort_values(['numvotes']).drop_duplicates(['primary_title',

# Print check to see how many duplicates left
print('Number of duplicated movie title and year of release', len(imdb_df_filtered[imdb

# Check the movie 'Cyrus' again to make sure it has no duplicates in the new dataframe
imdb_df_filtered[imdb_df_filtered['primary_title'] == 'Cyrus']
```

```
Number of duplicated movie title and year of release 0
```

Out[13]:

	movie_id	primary_title	original_title	start_year	runtime_minutes	genres	avera
2152	tt1336617	Cyrus	Cyrus	2010	91.0	Comedy,Drama,Romance	

IMDB Data Discussion

All duplicated entries were removed and the record with the highest `numvotes` was kept.

2. Join Dataframes

`tndb_df`, `bom_df`, and `imdb_df_filtered` will be joined based on the movie's title and year of release.

- `tndb_df` (TNDB data)
  - `movie` : Movie title in `string` datatype
  - `release_date` : Release date in format: "Mon day, Year" `string` datatype
- `bom_df` (BOM data)
  - `title` : Movie title
  - `year` : Year of release in `string` datatype
- `imdb_df_filtered` (IMDb data)
  - `primary_title` : Movie title
  - `start_year` : Year of release in `string` datatype

TNDB's `release_date` needs to be converted to `datetime64` data type and then the year stored in a new column for all the dataframes to be joined together correctly.

In [14]:
```
# Convert release_date to datetime64, extract year, and store in a new column
# Copy TNDB dataframe to clean
tndb_df_fixdate = tndb_df.copy()

# Convert string date to datetime
tndb_df_fixdate['release_date'] = pd.to_datetime(tndb_df['release_date'], format='%b %d

# Create a new column with year called `tndb_year`
tndb_df_fixdate['tndb_year'] = tndb_df_fixdate['release_date'].apply(lambda x: int(x.st
tndb_df_fixdate.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   release_date       5782 non-null   datetime64[ns]
 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        
 6   tndb_year          5782 non-null   int64         
dtypes: datetime64[ns](1), int64(2), object(4)
memory usage: 316.3+ KB
```

In [15]:
```
# Inner Join of TNDB and BOM dataframes on movie title and year released
tndb_bom_df = tndb_df_fixdate.merge(bom_df, how = 'inner', left_on = ['movie', 'tndb_ye
print("Total records in joined TNDB and BOM", len(tndb_bom_df))
tndb_bom_df.head(4)
```

```
Total records in joined TNDB and BOM 1215
```

Out[15]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
0	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	2011	Pira Carib Str

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
1	4	2015-05-01	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	2015	Aver A U
2	7	2018-04-27	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200	2018	Aver Ir
3	9	2017-11-17	Justice League	$300,000,000	$229,024,295	$655,945,209	2017	Ju Le

In [16]:
```
# Inner join of TNDB/BOM joined dataframe and IMDB
imdb_tndb_bom_df = tndb_bom_df.merge(imdb_df_filtered, how='inner', left_on = ['movie',
print("Total records in joined IMDB, TNDB, and BOM Dataframes", len(imdb_tndb_bom_df))
imdb_tndb_bom_df.head(4)
```

```
Total records in joined IMDB, TNDB, and BOM Dataframes 1025
```

Out[16]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
0	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	$410,600,000	$241,063,875	$1,045,663,875	2011	Pira Carib Str
1	4	2015-05-01	Avengers: Age of Ultron	$330,600,000	$459,005,868	$1,403,013,963	2015	Aver A U
2	7	2018-04-27	Avengers: Infinity War	$300,000,000	$678,815,482	$2,048,134,200	2018	Aver Ir
3	9	2017-11-17	Justice League	$300,000,000	$229,024,295	$655,945,209	2017	Ju Le

In [17]:
```
imdb_tndb_bom_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1025 entries, 0 to 1024
Data columns (total 20 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 1025 non-null   int64
 1   release_date       1025 non-null   datetime64[ns]
 2   movie              1025 non-null   object
 3   production_budget  1025 non-null   object
 4   domestic_gross_x   1025 non-null   object
 5   worldwide_gross    1025 non-null   object
 6   tndb_year          1025 non-null   int64
 7   title              1025 non-null   object
 8   studio             1025 non-null   object
 9   domestic_gross_y   1024 non-null   float64
```

```
 10  foreign_gross      925 non-null    object
 11  year               1025 non-null   int64
 12  movie_id           1025 non-null   object
 13  primary_title      1025 non-null   object
 14  original_title     1025 non-null   object
 15  start_year         1025 non-null   int64
 16  runtime_minutes    1025 non-null   float64
 17  genres             1025 non-null   object
 18  averagerating      1025 non-null   float64
 19  numvotes           1025 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(5), object(11)
memory usage: 168.2+ KB
```

Discussion: Joined Dataframes

Out of 73,856 records in the raw IMDB dataframe, 5,782 in TNDB datframe, and 3,387 in the BOM dataframe, I am left with 1,025 records in the joined dataframe. This is a limitation of the analysis as more data would make a better analysis.

3. Find and Correct Missing Values

Missing values can be in the format of `NaN` or as a placeholder.

In [18]:
```
print("Joined Dataframe Null Values")
imdb_tndb_bom_df.isnull().sum() # add up all the True (1) values to identify null value
```

Out[18]:
```
Joined Dataframe Null Values
id                     0
release_date           0
movie                  0
production_budget      0
domestic_gross_x       0
worldwide_gross        0
tndb_year              0
title                  0
studio                 0
domestic_gross_y       1
foreign_gross        100
year                   0
movie_id               0
primary_title          0
original_title         0
start_year             0
runtime_minutes        0
genres                 0
averagerating          0
numvotes               0
dtype: int64
```

Replace missing `foreign_gross` and `domestic_gross_y` with TNDB's data

There are 100 missing values in the `foreign_gross` column and 1 in the `domestic_gross_y` column derived from the BOM dataframe. For the missing value in the `domestic_gross_y` I can use the data in `domestic_gross_x` derived from the TNDB dataframe. The missing values in `foreign_gross` can be corrected by subtracting the values in `worldwide_gross` from

`domestic_gross_x` . Before this can be repaired `domestic_gross` , `worldwide_gross` , and `foreign_gross` need to be converted to a numerical data type.

Convert `domestic_gross_x` , `worldwide_gross` , `production_budget` , and `foreign_gross` to numerical data type

TNDB's `domestic_gross_x` and `worldwide_gross` is stored as a string data type. I will strip the string, remove the '$' and commas and convert to int data type.

In [19]:
```
# remove '$' and ',' from production_budget, domestic_gross_x, and worldwide_gross
imdb_tndb_bom_df['production_budget'] = imdb_tndb_bom_df['production_budget'].str.repla
imdb_tndb_bom_df['production_budget'] = imdb_tndb_bom_df['production_budget'].str.repla
imdb_tndb_bom_df['domestic_gross_x'] = imdb_tndb_bom_df['domestic_gross_x'].str.replace
imdb_tndb_bom_df['domestic_gross_x'] = imdb_tndb_bom_df['domestic_gross_x'].str.replace
imdb_tndb_bom_df['worldwide_gross'] = imdb_tndb_bom_df['worldwide_gross'].str.replace('
imdb_tndb_bom_df['worldwide_gross'] = imdb_tndb_bom_df['worldwide_gross'].str.replace('

# Convert to int datatype
imdb_tndb_bom_df['production_budget'] = imdb_tndb_bom_df['production_budget'].astype('i
imdb_tndb_bom_df['domestic_gross_x'] = imdb_tndb_bom_df['domestic_gross_x'].astype('int
imdb_tndb_bom_df['worldwide_gross'] = imdb_tndb_bom_df['worldwide_gross'].astype('int64

imdb_tndb_bom_df.head()
```

```
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/2456271369.py:3: FutureWarning: The de
fault value of regex will change from True to False in a future version. In addition, si
ngle character regular expressions will *not* be treated as literal strings when regex=T
rue.
  imdb_tndb_bom_df['production_budget'] = imdb_tndb_bom_df['production_budget'].str.repl
ace('$', '')
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/2456271369.py:5: FutureWarning: The de
fault value of regex will change from True to False in a future version. In addition, si
ngle character regular expressions will *not* be treated as literal strings when regex=T
rue.
  imdb_tndb_bom_df['domestic_gross_x'] = imdb_tndb_bom_df['domestic_gross_x'].str.replac
e('$', '')
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/2456271369.py:7: FutureWarning: The de
fault value of regex will change from True to False in a future version. In addition, si
ngle character regular expressions will *not* be treated as literal strings when regex=T
rue.
  imdb_tndb_bom_df['worldwide_gross'] = imdb_tndb_bom_df['worldwide_gross'].str.replace
('$', '')
```

Out[19]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
0	2	2011-05-20	Pirates of the Caribbean: On Stranger Tides	410600000	241063875	1045663875	2011	Pira Carib Str
1	4	2015-05-01	Avengers: Age of Ultron	330600000	459005868	1403013963	2015	Ave A l
2	7	2018-04-27	Avengers: Infinity War	300000000	678815482	2048134200	2018	Ave I

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
3	9	2017-11-17	Justice League	300000000	229024295	655945209	2017	J Le
4	10	2015-11-06	Spectre	300000000	200074175	879620923	2015	Sp

Converting datatype of `foreign_gross`

`foreign_gross` is listed as an object data type which means it must contain strings and integers in that column. As shown in the table above for *Aveengers: Infinity War*, 1,369.5 is listed as the foreign gross revenue. This value is in the billions so the comma (',') needs to be stripped out and converted to the correct value.

In [20]:
```
# Extract values in foreign_gross that have commas
imdb_tndb_bom_df[imdb_tndb_bom_df['foreign_gross'].str.contains(',', na=False)]
```

Out[20]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
2	7	2018-04-27	Avengers: Infinity War	300000000	678815482	2048134200	2018	Aver In
15	23	2017-04-14	The Fate of the Furious	250000000	225764765	1234846267	2017	The Fu
21	34	2015-06-12	Jurassic World	215000000	652270625	1648854864	2015	Ju V
39	67	2015-04-03	Furious 7	190000000	353007020	1518722794	2015	Furi

There are only 4 records with comma in the `foreign_gross` column. Investigating these movies shows that these values represent billions of dollars in revenue so 1,369.5 for *Avenger's Infinity War* is actually 1,369,500,000 in foreign gross revenue.

I will strip out the comma, convert to a `float` data type and multiply by 1,000,000.

In [21]:
```
# Fix `foreign_gross` data that was in billions
foreign_gross_to_replace = imdb_tndb_bom_df[imdb_tndb_bom_df['foreign_gross'].str.conta
                                            ['foreign_gross'].str.replace(',', '').astype('
imdb_tndb_bom_df.loc[imdb_tndb_bom_df['foreign_gross'].str.contains(',', na=False), 'fo

# Convert all data in foreign_gross column to float64
imdb_tndb_bom_df['foreign_gross'] = imdb_tndb_bom_df['foreign_gross'].astype('float64')
```

In [22]:
```
# Check if was replaced correctly
imdb_tndb_bom_df[imdb_tndb_bom_df['title'] == 'Avengers: Infinity War']['foreign_gross'
```

Out[22]:
```
2    1.369500e+09
Name: foreign_gross, dtype: float64
```

Replace the missing domestic_gross and foreign_gross data from BOM with the data from TNDB

In [23]:
```
# Any records with missing values in `domestic_gross_y` and `foreign_gross`?
statement = 'Missing domestic_gross_y & foreign_gross: '
print(statement, len(imdb_tndb_bom_df[(imdb_tndb_bom_df['domestic_gross_y'].isna() ) &

# Replace missing domestic_gross_y data
imdb_tndb_bom_df.loc[imdb_tndb_bom_df['domestic_gross_y'].isna(), 'domestic_gross_y'] =

# Replace missing foreign_gross data
worldwide_replace = imdb_tndb_bom_df[imdb_tndb_bom_df['foreign_gross'].isna()]['worldwi
domestic_replace = imdb_tndb_bom_df[imdb_tndb_bom_df['foreign_gross'].isna()]['domestic
imdb_tndb_bom_df.loc[imdb_tndb_bom_df['foreign_gross'].isna(), 'foreign_gross'] = world
```

```
Missing domestic_gross_y & foreign_gross:  0
```

In [24]:
```
print("Joined Dataframe Null Values")
imdb_tndb_bom_df.isnull().sum() # add up all the True (1) values to identify null value
```

Out[24]:
```
Joined Dataframe Null Values
id                   0
release_date         0
movie                0
production_budget    0
domestic_gross_x     0
worldwide_gross      0
tndb_year            0
title                0
studio               0
domestic_gross_y     0
foreign_gross        0
year                 0
movie_id             0
primary_title        0
original_title       0
start_year           0
runtime_minutes      0
genres               0
averagerating        0
numvotes             0
dtype: int64
```

4. Check for placeholders

Locate Top Occuring Values

Some databases have placeholders instead of null or NaN values. I will search for top occuring values to see if placeholders or data artifacts that indicate placeholders.

In [25]:
```
# Look for top occuring values
print('Joined Dataframe\n')
for col in imdb_tndb_bom_df.columns:
    print(col, '\n', imdb_tndb_bom_df[col].value_counts(normalize = True).head(10), '\n
```

```
Joined Dataframe
```

```
id
 64    0.016585
60    0.016585
29    0.015610
62    0.014634
78    0.014634
34    0.014634
70    0.014634
14    0.014634
51    0.014634
36    0.013659
Name: id, dtype: float64

release_date
 2010-10-08    0.007805
2014-10-10    0.006829
2016-06-24    0.005854
2011-09-30    0.005854
2013-12-25    0.005854
2011-09-23    0.005854
2011-10-28    0.004878
2011-07-29    0.004878
2011-10-21    0.004878
2015-10-16    0.004878
Name: release_date, dtype: float64

movie
 Pirates of the Caribbean: On Stranger Tides    0.000976
Life of the Party                               0.000976
Sausage Party                                   0.000976
The Crazies                                     0.000976
The Switch                                      0.000976
Leap Year                                       0.000976
The Book Thief                                  0.000976
Take Me Home Tonight                            0.000976
Won't Back Down                                 0.000976
Neighbors                                       0.000976
Name: movie, dtype: float64

production_budget
 20000000    0.042927
40000000    0.041951
30000000    0.037073
10000000    0.036098
35000000    0.033171
25000000    0.029268
15000000    0.029268
5000000     0.029268
50000000    0.028293
60000000    0.020488
Name: production_budget, dtype: float64

domestic_gross_x
 0            0.002927
241063875    0.000976
42469946     0.000976
100292856    0.000976
97670358     0.000976
39123589     0.000976
27758465     0.000976
```

```
25918920     0.000976
21488481     0.000976
6928068      0.000976
Name: domestic_gross_x, dtype: float64

worldwide_gross
 1045663875    0.000976
65759911      0.000976
141344255     0.000976
56445534      0.000976
49858465      0.000976
32618920      0.000976
76086711      0.000976
7576604       0.000976
5745503       0.000976
270944428     0.000976
Name: worldwide_gross, dtype: float64

tndb_year
 2011    0.134634
2010    0.132683
2016    0.118049
2012    0.115122
2013    0.113171
2015    0.112195
2014    0.106341
2018    0.084878
2017    0.082927
Name: tndb_year, dtype: float64

title
 Pirates of the Caribbean: On Stranger Tides    0.000976
Life of the Party                               0.000976
Sausage Party                                   0.000976
The Crazies                                     0.000976
The Switch                                      0.000976
Leap Year                                       0.000976
The Book Thief                                  0.000976
Take Me Home Tonight                            0.000976
Won't Back Down                                 0.000976
Neighbors                                       0.000976
Name: title, dtype: float64

studio
 Uni.       0.109268
Fox        0.100488
WB         0.092683
BV         0.068293
Sony       0.066341
Par.       0.064390
LGF        0.051707
FoxS       0.040976
WB (NL)    0.036098
Focus      0.031220
Name: studio, dtype: float64

domestic_gross_y
 1800000.0     0.006829
6900000.0     0.004878
35100000.0    0.004878
```

```
59700000.0    0.003902
6700000.0     0.003902
23200000.0    0.003902
44900000.0    0.003902
3100000.0     0.003902
5700000.0     0.003902
37700000.0    0.003902
Name: domestic_gross_y, dtype: float64

foreign_gross
 4200000.0     0.006829
6300000.0     0.004878
11300000.0    0.004878
6600000.0     0.003902
1200000.0     0.003902
21900000.0    0.003902
5200000.0     0.003902
58400000.0    0.003902
4300000.0     0.003902
1100000.0     0.002927
Name: foreign_gross, dtype: float64

year
 2011    0.134634
2010    0.132683
2016    0.118049
2012    0.115122
2013    0.113171
2015    0.112195
2014    0.106341
2018    0.084878
2017    0.082927
Name: year, dtype: float64

movie_id
 tt1298650    0.000976
tt5619332    0.000976
tt1700841    0.000976
tt0455407    0.000976
tt0889573    0.000976
tt1216492    0.000976
tt0816442    0.000976
tt0810922    0.000976
tt1870529    0.000976
tt2004420    0.000976
Name: movie_id, dtype: float64

primary_title
 Pirates of the Caribbean: On Stranger Tides    0.000976
Life of the Party                               0.000976
Sausage Party                                   0.000976
The Crazies                                     0.000976
The Switch                                      0.000976
Leap Year                                       0.000976
The Book Thief                                  0.000976
Take Me Home Tonight                            0.000976
Won't Back Down                                 0.000976
Neighbors                                       0.000976
Name: primary_title, dtype: float64
```

```
original_title
 Pirates of the Caribbean: On Stranger Tides    0.000976
Life of the Party                               0.000976
Sausage Party                                   0.000976
The Crazies                                     0.000976
The Switch                                      0.000976
Leap Year                                       0.000976
The Book Thief                                  0.000976
Take Me Home Tonight                            0.000976
Won't Back Down                                 0.000976
Neighbors                                       0.000976
Name: original_title, dtype: float64

start_year
 2011    0.134634
2010    0.132683
2016    0.118049
2012    0.115122
2013    0.113171
2015    0.112195
2014    0.106341
2018    0.084878
2017    0.082927
Name: start_year, dtype: float64

runtime_minutes
 107.0    0.030244
105.0    0.029268
100.0    0.028293
103.0    0.028293
106.0    0.028293
101.0    0.028293
92.0     0.026341
102.0    0.025366
98.0     0.025366
109.0    0.025366
Name: runtime_minutes, dtype: float64

genres
 Adventure,Animation,Comedy    0.058537
Comedy,Drama,Romance          0.041951
Action,Adventure,Sci-Fi       0.040976
Comedy                        0.035122
Comedy,Drama                  0.032195
Comedy,Romance                0.027317
Drama                         0.024390
Drama,Romance                 0.024390
Action,Adventure,Fantasy      0.024390
Horror,Mystery,Thriller       0.023415
Name: genres, dtype: float64

averagerating
 6.3    0.059512
6.6    0.052683
6.2    0.048780
7.0    0.047805
6.4    0.046829
6.5    0.043902
6.8    0.041951
7.1    0.040976
```

```
7.2    0.040000
6.7    0.037073
Name: averagerating, dtype: float64

numvotes
 38667     0.001951
447624    0.000976
20932     0.000976
104465    0.000976
90661     0.000976
86125     0.000976
119023    0.000976
48393     0.000976
5915      0.000976
266020    0.000976
Name: numvotes, dtype: float64
```

Placeholder of 0

A placeholder of 0 was in `domestic_gross_x` which is derived from the TNDB data. Instead of null values, a $0 placeholder may have been used. I will be using `domestic_gross_y` derived from the BOM data and not `domestic_gross_y` from TNDB so this placeholder can be ignored as `domestic_gross_x` will be dropped.

I want to check if there are $0 placeholders for `domestic_gross_y` that is derived from the BOM data. The results show that there is only one which did not have a domestic release.

In [26]:
```
imdb_tndb_bom_df[imdb_tndb_bom_df['domestic_gross_y'] == 0]
```

Out[26]:

	id	release_date	movie	production_budget	domestic_gross_x	worldwide_gross	tndb_year	
847	36	2010-10-08	It's a Wonderful Afterlife	10000000	0	1642939	2010	W

5. Remove Irrelevant Columns

The following columns will be kept for the dataframe that will be analyzed

- `title`
- `release_date`
- `start_year`
- `production_budget`
- `domestic_gross_y`
- `foreign_gross`
- `genres`
- `averagerating`
- `numvotes`

In [27]:
```
# Drop the columns not being used
```

```
analysis_df = imdb_tndb_bom_df[imdb_tndb_bom_df.columns.drop(['id', 'movie', 'studio',
print(analysis_df.info())
analysis_df.head(3)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1025 entries, 0 to 1024
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   release_date       1025 non-null   datetime64[ns]
 1   production_budget  1025 non-null   int64
 2   title              1025 non-null   object
 3   domestic_gross_y   1025 non-null   float64
 4   foreign_gross      1025 non-null   float64
 5   start_year         1025 non-null   int64
 6   genres             1025 non-null   object
 7   averagerating      1025 non-null   float64
 8   numvotes           1025 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(3), object(2)
memory usage: 80.1+ KB
None
```

Out[27]:

	release_date	production_budget	title	domestic_gross_y	foreign_gross	start_year	
0	2011-05-20	410600000	Pirates of the Caribbean: On Stranger Tides	241100000.0	8.046000e+08	2011	Action,Adven
1	2015-05-01	330600000	Avengers: Age of Ultron	459000000.0	9.464000e+08	2015	Action,Adv
2	2018-04-27	300000000	Avengers: Infinity War	678800000.0	1.369500e+09	2018	Action,Adv

In [28]:

```
# Summary statistics of TNDB dataframe
analysis_df.describe().apply(lambda s: s.apply('{0:.2f}'.format))
```

Out[28]:

	production_budget	domestic_gross_y	foreign_gross	start_year	averagerating	numvotes
count	1025.00	1025.00	1025.00	1025.00	1025.00	1025.00
mean	52763544.05	69586337.84	105851111.34	2013.66	6.46	146317.85
std	58982785.69	89888424.34	170359756.43	2.55	0.94	178041.96
min	50000.00	0.00	0.00	2010.00	1.60	24.00
25%	13000000.00	14300000.00	8600000.00	2011.00	5.90	37939.00
50%	30000000.00	39300000.00	36400000.00	2014.00	6.50	86118.00
75%	68000000.00	84800000.00	115700000.00	2016.00	7.10	181189.00
max	410600000.00	700100000.00	1369500000.00	2018.00	8.80	1841066.00

6. Calculate worldwide gross revenue and profit estimate

In [29]:
```
# Calculate worldwide gross revenue
analysis_df['worldwide_gross'] = analysis_df['domestic_gross_y'] + analysis_df['foreign

# Calculate estimated profit
analysis_df['profit_estimate'] = analysis_df['worldwide_gross'] - analysis_df['producti
analysis_df.head()

analysis_df.head(3)
```

```
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/113827421.py:2: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  analysis_df['worldwide_gross'] = analysis_df['domestic_gross_y'] + analysis_df['foreig
n_gross']
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/113827421.py:5: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  analysis_df['profit_estimate'] = analysis_df['worldwide_gross'] - analysis_df['product
ion_budget']
```

Out[29]:

	release_date	production_budget	title	domestic_gross_y	foreign_gross	start_year	
0	2011-05-20	410600000	Pirates of the Caribbean: On Stranger Tides	241100000.0	8.046000e+08	2011	Action,Adven
1	2015-05-01	330600000	Avengers: Age of Ultron	459000000.0	9.464000e+08	2015	Action,Adv
2	2018-04-27	300000000	Avengers: Infinity War	678800000.0	1.369500e+09	2018	Action,Adv

7. Convert Genre string to list of strings

The information in the `genres` column is a long string with genres separated by a comma (,). I will convert this to a list of strings by splitting the string by the comma and appending each to a list that will be stored in the `genres` column.

In [30]:
```
# Convert Genres into a list

# Convert all records in `genres` to string
```

```
analysis_df['genres'] = analysis_df['genres'].apply(str)

# Function to take a column and split the string by commas
def split_genres(column):
    # coerce to string
    return column.split(',')

analysis_df['genres'] = analysis_df['genres'].apply(split_genres)
analysis_df.head(3)
```

```
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/602391359.py:4: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  analysis_df['genres'] = analysis_df['genres'].apply(str)
C:\Users\kevin\AppData\Local\Temp/ipykernel_39572/602391359.py:11: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  analysis_df['genres'] = analysis_df['genres'].apply(split_genres)
```

Out[30]:

	release_date	production_budget	title	domestic_gross_y	foreign_gross	start_year	genres
0	2011-05-20	410600000	Pirates of the Caribbean: On Stranger Tides	241100000.0	8.046000e+08	2011	[Action, Adventure, Fantasy]
1	2015-05-01	330600000	Avengers: Age of Ultron	459000000.0	9.464000e+08	2015	[Action, Adventure, Sci-Fi]
2	2018-04-27	300000000	Avengers: Infinity War	678800000.0	1.369500e+09	2018	[Action, Adventure, Sci-Fi]

Data Analysis

What is the total number of movies in this analysis?

In [32]:

```
print('Total number of movies in TNDB data:', len(tndb_df))
print('Total number of movies in BOM data:', len(bom_df))
print('Total number of movies in filtered IMDB data:', len(imdb_df_filtered))
print('Total number of movies in this analysis:', len(analysis_df))
```

```
Total number of movies in TNDB data: 5782
Total number of movies in BOM data: 3387
```

```
Total number of movies in filtered IMDB data: 73271
Total number of movies in this analysis: 1025
```

What is the timespan of the movies in this data?

In [33]:

```
min_release_date = round(analysis_df['start_year'].min())
max_release_date = round(analysis_df['start_year'].max())
print(min_release_date, ' to ', max_release_date)
```

```
2010  to  2018
```

What is the mean and median ranking for all movies?

In [34]:

```
print('Mean IMDb ranking:', round(analysis_df['averagerating'].mean(), 1))
print('Median IMDb ranking:', analysis_df['averagerating'].median())
```

```
Mean IMDb ranking: 6.5
Median IMDb ranking: 6.5
```

What is the distribution of movie rankings?

In [31]:

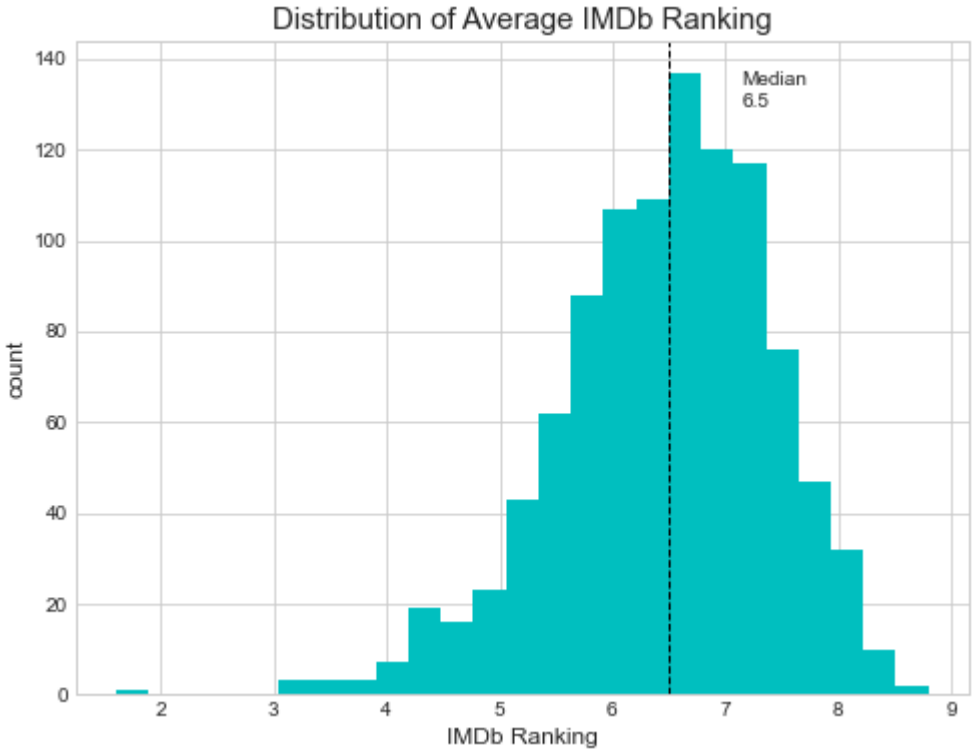
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (8, 6)
plt.style.use('seaborn-whitegrid')
```

In [35]:

```
# Make Plot
analysis_df.hist(column = 'averagerating', bins=25, color = 'c')

# Set x, y-axis labels and title
plt.ylabel('count', fontsize = 12)
plt.xlabel('IMDb Ranking', fontsize = 12)
plt.title('Distribution of Average IMDb Ranking', fontdict={'fontsize': 15})

# Draw a verticle line and post the median
min_ylim, max_ylim = plt.ylim()
x = imdb_tndb_bom_df['averagerating']
plt.axvline(x.median(), color='k', linestyle='dashed', linewidth=1) # Makes verticle li
plt.text(x.median()*1.1, max_ylim*0.9, 'Median\n{:.1f}'.format(x.median())) # Adds text

#plt.savefig('./img/fig/hist_ranking.png', bbox_inches = 'tight')
```

Out[35]:

```
Text(7.15, 129.465, 'Median\n6.5')
```



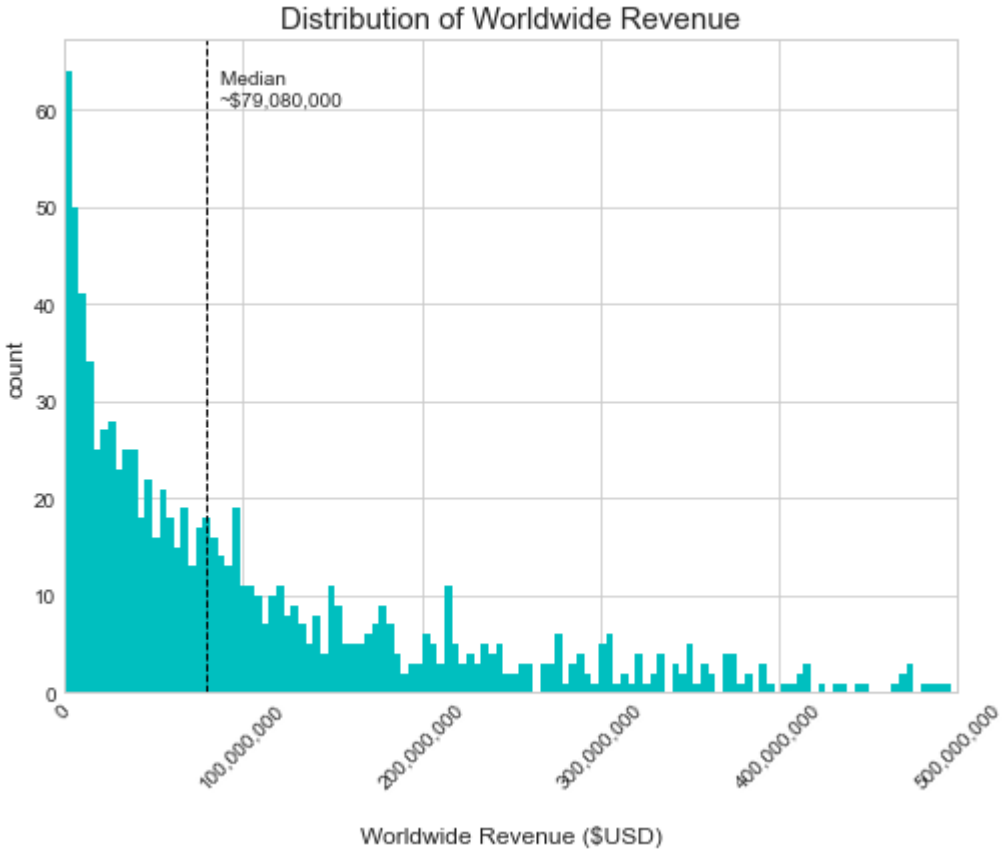
What is the distribution of worldwide movie revenue?

In [36]:

```
print('Median Worldwide Gross Revenue:', round(imdb_tndb_bom_df['worldwide_gross'].medi

# Make PLot
analysis_df.hist(column = 'worldwide_gross', bins=500, color = 'c')

# Set x, y-axis labels and title
plt.ylabel('count', fontsize = 12)
plt.xlabel('\nWorldwide Revenue ($USD)', fontsize = 12)
plt.title('Distribution of Worldwide Revenue', fontdict={'fontsize': 15})

# Set x-axis limit because heavilty left-skewed
plt.xlim(xmin = 0, xmax = 500000000)

# Set x-axis to have normal ticks and not exponential
plt.gca().xaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'
# reference Weiner, Iddo @ https://stackoverflow.com/a/70272061/1144724
plt.xticks(rotation = 45)

# Draw a verticle line and post the median
min_ylim, max_ylim = plt.ylim()
x = imdb_tndb_bom_df['worldwide_gross']
plt.axvline(x.median(), color='k', linestyle='dashed', linewidth=1) # Makes verticle li
plt.text(x.median()*1.1, max_ylim*0.9, 'Median \n~${:,.0f}'.format(round(x.median(), -4
# Reference: Porripeikko @ https://stackoverflow.com/a/52961228/1144724

# Save plot
#plt.savefig('./img/fig/hist_worldwide_revenue.png', bbox_inches = 'tight')
```

Out[36]:

```
Median Worldwide Gross Revenue: 79100000.0
Text(86984345.80000001, 60.480000000000004, 'Median \n~$79,080,000')
```



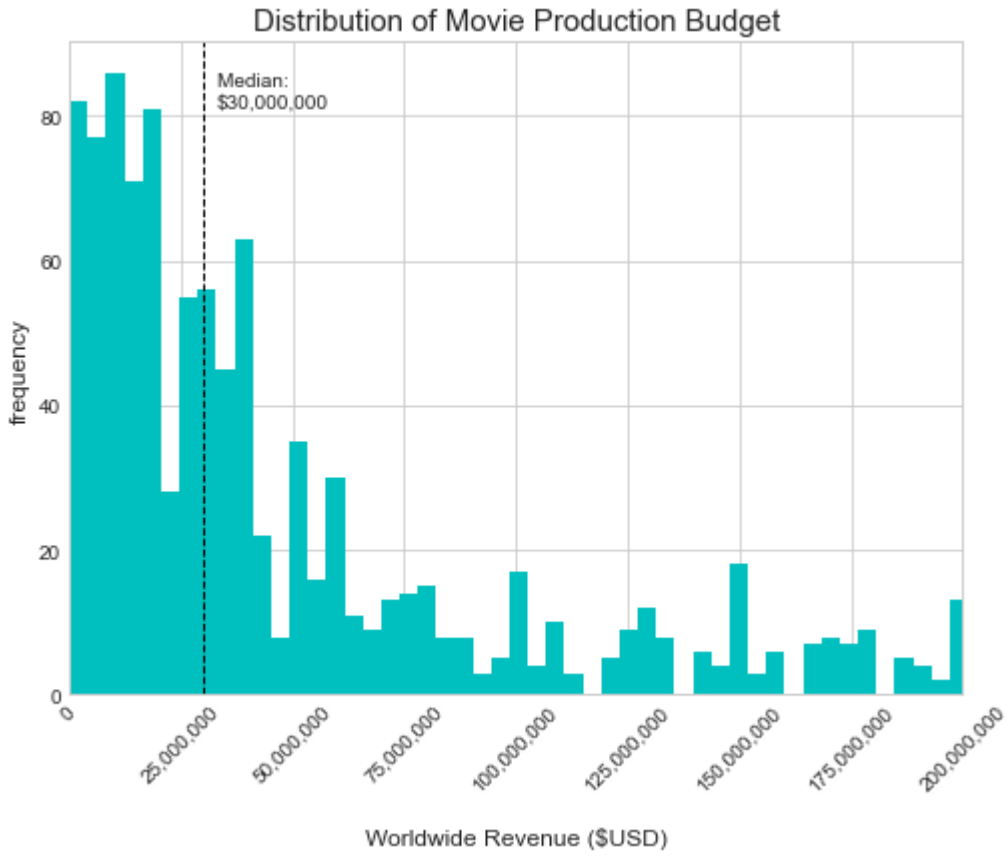
What is the distribution of movie production budget?

In [37]:

```
# Return and print median
print('Median Production Budget:', round(imdb_tndb_bom_df['production_budget'].median()

# Make plot
analysis_df.hist(column = 'production_budget', bins=100, color = 'c')

# Set x, y-axis labels and title
plt.ylabel('frequency', fontsize = 12)
plt.xlabel('\nWorldwide Revenue ($USD)', fontsize = 12)
plt.title('Distribution of Movie Production Budget', fontdict={'fontsize': 15})

# Set x-axis limit because heavilty left-skewed
plt.xlim(xmin = 0, xmax = 200000000)

# Set x-axis to have normal ticks and not exponential
plt.gca().xaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'
plt.xticks(rotation = 45)

# Draw a verticle line and post the median
min_ylim, max_ylim = plt.ylim()
x = imdb_tndb_bom_df['production_budget']
plt.axvline(x.median(), color='k', linestyle='dashed', linewidth=1) # Makes verticle li
plt.text(x.median()*1.1, max_ylim*0.9, 'Median: \n${:,.0f}'.format(round(x.median())))

#plt.savefig('./img/fig/hist_production_budget.png', bbox_inches = 'tight')
```

Out[37]:

```
Median Production Budget: 30000000
Text(33000000.000000004, 81.27, 'Median: \n$30,000,000')
```



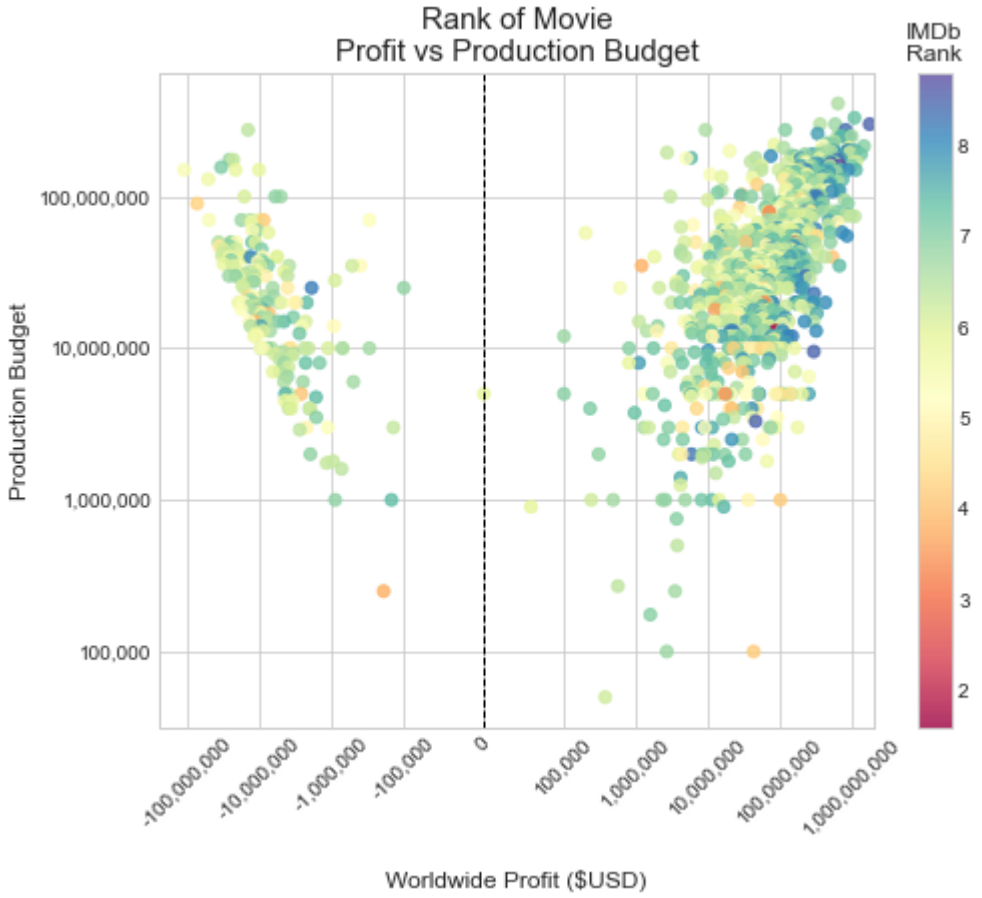
Does production budget matter for profitable and highly ranked movies?

In [38]:

```
# scatterplot of Budget versus Revenue with grouped by ranking

# How many movies made a profit?
print('Percent of movies that made a profit', (len(analysis_df[analysis_df['profit_esti

# Gather required data
x = analysis_df['profit_estimate']
y = analysis_df['production_budget']
colors = analysis_df['averagerating']

# Make figure
fig2 = plt.figure()
scatter_plot = plt.scatter(x, y, c = colors, alpha = 0.8, cmap = 'Spectral')

# Add x-axis
plt.xscale('symlog', linthresh = 100000)
plt.xlim((-250000000,2e+09))
plt.xlabel('\nWorldwide Profit ($USD)', fontsize = 12)
plt.title('Rank of Movie\nProfit vs Production Budget', fontdict={'fontsize': 15})
plt.gca().xaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'
plt.xticks(rotation = 45)

# y-axis
plt.yscale('log')
plt.ylabel('Production Budget', fontsize = 12)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'
```

```
# Makes verticle line at 0
plt.axvline(0, color='k', linestyle='dashed', linewidth=1)

# Give colorbar a title
clb = plt.colorbar()
clb.ax.set_title('IMDb\nRank')
#fig2.savefig('./img/fig/scatter_profit_versus_budget.png', bbox_inches = 'tight')
```

```
Percent of movies that made a profit 82.82926829268293
```



Discussion

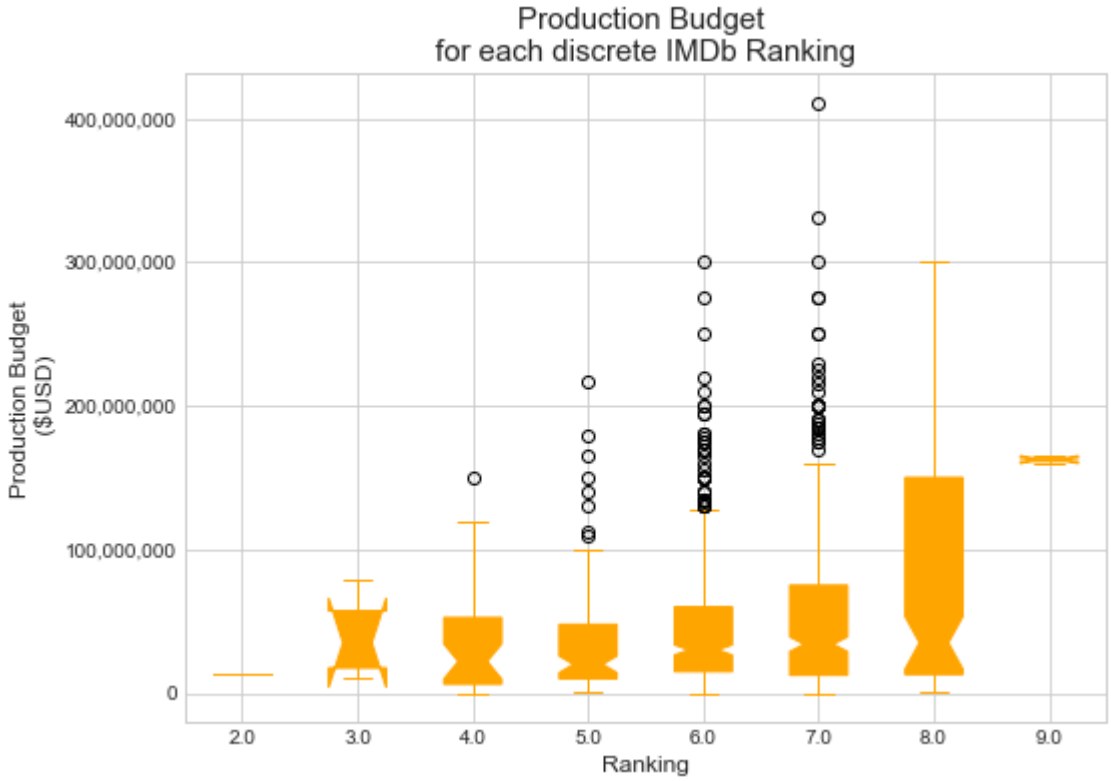
Profit was estimated by subtracting worldwide gross revenue by the production budget. 83% of movies in this analysis generated a profit and 17% did not. There are movies with high IMDb raning (>7) and movies with high production budget (> $100 million) that did not make a profit. Conversly there are some movies with low ranking and low budget that did make a profit.

What is the average budget for each discrete ranking?

In [39]:

```
# Average budget for each ranked value

# Extract necessary data: ranking (averagerating), production_budget
boxplot_ranking_budget = analysis_df.filter(['production_budget', 'averagerating'])

# Turn continuous ranking data into categorical data
boxplot_ranking_budget['averagerating'] = round(boxplot_ranking_budget['averagerating']

# Produce the boxplot
```

```
box = boxplot_ranking_budget.boxplot(column = 'production_budget', by='averagerating',
                                     notch = True, patch_artist=True, color = 'orange')
# Set Title
plt.suptitle('')
plt.title('Production Budget \nfor each discrete IMDb Ranking', fontdict={'fontsize': 1

# x-axis
plt.xlabel('Ranking', fontsize = 12)
plt.xticks(rotation = 0)

# y-axis
plt.ylabel('Production Budget\n ($USD)', fontsize = 12)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'

# Save the boxplot to file
#plt.savefig('./img/fig/boxplot_ranking_budget.png', bbox_inches = 'tight')
```



Discussion

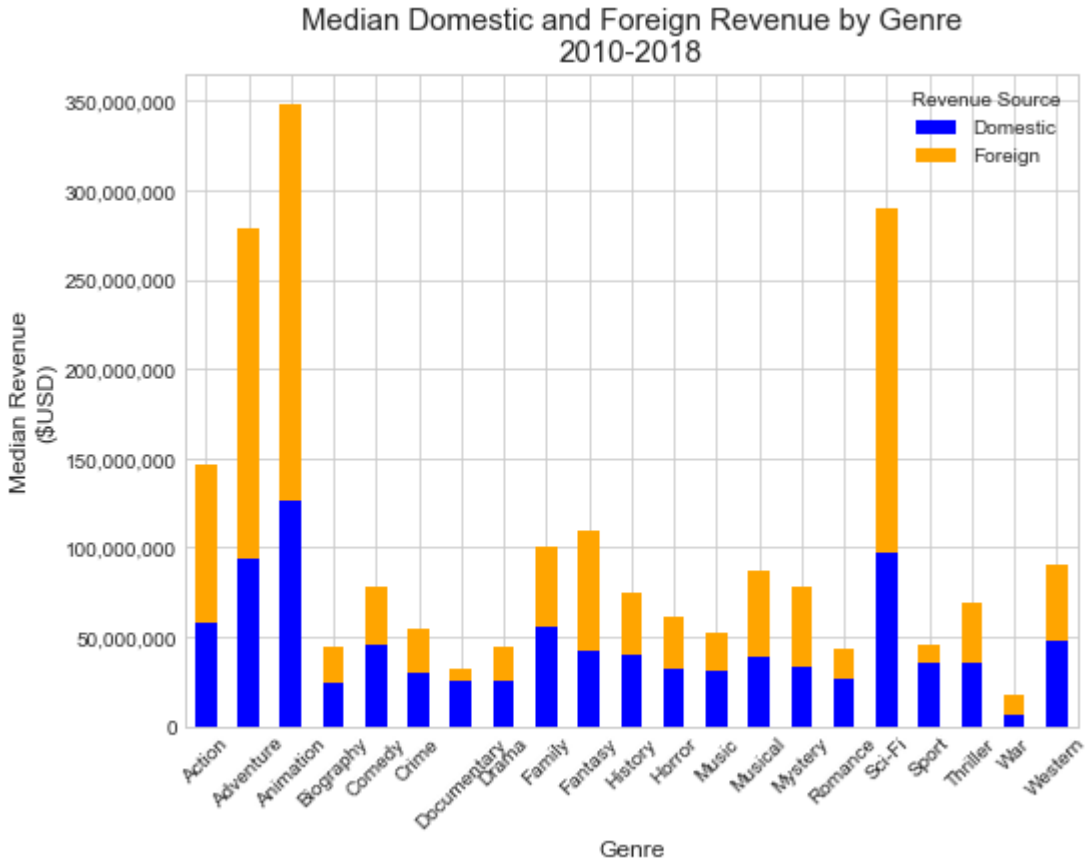
The median production budget for each discrete IMDb ranking is about the same. There are many values outside the IQR for movies ranked between 6 to 8. Movies ranked between 9 to 10 did have a higher median production budget but there are very few movies in this ranking.

What is the median worldwide revenue generated for each genre?

In [40]:

```
# Stacked Barplot showing domestic and foregin revenue grouped by genre
# Reference: jezrael @ https://stackoverflow.com/questions/72272575/plotting-a-stacked-

# filter needed data from imdb_tndb_bom_df
barplot_stacked_df = analysis_df.filter(['domestic_gross_y', 'foreign_gross', 'genres']
df = barplot_stacked_df.explode('genres').melt('genres').pivot_table(index='genres', co
                                                                     values='value', ag
```

```
# Make stacked bar chart
barplot_stacked_revenue = df.plot.bar(stacked=True, color=['blue', 'orange']) # Set col

# Remove legend
plt.legend(['Domestic', 'Foreign'], title = 'Revenue Source')

# x-axis
plt.xlabel('Genre', fontsize = 12)
plt.title('Median Domestic and Foreign Revenue by Genre\n2010-2018', fontdict={'fontsiz
plt.xticks(rotation = 45)

# y-axis
plt.ylabel('Median Revenue\n($USD)', fontsize = 12)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'

# Save plot to file
#plt.savefig('./img/fig/barplot_genre_revenue.png', bbox_inches = 'tight')
```



Discussion

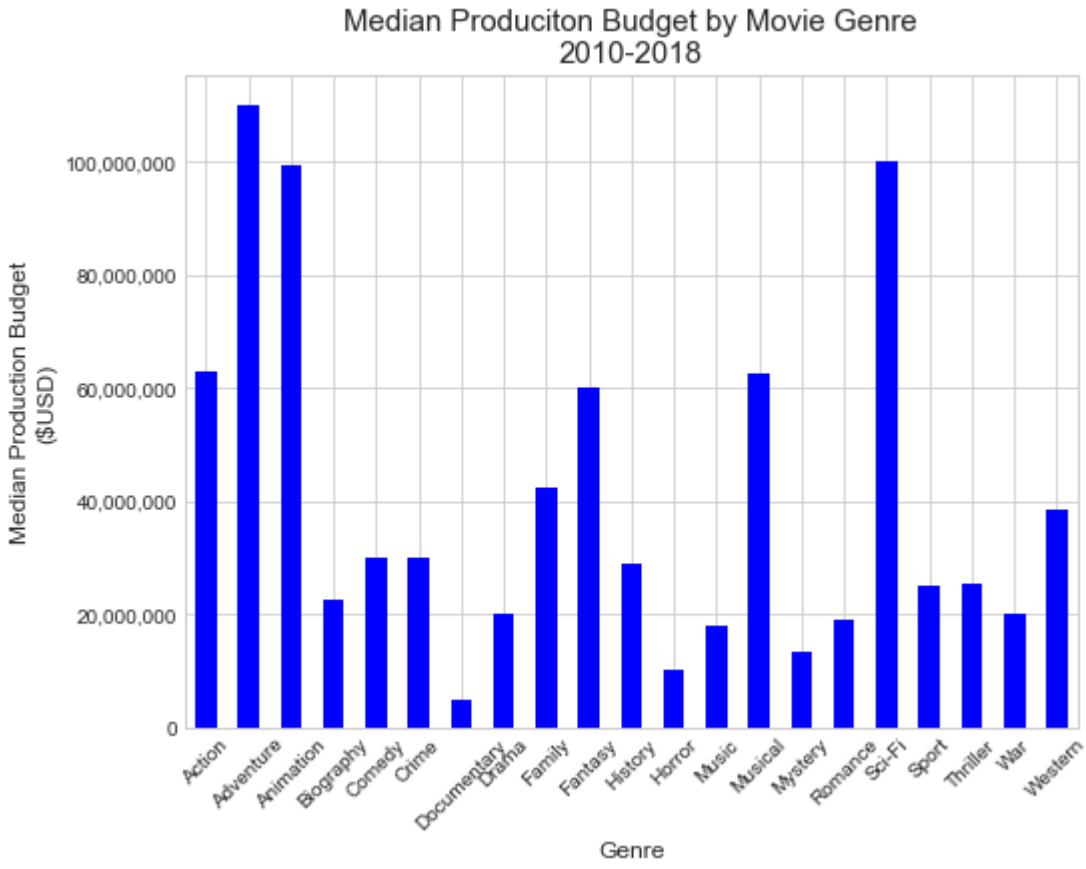
Action, adventure, animation, and sci-fi have the highest total worldwide revenue. A movie with one or more of these genres would be more likely to generate enough revenue to sustain this project.

What is the production budget for each genre?

In [42]:

```
# Box plot showing produciton budget per genre

# Filter necessary variables
barplot_genre_budget = analysis_df.filter(['production_budget', 'genres'])
df = barplot_genre_budget.explode('genres').melt('genres').pivot_table(index='genres',
```

```
# Make stacked bar chart
df.plot.bar(stacked=True, color = 'blue') # Set color https://www.tutorialkart.com/matp

# Set title
plt.title('Median Produciton Budget by Movie Genre\n2010-2018', fontdict={'fontsize': 1

# Remove legend
plt.legend('', frameon = False)

# X-axis
plt.xlabel('Genre', fontsize = 12)
plt.xticks(rotation = 45)

# Y-axis
plt.ylabel('Median Production Budget\n($USD)', fontsize = 12)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'

# Save plot to file
#plt.savefig('./img/fig/barplot_genre_budget.png', bbox_inches = 'tight')
```



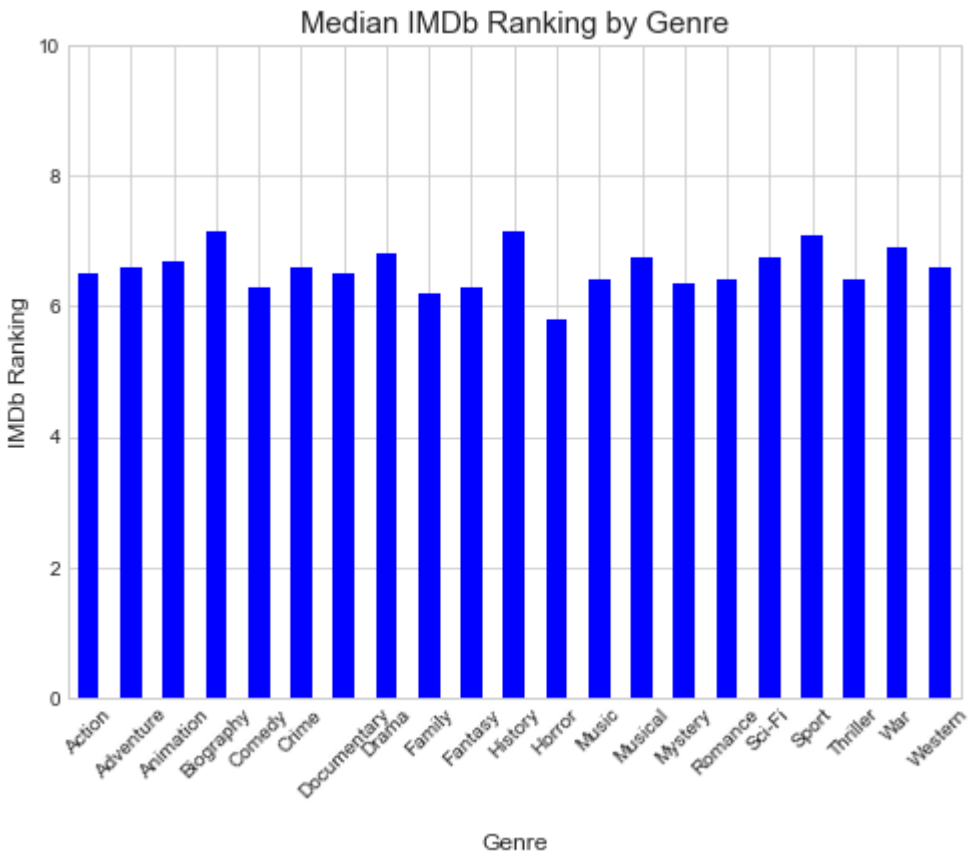
Discussion

Along with having a high worldwide gross revenue, the genres of action, adventure, animation, and science fiction also have the highest production budgets of all the genres. Movies in the action, adventure, animation, and science fiction genres have a heavy use of computer generated imagry (CGI) which would explain the high production budgets.

How does the different genres compare with IMDb ranking?

In [43]:
```
# Boxplot of Genre by ranking

# Group by Genre, let ranking be a continuous variable
barplot_genre_ranking = analysis_df.filter(['genres', 'averagerating'])

df = barplot_genre_ranking.explode('genres').melt('genres').pivot_table(index='genres',
                                                                        values='value',
# Make barplot
df.plot.bar(color = ['blue'])

# Set Title
plt.title('Median IMDb Ranking by Genre', fontdict={'fontsize': 15})

# x-axis
plt.xlabel('\nGenre', fontsize = 12)
plt.xticks(rotation = 45)

# y-axis
plt.ylabel('\nIMDb Ranking', fontsize = 12)
plt.ylim(ymin = 0, ymax = 10)

# Legend
plt.legend('')
#plt.savefig('./img/fig/barplot_genre_ranking.png', bbox_inches = 'tight')
```

Out[43]:
```
<matplotlib.legend.Legend at 0x22ceea82610>
```



Discussion

The genres did not have much difference between them in regards to IMDb user ranking.

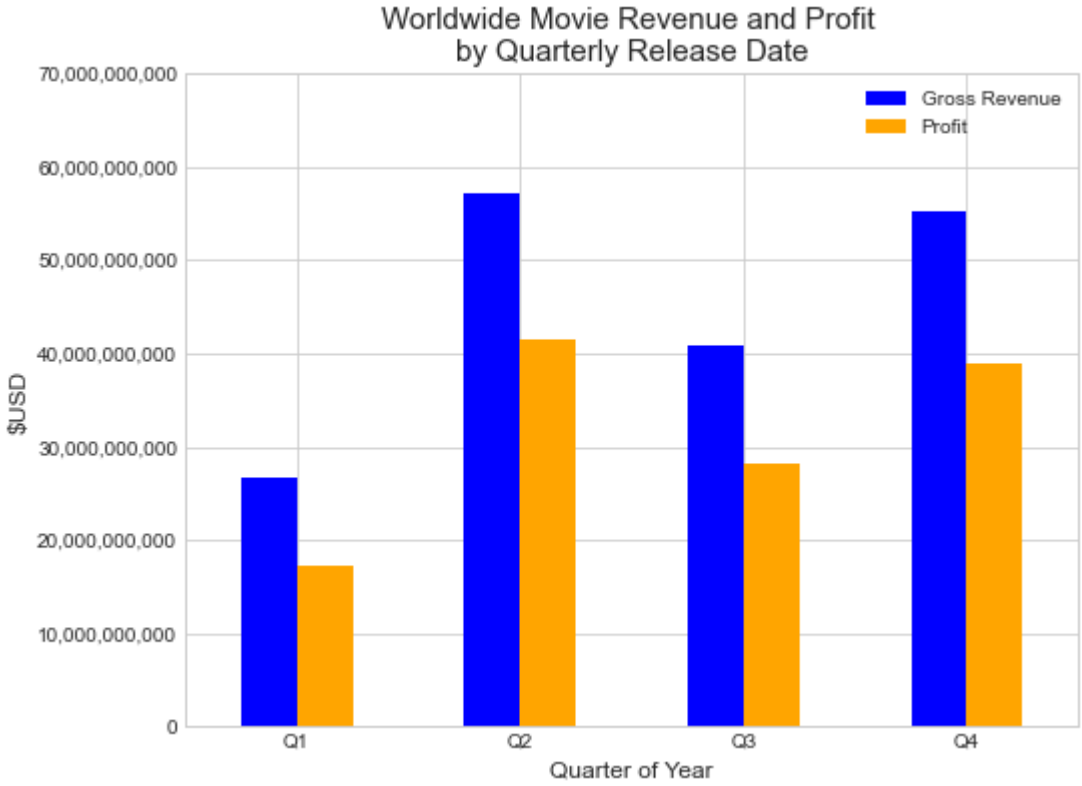
What quarter of the year for movie release has the highest revenue?

In [44]:

```
# Box plot showing total box office revenue and profit with quarter of release

# Extract necessary data: release_date, worldwide_gross
barplot_quarter = analysis_df.filter(['release_date', 'worldwide_gross'])

# convert release_date to quarter of the year
barplot_quarter['release_date'] = barplot_quarter['release_date'].dt.quarter # replace

# remap the quarterly values according to dictionary
quarter_dict = {1: 'Q1', 2: 'Q2', 3: 'Q3', 4: 'Q4'}
barplot_quarter.replace({'release_date': quarter_dict}, inplace = True)

# Make a new dataframe that finds the total worldwide gross revenue by quarter
df1 = barplot_quarter.groupby('release_date')['worldwide_gross'].sum()

# Extra necessary data: release_date, profit_estimate
barplot_quarter_profit = analysis_df.filter(['release_date', 'profit_estimate'])

# convert release_date to quarter of the year
barplot_quarter_profit['release_date'] = barplot_quarter_profit['release_date'].dt.quar
quarter_dict = {1: 'Q1', 2: 'Q2', 3: 'Q3', 4: 'Q4'} # remap dictionary
barplot_quarter_profit.replace({'release_date': quarter_dict}, inplace = True) # remap

# Make a new dataframe that finds the total profit by quarter
df2 = barplot_quarter_profit.groupby('release_date')['profit_estimate'].sum()

# concatenate df1 and df2
df3 = pd.concat([df1, df2], axis=1, join='inner')

# Make the bar plot
df3.plot.bar(color = ['blue', 'orange'])

# Set Title
plt.title('Worldwide Movie Revenue and Profit \nby Quarterly Release Date', fontdict={'

# x-axis
plt.xlabel('Quarter of Year', fontsize = 12)
plt.xticks(rotation = 0)

# y-axis
plt.ylabel('$USD', fontsize = 12)
plt.ylim(ymin = 0, ymax = 70000000000)
plt.gca().yaxis.set_major_formatter(plt.matplotlib.ticker.StrMethodFormatter('{x:,.0f}'

# Legend
plt.legend(['Gross Revenue', 'Profit'])
#plt.savefig('./img/fig/barplot_comp_quarter.png', bbox_inches = 'tight')
```

Out[44]:

```
<matplotlib.legend.Legend at 0x22ceea9a640>
```



Discussion

Worldwide gross revenue is highest when movies are released in the second and fourth quarters of the year. This is during the time when people are on holiday and working less. They have time to go to movie theaters to see a movie.

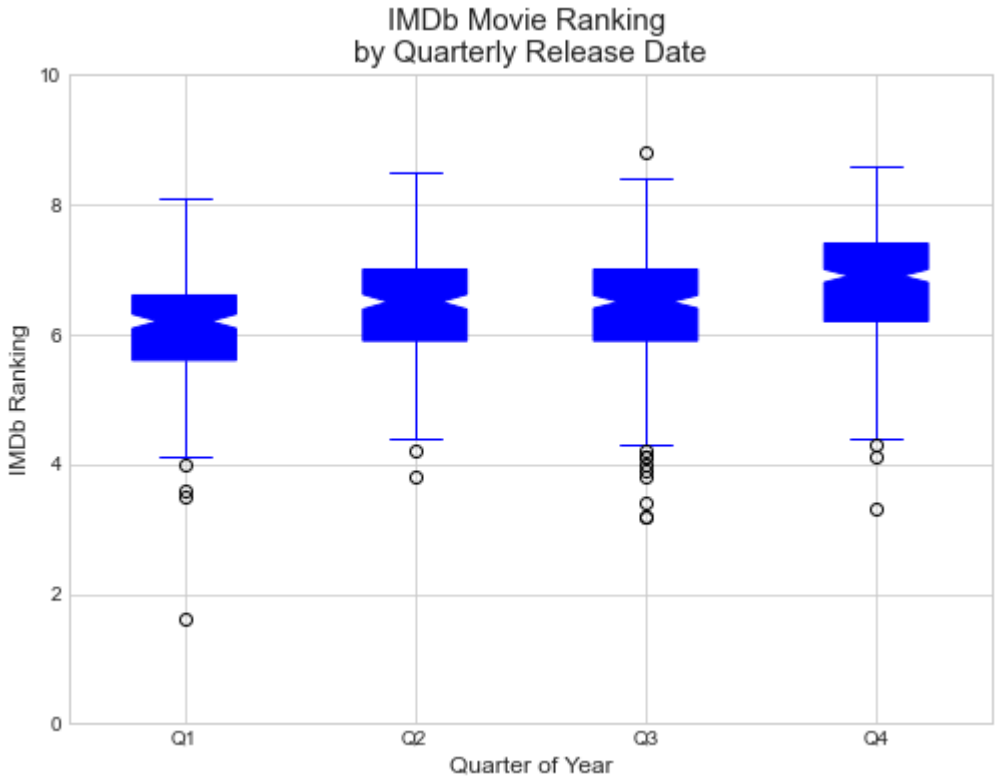
Does rankings of movies change by quarter of the year it is released?

In [45]:

```
# Ranking by quarterly release
## Reference: https://stackoverflow.com/a/40359047/1144724

# Extract necessary data
# Ranking (averagerating), datetime (release_date)
boxplot_quarter_ranking = analysis_df.filter(['release_date', 'averagerating'])

# Convert datetime to quarterly
boxplot_quarter_ranking['release_date'] = boxplot_quarter_ranking['release_date'].dt.qu
quarter_dict = {1: 'Q1', 2: 'Q2', 3: 'Q3', 4: 'Q4'} # remap dictionary
boxplot_quarter_ranking.replace({'release_date': quarter_dict}, inplace = True) # remap

# Produce the boxplot
box = boxplot_quarter_ranking.boxplot(column = 'averagerating', by='release_date', \
                                      notch = True, patch_artist=True, color = 'blue')
# Set Title
plt.suptitle('')
plt.title('IMDb Movie Ranking \nby Quarterly Release Date', fontdict={'fontsize': 15})

# x-axis
plt.xlabel('Quarter of Year', fontsize = 12)
plt.xticks(rotation = 0)
```

```
# y-axis
plt.ylabel('IMDb Ranking', fontsize = 12)
plt.ylim(ymin = 0, ymax = 10)

# Save the boxplot to file
#plt.savefig('./img/fig/boxplot_quarter_ranking.png', bbox_inches = 'tight')
```

Out[45]: (0.0, 10.0)



In [46]:
```
# print median of each quarter
q_median_rank = boxplot_quarter_ranking.groupby('release_date').median()
q_median_rank
```

Out[46]:

release_date	averagerating
Q1	6.2
Q2	6.5
Q3	6.5
Q4	6.9

Discussion

IMDb ranking of movies based on the quarter of the year they are released are all around 6.5 with Q1 being slightly lower and Q4 being slightly higher.

Conclusion

Microsoft is creating a new movie studio and wants to know what types of movies they should create to be successful. Success can be measured with the amount of profit that the movie makes and the reputation as measured in viewer ranking. Data is from the Internet Movie Database (IMDb), which contains IMDb user generated rankings, Box Office Mojo (BOM), which contains domestic and foreign revenue, and The Numbers (TNDB), that contains movie production budget. This data was cleaned and joined to provide analysis of 1,025 movies released betwen 2010 to 2018.

To be successful, Microsoft needs to know what movie genres have the highest ranking and profit, when is the best time of the year to release a movie, and how much should be budgeted for these movies.

My analysis shows that the average ranking of these movies is 6.5 out of 10 and does not change much between genres. Half of the movies in the analysis had worldwide gross revenue below $79 million and a production budget below \$30 million. Action, adventure, animation, and science fiction were the top grossing genres with median worldwide box office revenue greater than $100 million. They also had higher produciton budgets as compared to other genres. The 2nd and 4th quarters are associated with higher revenue. This because less people are working or going to school during these times.

Based on this analysis, I recommend that Microsoft focus on action, adventure, animation, and science-fiction movies. They should plan to releaes these movies during the second and fourth quarter of the year and should plan to budget between $60 million to \$100 million on producing each movie.