The Movie Database: 10-Year Quarterly Analysis

Table of Contents

Introduction

> In this project, a subset of data associated with The Movie Dataset found on Kaggle, will be analyzed. The past 10 years of data will be extracted to find which quarter of the year results in the least revenue on average and how the number of movies released affected the revenue.

In [2]:
```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sb
%matplotlib inline
```

Custom Functions

add_quarter: A function for creating a quarter column. The month will be extracted from the `release_date`, divided into yearly quarters, and appended to the selected dataframe in a `quarter` column.

In [2]:
```
def add_quarter(dfname):
    # Get release month
    month = dfname['release_date'].dt.month

    # Bin edges will used to cut the data into quarters
    bin_edges = [0, 3, 6, 9, 12]

    # Bin names will be used to label the quarters
    bin_names = ['Q1', 'Q2', 'Q3','Q4']

    # Create column
    dfname['quarter'] = pd.cut(month, bin_edges, labels=bin_names)

    pass
```

qtr_rev_means: A function for determining the average revenue by quarter. It accepts the dataframe name (`dfname`) as the parameter, and if NaN is present, value is filled with 0.

In [3]:
```
def qtr_rev_means(dfname):
    means = dfname.groupby(['quarter']).revenue.mean()
    result = means.fillna(0)
    return result
```

dtype_convert: A function for simple datatype conversions that operates by collecting the dataframe name (`dfname`), column name (`column`), and desired datatype (`datatype`).

Note: Not to be used for more complex conversions such as datetime.

In [4]:
```
def dtype_convert(dfname, column, datatype):
    dfname['' + column + ''] = dfname['' + column + ''].astype(datatype)
    return dfname.dtypes
```

Data Wrangling

General Properties

In [5]:
```
# Loads The Movie Database dataset
df = pd.read_csv('tmdb-movies.csv')
df.head()
```

Out[5]:

	id	imdb_id	popularity	budget	revenue	original_title	cast
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic...
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel...
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...

5 rows × 21 columns

In [6]:

```
# Get information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    10866 non-null  int64  
 1   imdb_id               10856 non-null  object 
 2   popularity            10866 non-null  float64
 3   budget                10866 non-null  int64  
 4   revenue               10866 non-null  int64  
 5   original_title        10866 non-null  object 
 6   cast                  10790 non-null  object 
 7   homepage              2936 non-null   object 
 8   director              10822 non-null  object 
 9   tagline               8042 non-null   object 
 10  keywords              9373 non-null   object 
 11  overview              10862 non-null  object 
 12  runtime               10866 non-null  int64  
 13  genres                10843 non-null  object 
 14  production_companies  9836 non-null   object 
 15  release_date          10866 non-null  object 
 16  vote_count            10866 non-null  int64  
 17  vote_average          10866 non-null  float64
 18  release_year          10866 non-null  int64  
 19  budget_adj            10866 non-null  float64
 20  revenue_adj           10866 non-null  float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

In [7]:
```
# Get statistics
df.describe()
```

Out[7]:

	id	popularity	budget	revenue	runtime	vote_cou
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.0000
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.3897
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.6190
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.0000
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.0000
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.0000
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.7500
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.0000

Data Cleaning

> In this section, a subset of data that captures the previous 10-years will be created and only include relevant columns (listed below). Missing/null and

> duplicate rows will be investigated and resolved as well as any incorrect data types.

Create the New Dataframe

Capture the previous 10 years of data that only includes known essential columns

In [8]:
```
# Create new data frame
ten_df = df[['id', 'popularity', 'revenue', 'original_title',
       'genres', 'release_date', 'release_year']].query('release_year >= 2006')
ten_df.head()
```

Out[8]:

	id	popularity	revenue	original_title	genres	release_date	rel
0	135397	32.985763	1513528810	Jurassic World	Action|Adventure|Science Fiction|Thriller	6/9/15	
1	76341	28.419936	378436354	Mad Max: Fury Road	Action|Adventure|Science Fiction|Thriller	5/13/15	
2	262500	13.112507	295238201	Insurgent	Adventure|Science Fiction|Thriller	3/18/15	
3	140607	11.173104	2068178225	Star Wars: The Force Awakens	Action|Adventure|Science Fiction|Fantasy	12/15/15	
4	168259	9.335014	1506249360	Furious 7	Action|Crime|Thriller	4/1/15	

In [9]:
```
# Confirm release_year range
ten_df['release_year'].describe()
```

Out[9]:
```
count    5481.000000
mean     2010.942711
std         2.829202
min      2006.000000
25%      2009.000000
50%      2011.000000
75%      2013.000000
max      2015.000000
Name: release_year, dtype: float64
```

In [10]:
```
# View updated info
ten_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5481 entries, 0 to 7824
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              5481 non-null   int64  
 1   popularity      5481 non-null   float64
 2   revenue         5481 non-null   int64  
 3   original_title  5481 non-null   object 
 4   genres          5466 non-null   object 
 5   release_date    5481 non-null   object 
 6   release_year    5481 non-null   int64  
dtypes: float64(1), int64(3), object(3)
memory usage: 342.6+ KB
```

Duplicates

Find and remove duplicate rows

In [11]:
```
# Check for duplicate movies
ten_df.duplicated('id').sum()
```

Out[11]: 1

In [12]:
```
# View duplicated data
ten_df[ten_df['id'].duplicated()]
```

Out[12]:

	id	popularity	revenue	original_title	genres	release_d
2090	42194	0.59643	967000	TEKKEN	Crime|Drama|Action|Thriller|Science Fiction	3/20

In [13]:
```
# Drop duplicate movie
ten_df.drop_duplicates(inplace=True)
ten_df.duplicated('id').sum()
```

Out[13]: 0

Missing/Null Data

The info of the 10-year dataframe shows that there are missing genre field values. These will need to be resolved.

In [14]:
```
# Check for rows with missing genre
ten_df[ten_df['genres'].isnull()]
```

Out[14]:

	id	popularity	revenue	original_title	genres	release_date	release_year
424	363869	0.244648	0	Belli di papÃ	NaN	10/29/15	2015
620	361043	0.129696	0	All Hallows' Eve 2	NaN	10/6/15	2015
997	287663	0.330431	0	Star Wars Rebels: Spark of Rebellion	NaN	10/3/14	2014
1712	21634	0.302095	0	Prayers for Bobby	NaN	2/27/09	2009
1897	40534	0.020701	0	Jonas Brothers: The Concert Experience	NaN	2/27/09	2009
2370	127717	0.081892	0	Freshman Father	NaN	6/5/10	2010
2376	315620	0.068411	0	Doctor Who: A Christmas Carol	NaN	12/25/10	2010
3279	54330	0.145331	0	ì•„ê¸°ì™€ ë‚˜	NaN	8/13/08	2008
4547	123024	0.520520	0	London 2012 Olympic Opening Ceremony: Isles of...	NaN	7/27/12	2012
4732	139463	0.235911	0	The Scapegoat	NaN	9/9/12	2012
4797	369145	0.167501	0	Doctor Who: The Snowmen	NaN	12/25/12	2012
4890	126909	0.083202	0	Cousin Ben Troop Screening	NaN	1/1/12	2012
5830	282848	0.248944	0	Doctor Who: The Time of the Doctor	NaN	12/25/13	2013
5934	200204	0.067433	0	Prada: Candy	NaN	3/25/13	2013
6043	190940	0.039080	0	Bombay Talkies	NaN	5/3/13	2013

> Since these are objects and a relatively low number are missing, these rows can be dropped.

In [15]:

```
# Drop rows with missing genres
ten_df.dropna(inplace=True)
ten_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5465 entries, 0 to 7824
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              5465 non-null   int64  
 1   popularity      5465 non-null   float64
 2   revenue         5465 non-null   int64  
 3   original_title  5465 non-null   object 
 4   genres          5465 non-null   object 
 5   release_date    5465 non-null   object 
 6   release_year    5465 non-null   int64  
dtypes: float64(1), int64(3), object(3)
memory usage: 341.6+ KB
```

> The revenue data does appear to have null values, but will need to be reviewed for values of 0. If found, it will be replaced with the average revenue.

In [16]:
```
# Investigate revenue data
ten_df.query('revenue == 0')
```

Out[16]:

	id	popularity	revenue	original_title	genres	release_date	rel
48	265208	2.932340	0	Wild Card	Thriller|Crime|Drama	1/14/15	
67	334074	2.331636	0	Survivor	Crime|Thriller|Action	5/21/15	
74	347096	2.165433	0	Mythica: The Darkspore	Action|Adventure|Fantasy	6/24/15	
75	308369	2.141506	0	Me and Earl and the Dying Girl	Comedy|Drama	6/12/15	
92	370687	1.876037	0	Mythica: The Necromancer	Fantasy|Action|Adventure	12/19/15	
...	...	...	...	...	...	...	
7818	46169	0.019669	0	Twitches Too	Drama|Family|Fantasy|TV Movie	10/12/07	
7820	21623	0.017396	0	Beneath	Horror|Mystery|Thriller	7/8/07	
7821	39561	0.013017	0	Testosteron	Comedy	3/2/07	
7822	36443	0.010471	0	The Union: The Business Behind Getting High	Comedy|Documentary	5/8/07	
7823	19934	0.009512	0	Ce soir je dors chez toi	Comedy	11/21/07	

3296 rows × 7 columns

In [17]:

```
# Set revenue values of 0 to the mean revenue
rev_mean = ten_df['revenue'].mean().astype(int)
ten_df['revenue'] = ten_df['revenue'].replace(0, rev_mean)
```

In [18]:

```
# Validate change
ten_df.query('revenue == 0').count()
```

Out[18]:

```
id                0
popularity        0
revenue           0
original_title    0
genres            0
release_date      0
release_year      0
dtype: int64
```

> The release data is shown as having a datatype of object (string). It will need to be corrected by converting it to datetime instead.

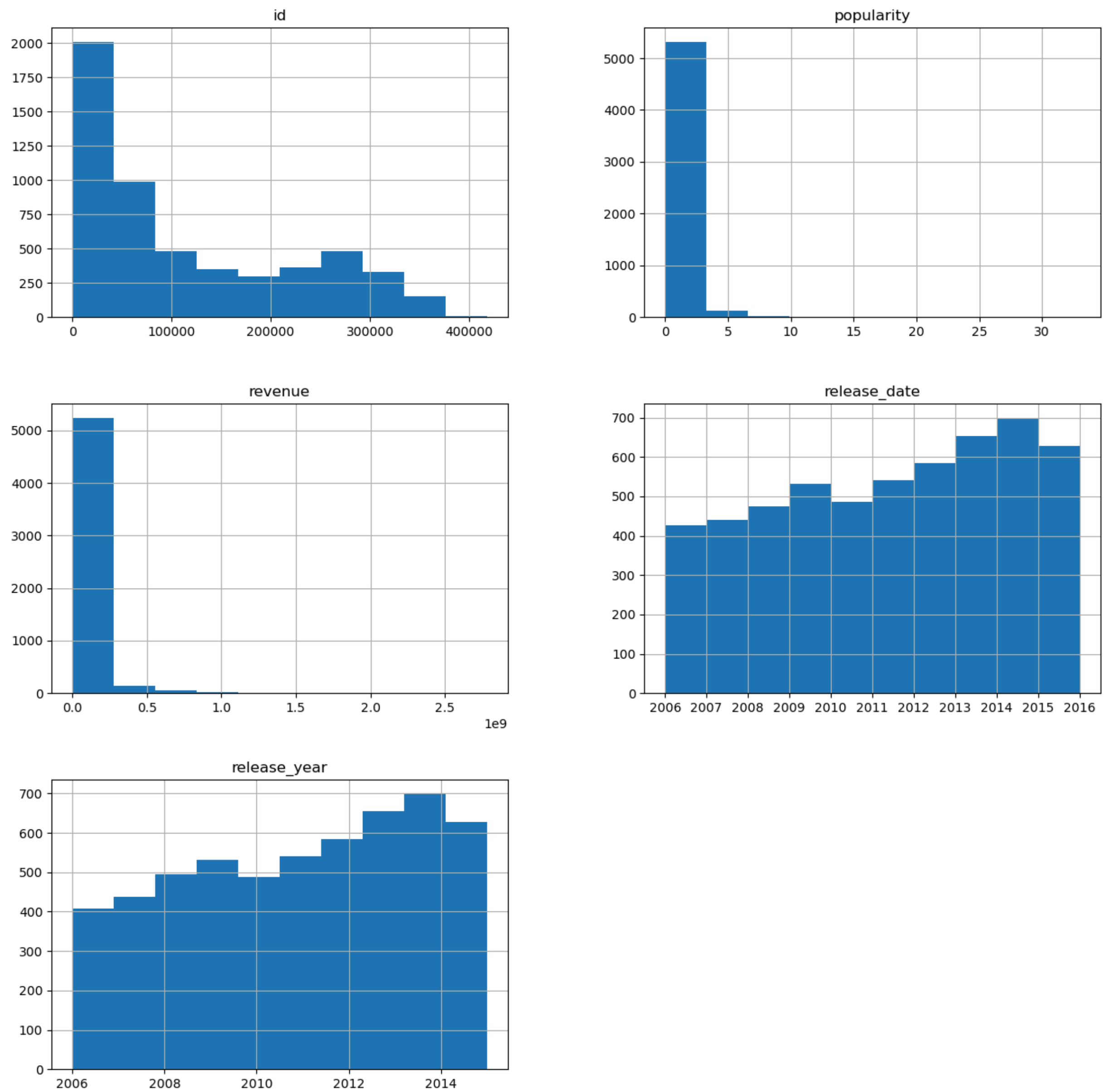
In [19]:
```python
# Convert release_date from string to datetime datatype
ten_df['release_date'] = pd.to_datetime(ten_df['release_date'], format='%m/%d/%y')
ten_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5465 entries, 0 to 7824
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              5465 non-null   int64
 1   popularity      5465 non-null   float64
 2   revenue         5465 non-null   int64
 3   original_title  5465 non-null   object
 4   genres          5465 non-null   object
 5   release_date    5465 non-null   datetime64[ns]
 6   release_year    5465 non-null   int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(2)
memory usage: 341.6+ KB
```

Exploratory Data Analysis

> First, the columns of the dataset will be examined to extract any useful information that can be derived from the individual attributes of the dataset; separated by **qualitative** and **quantitative** data.

In [20]:
```python
#Plot histograms: Qualitative data
ten_df.hist(figsize = (15,15));
```

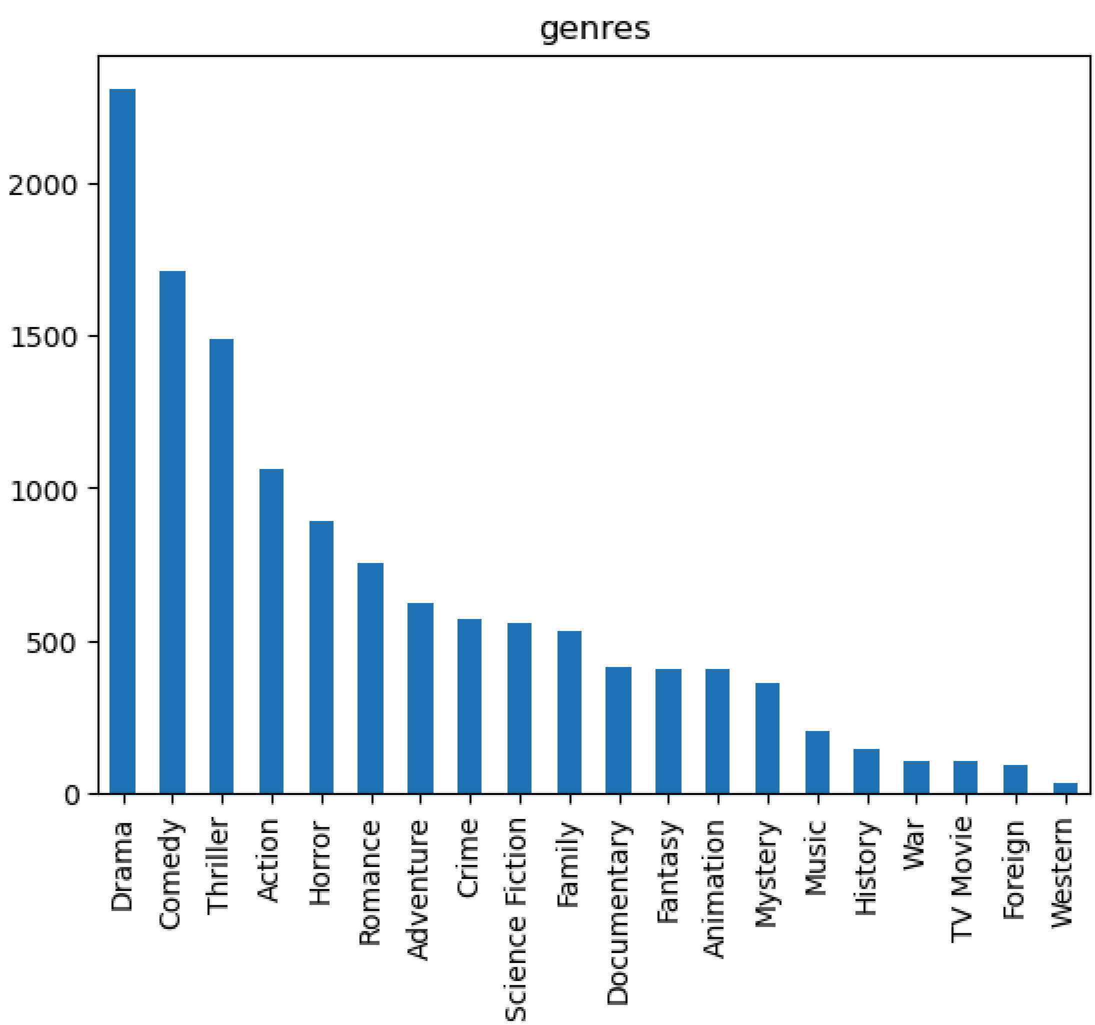


> One observation from the historic data is that the number of movies released increased over time. This information is shown by the gradual count increase on both the `release_date` and `release_year` charts.

In [21]:
```
# Separate the 'genres' column and regroup by individual genre
genre_list = ten_df['genres'].str.cat(sep='|')
genre_split = pd.Series(genre_list.split('|'))
genre_split.value_counts()
```

Out[21]:
```
Drama              2304
Comedy             1712
Thriller           1488
Action             1060
Horror              893
Romance             756
Adventure           624
Crime               571
Science Fiction     558
Family              530
Documentary         411
Fantasy             405
Animation           405
Mystery             363
Music               204
History             144
War                 108
TV Movie            103
Foreign              95
Western              36
Name: count, dtype: int64
```

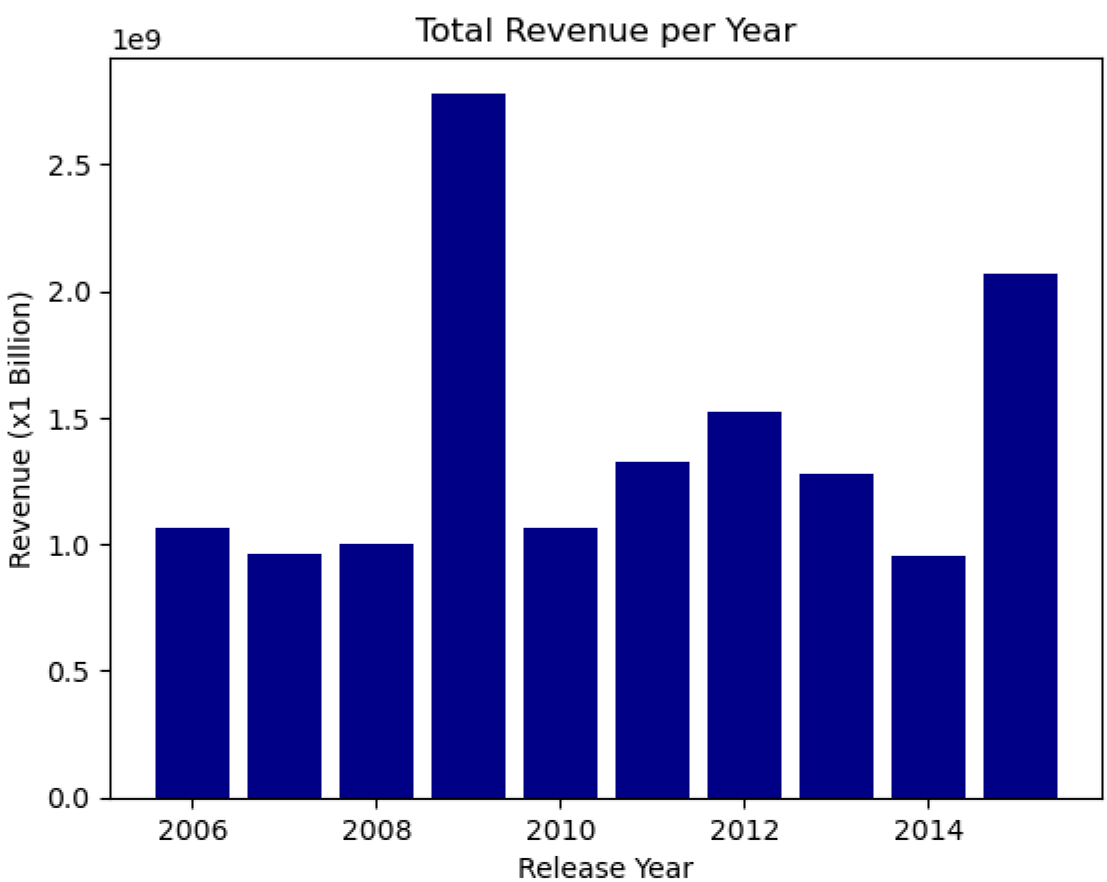
In [22]:
```python
# Plot bar chart: Quantitative data
genre_split.value_counts().plot(title = 'genres', kind='bar');
```



> The majority of movies were categorized as a drama, comedy, and/or thriller, respectively, during for this decade of data. Relatively, there were very few categorized as western, TV movies, or foreign.

> Now that the individual features have been explored, the relationship between the features can be inspected. This section will focus on how `revenue' relates to some of the other features.

In [23]:
```
# Plot bar chart: "Revenue by Release Year"
plt.bar(ten_df['release_year'], ten_df['revenue'], color = 'darkblue')
plt.xlabel('Release Year')
plt.ylabel('Revenue (x1 Billion)')
plt.title('Total Revenue per Year')
plt.show()
```



> The above chart shows 2009 and 2015 standing out with significantly higher revenues and 2012 appearing to be above the normal range too, while the remaining years have revenues that are realtively consistent. The observations suggests that these 2009 and 2015 years experienced financial success possibly due to well received film(s) in some capacity.

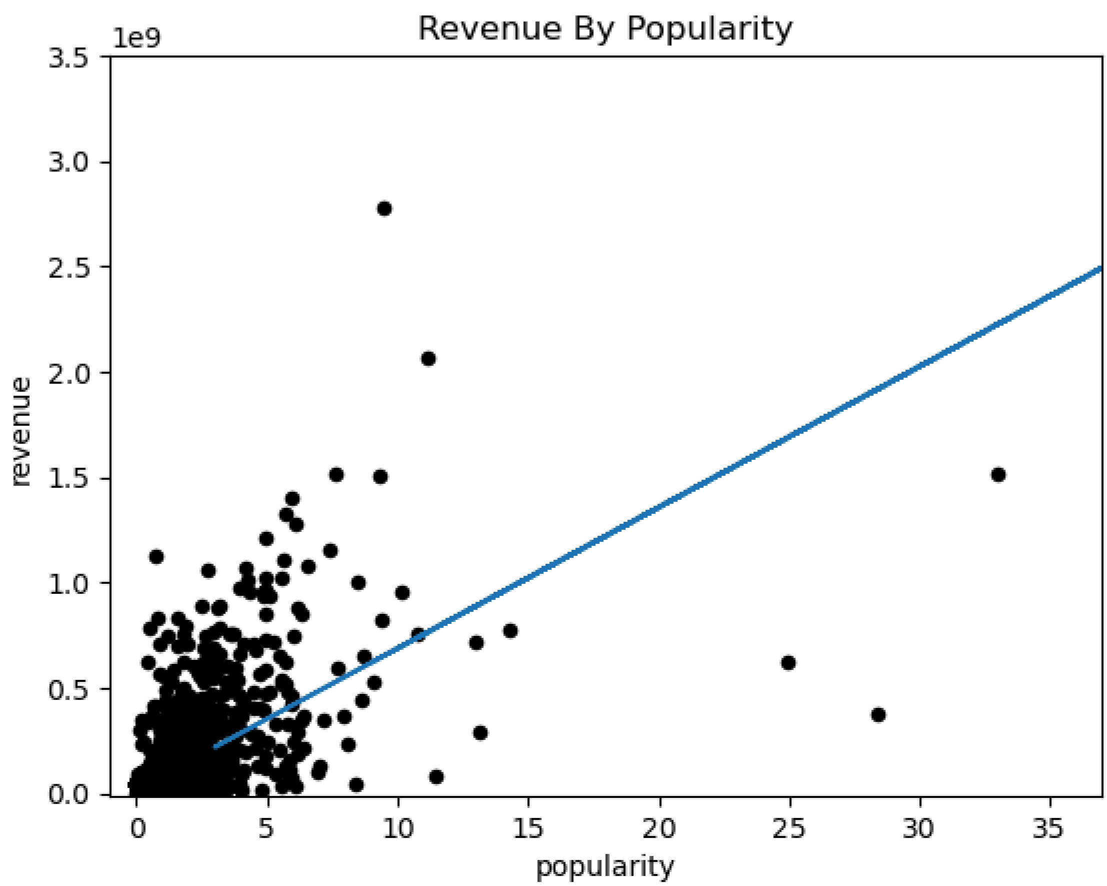
> To see if there is any correlation, the revenue in comparison to popularity will be examined.

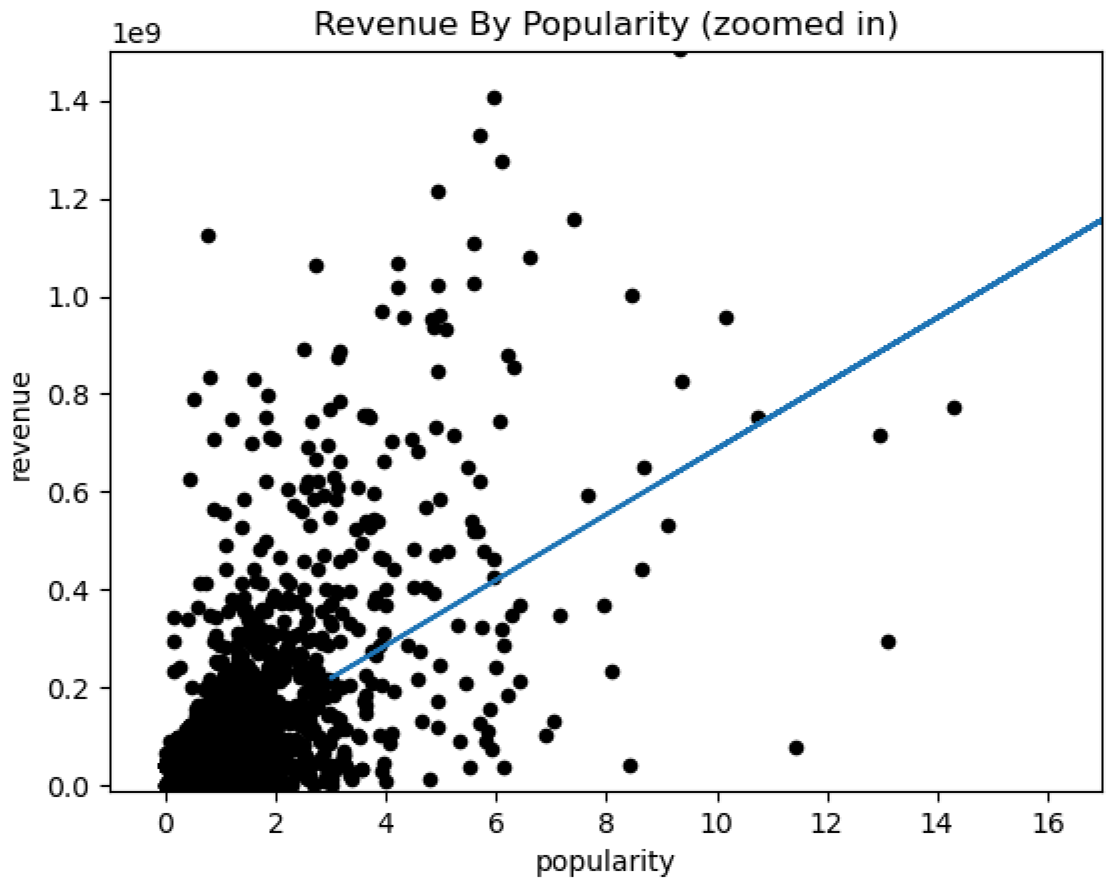
In [24]:
```
#Plot Scatter: "Revenue by Popularity"
import statsmodels.api as sm
import scipy
model = sm.OLS(ten_df.revenue, sm.add_constant(ten_df.popularity))
p = model.fit().params
x= ten_df.revenue
ax = ten_df.plot(x='popularity', y='revenue', title = 'Revenue By Popularity', kind
ax.plot(x, p.const + p.popularity* x)
ax.set_xlim([-1, 37])
ax.set_ylim([-10000000, 3500000000])
print("correlation : ", scipy.stats.pearsonr(ten_df.popularity, ten_df.revenue)),


model = sm.OLS(ten_df.revenue, sm.add_constant(ten_df.popularity))
p = model.fit().params
x= ten_df.revenue
ax = ten_df.plot(x='popularity', y='revenue', title = 'Revenue By Popularity (zoome
ax.plot(x, p.const + p.popularity* x)
ax.set_xlim([-1, 17])
ax.set_ylim([-10000000, 1500000000])
```

```
correlation :  PearsonRResult(statistic=0.6344262650111495, pvalue=0.0)
```

Out[24]:
```
(-10000000.0, 1500000000.0)
```





> The correlation is positive between revenue and popularity. There are a few outliers shown in the first scatter plot for both features. The use of queries will help determine if there is any relation to the previous bar chart.

In [25]:
```
# View population outliers
ten_df.query('popularity > 20')
```

Out[25]:

	id	popularity	revenue	original_title	genres	release_date
0	135397	32.985763	1513528810	Jurassic World	Action|Adventure|Science Fiction|Thriller	2015-06-09
1	76341	28.419936	378436354	Mad Max: Fury Road	Action|Adventure|Science Fiction|Thriller	2015-05-13
629	157336	24.949134	621752480	Interstellar	Adventure|Drama|Science Fiction	2014-11-05

In [26]:
```
# View revenue outliers
ten_df.query('revenue > 1.5e9')
```

Out[26]:

	id	popularity	revenue	original_title	genres	release
0	135397	32.985763	1513528810	Jurassic World	Action|Adventure|Science Fiction|Thriller	2015-
3	140607	11.173104	2068178225	Star Wars: The Force Awakens	Action|Adventure|Science Fiction|Fantasy	2015-
4	168259	9.335014	1506249360	Furious 7	Action|Crime|Thriller	2015-
1386	19995	9.432768	2781505847	Avatar	Action|Adventure|Fantasy|Science Fiction	2009-
4361	24428	7.637767	1519557910	The Avengers	Science Fiction|Action|Adventure	2012-

> As shown, some of the extremely popular movies were released in 2015. This also is observed with the movies that have extremely high revenues. However, it is still not definitive if the revenue for each year is effected by the highly popular movies.

> An additional step can be taken to see how all three, release year, title, and revenue, relate.

In [27]:

```
# View movies with the highest revenue by year
max_rev_ind = ten_df.groupby('release_year')['revenue'].idxmax()
top_rev_movies = ten_df.loc[max_rev_ind][['release_year', 'original_title', 'revenu
top_rev_movies = top_rev_movies.sort_values(by='release_year', ascending = False)
top_rev_movies
```

Out[27]:

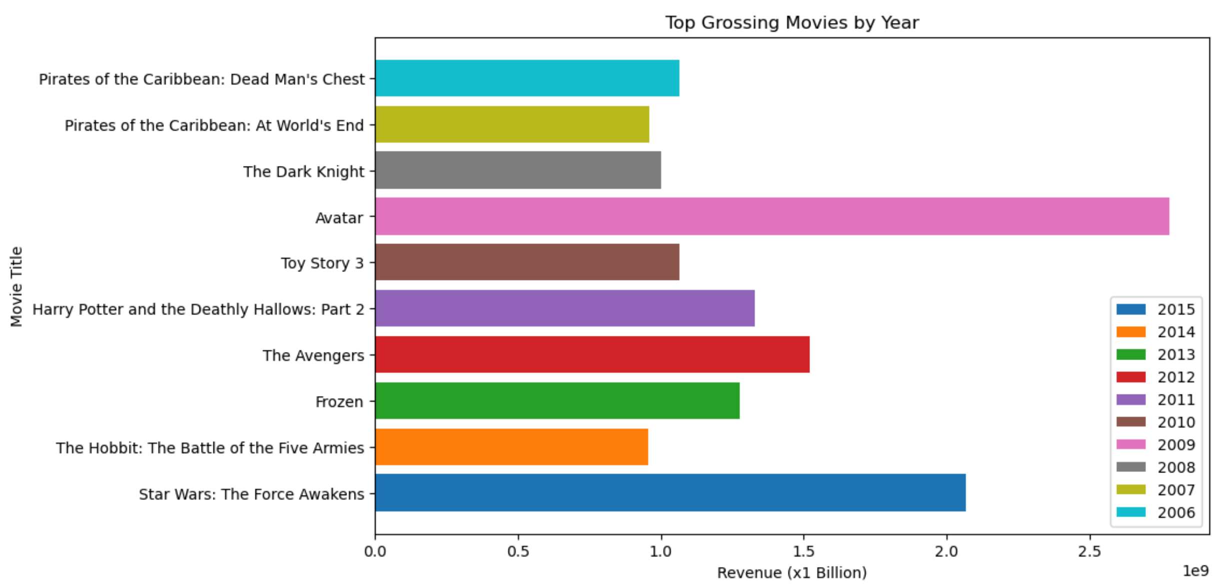
	release_year	original_title	revenue	release_date
3	2015	Star Wars: The Force Awakens	2068178225	2015-12-15
634	2014	The Hobbit: The Battle of the Five Armies	955119788	2014-12-10
5422	2013	Frozen	1274219009	2013-11-27
4361	2012	The Avengers	1519557910	2012-04-25
3374	2011	Harry Potter and the Deathly Hallows: Part 2	1327817822	2011-07-07
1930	2010	Toy Story 3	1063171911	2010-06-16
1386	2009	Avatar	2781505847	2009-12-10
2875	2008	The Dark Knight	1001921825	2008-07-16
7387	2007	Pirates of the Caribbean: At World's End	961000000	2007-05-19
6555	2006	Pirates of the Caribbean: Dead Man's Chest	1065659812	2006-06-20

In [28]:
```python
# Add `quarter` column
add_quarter(top_rev_movies)
top_rev_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, 3 to 6555
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   release_year    10 non-null     int64
 1   original_title  10 non-null     object
 2   revenue         10 non-null     int64
 3   release_date    10 non-null     datetime64[ns]
 4   quarter         10 non-null     category
dtypes: category(1), datetime64[ns](1), int64(2), object(1)
memory usage: 614.0+ bytes
```

In [29]:
```python
# Plot bar chart: "Top Grossing Movies by Year"
plt.figure(figsize=(10, 6))
years = top_rev_movies['release_year']
for year in years:
    year_data = top_rev_movies[top_rev_movies['release_year'] == year]
    plt.barh(year_data['original_title'], year_data['revenue'], label=year)

plt.ylabel('Movie Title')
plt.xlabel('Revenue (x1 Billion)')
plt.title('Top Grossing Movies by Year')
plt.legend()
plt.show()
```



> From this chart, we can see that the three highest grossing movies, "Star Wars: The Force Awakens", "The Avengers", and "Avatar" were released in 2015, 2012, and 2009; respectively. This coincides with the outlier revenue data in the scatter plot, and the top revenue by release year. From this, it can be

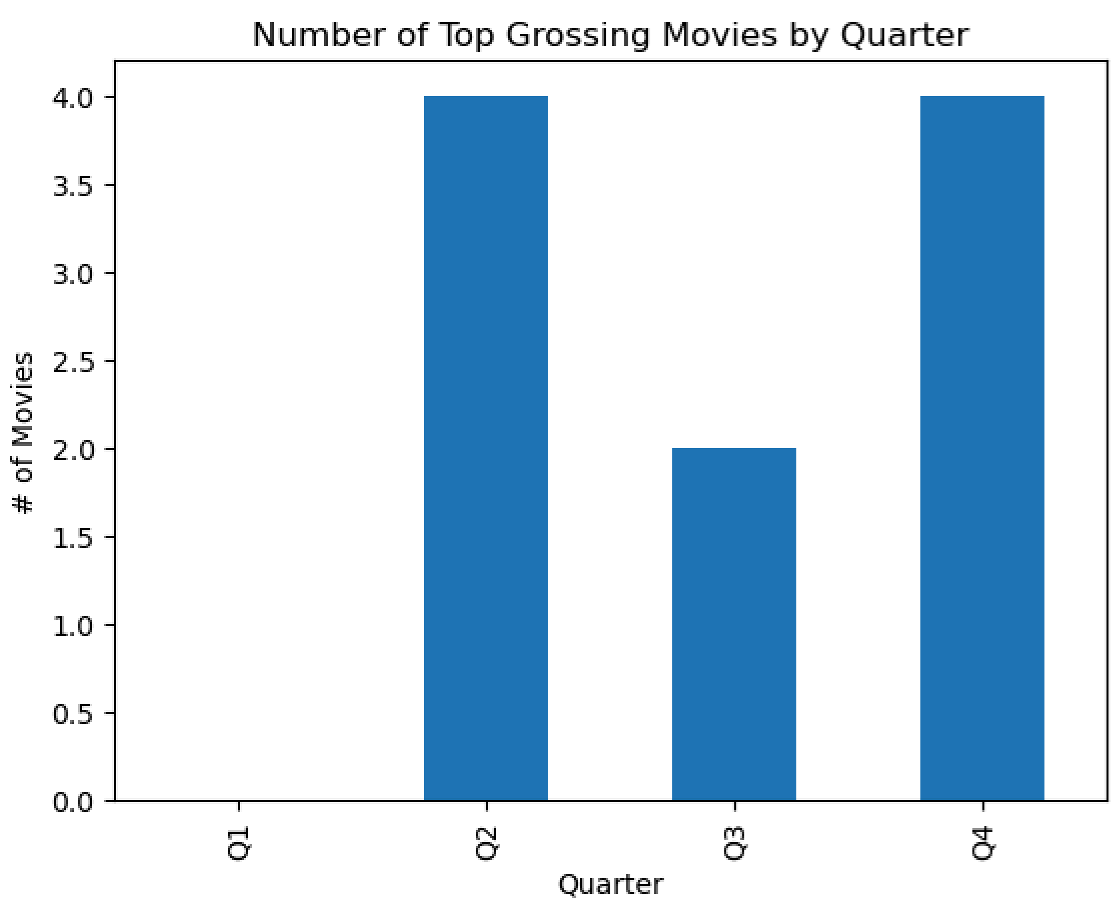
> inferred that those movies played a significant role in overall revenue for each of those years.

> Another step can be taken to show in which quarter these top grossing movies fall.

In [30]:
```
# Display the number of Top Grossing Movies by Quarter
top_qtr_cnt = top_rev_movies.groupby(['quarter'])['release_year'].count()
top_qtr_cnt
```

Out[30]:
```
quarter
Q1    0
Q2    4
Q3    2
Q4    4
Name: release_year, dtype: int64
```

In [31]:
```
top_qtr_cnt.plot(xlabel = 'Quarter', ylabel = '# of Movies', title ='Number of Top
```



> It is shown here that Q1 held none of the top producing movies over the 10-year period with Q2 and Q4 tying at 4 and Q3 having 2. From this an assumption may be made that due to a lack of a blockbuster, Q1 does not bring in as much revenue on average as the other three quarters. Next, an

> analysis will be done to find out if one were to make that assumption, would it hold true.

Research Question 1: Which quarter of the year typically brings in the least revenue?

> To answer this question, the release months will need to be grouped into yearly quarters. Once completed, average revenue can then be calculated and charted.

In [32]:
```
# Add quarter column to the 10-year dataframe
add_quarter(ten_df)
ten_df.head()
```

Out[32]:

	id	popularity	revenue	original_title	genres	release_date	rele
0	135397	32.985763	1513528810	Jurassic World	Action|Adventure|Science Fiction|Thriller	2015-06-09	
1	76341	28.419936	378436354	Mad Max: Fury Road	Action|Adventure|Science Fiction|Thriller	2015-05-13	
2	262500	13.112507	295238201	Insurgent	Adventure|Science Fiction|Thriller	2015-03-18	
3	140607	11.173104	2068178225	Star Wars: The Force Awakens	Action|Adventure|Science Fiction|Fantasy	2015-12-15	
4	168259	9.335014	1506249360	Furious 7	Action|Crime|Thriller	2015-04-01	

> With the new column created, the rows can now be cluster based on the quarter in which it was released to find how much revenue was generated on average during those periods.

In [33]:
```
# Get the average quarterly revenue
qtr_rev_means(ten_df)
```

Out[33]:
```
quarter
Q1    5.516271e+07
Q2    8.677324e+07
Q3    5.332449e+07
Q4    7.132752e+07
Name: revenue, dtype: float64
```

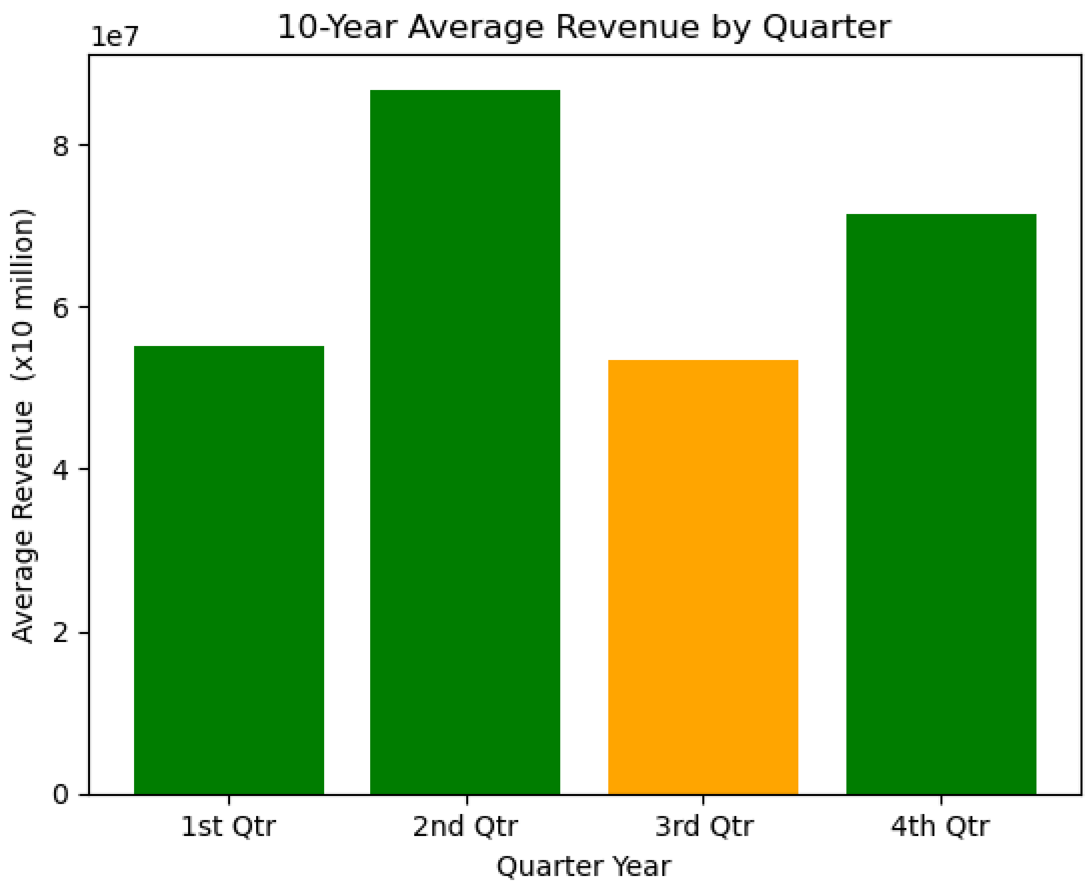
In [34]:
```
# Plot bar chart: "10-Year Average Revenue by Quarter"
locations = [1, 2, 3, 4]
heights = qtr_rev_means(ten_df)
```

```
color = ['green', 'green', 'orange', 'green']

labels = ['1st Qtr', '2nd Qtr', '3rd Qtr', '4th Qtr']

plt.bar(locations, heights, tick_label=labels, color=color)
plt.title('10-Year Average Revenue by Quarter')
plt.xlabel('Quarter Year')
plt.ylabel('Average Revenue  (x10 million)');


plt.show()
```



Answer 1:

On average, the 3rd quarter produced the least revenue during the 10-year period. Films during this time generated an average of ~ $53M which is just shy of the 1st quarter at ~ $55M

Research Question 2: Do more movie releases within a quarter equate to greater revenue?

> In order to answer this question, the total number of movies released for each quarter will need to be gathered and compared with the revenue.

In [35]:
```python
# Get total movies released per quarter
releases_count = ten_df.groupby(['quarter'])['original_title'].count()
releases_count
```

Out[35]:
```
quarter
Q1    1278
Q2    1189
Q3    1630
Q4    1368
Name: original_title, dtype: int64
```

In [36]:
```python
# Get total revenue per quarter
total_rev = ten_df.groupby(['quarter'])['revenue'].sum()
total_rev
```

Out[36]:
```
quarter
Q1     70497942946
Q2    103173383811
Q3     86918916163
Q4     97576042417
Name: revenue, dtype: int64
```

> Next, the number of releases and revenue for each quarter will be combined to better manage the data

In [37]:
```python
# Append the count and revenue data
bin_names = sorted(ten_df['quarter'].unique())
combo = np.column_stack((bin_names,releases_count, total_rev))
```

In [38]:
```python
# Create new dataframe from the combined arrays
rr_df = pd.DataFrame(combo, columns=['quarter', 'releases', 'total_revenue'])

rr_df.head()
```

Out[38]:

	quarter	releases	total_revenue
0	Q1	1278	70497942946
1	Q2	1189	103173383811
2	Q3	1630	86918916163
3	Q4	1368	97576042417

In [39]:
```python
rr_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   quarter        4 non-null      object
 1   releases       4 non-null      object
 2   total_revenue  4 non-null      object
dtypes: object(3)
memory usage: 228.0+ bytes
```

> Since two arrays were joined, the data was read as objects (strings), so the datatypes for the `releases` and `total_revenue` columns need to be corrected.

In [40]:
```
# Correct datatypes
dtype_convert(rr_df, 'total_revenue', 'float')
dtype_convert(rr_df, 'releases', 'int')
```

Out[40]:
```
quarter           object
releases           int64
total_revenue    float64
dtype: object
```

> A pie chart will be used to compare each quarter of the total number of releases within the 10-year span.

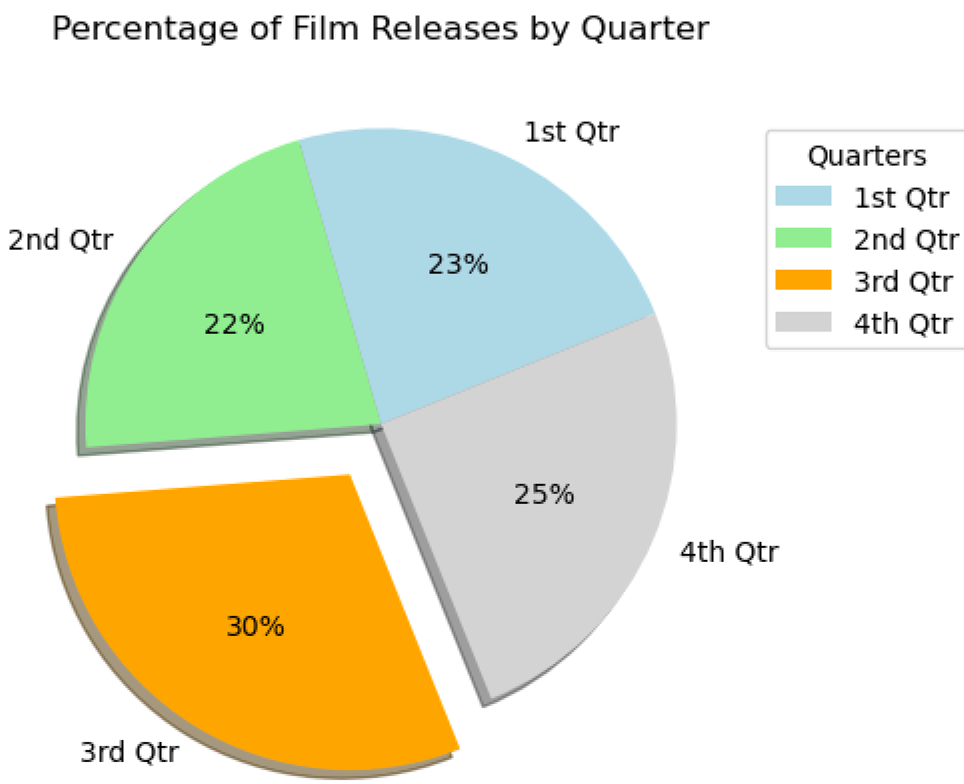
In [41]:
```
# Plot pie chart: "Percentage of Film Releases by Quarter"
sizes = rr_df['releases']
explode = (0, 0, 0.2, 0)
c_color = ['lightblue', 'lightgreen', 'orange', 'lightgray']

fig, ax = plt.subplots()
ax.pie(sizes, explode=explode, autopct='%1.0f%%',labels =labels, colors=c_color,
       shadow=True, startangle=22)

plt.legend(labels,
          title="Quarters",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1.5))

plt.title('Percentage of Film Releases by Quarter')

plt.show()
```



In [42]:
```
# Display release in descending order
rel_sort = rr_df.sort_values(by=['releases'], ignore_index=True, ascending=False)
rel_sort.drop(['total_revenue'],axis=1)
```

Out[42]:

	quarter	releases
0	Q3	1630
1	Q4	1368
2	Q1	1278
3	Q2	1189

> The data shows that in relation to the other quarters, the 3rd Quarter had the most movie releases at over 1600. That is 8% more that the quarter with the least releases, the 2nd quarter, with a little shy of 1300 releases.

> Now to explore the total revenue. Based upon the same time data, the gross revenue will be analyzed.

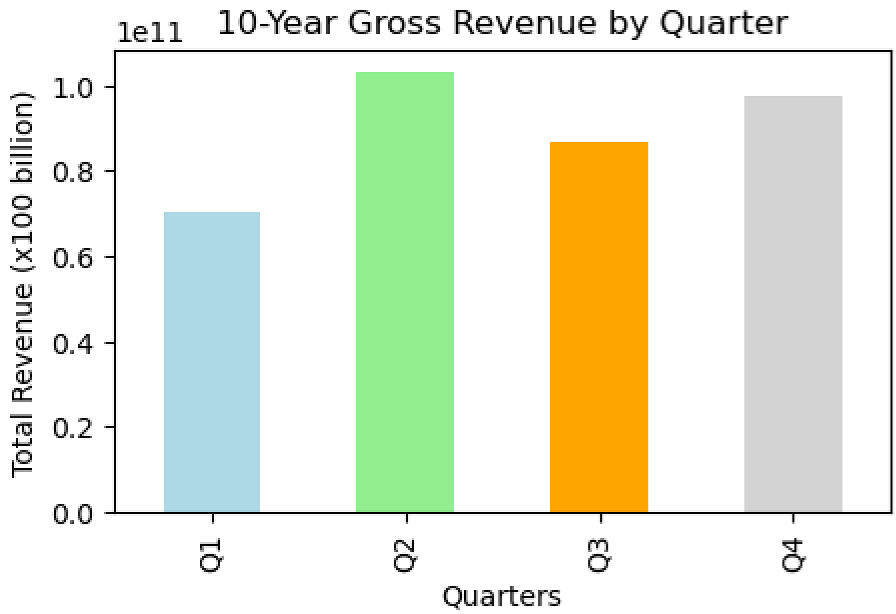
In [43]:
```
# Plot bar chart: "10-Year Gross Revenue by Quarter"

rr_df.plot(x='quarter', y='total_revenue', kind='bar',
           legend=False, figsize=(5,3), color=c_color)

plt.title('10-Year Gross Revenue by Quarter')
plt.xlabel('Quarters')
```

```python
plt.ylabel('Total Revenue (x100 billion)')

plt.show()
```



In [44]:
```python
# Display the the total revenue in descending order
rev_sort = rr_df.sort_values(by=['total_revenue'], ignore_index=True, ascending=Fal
rev_sort.drop(['releases'],axis=1)
```

Out[44]:

	quarter	total_revenue
0	Q2	1.031734e+11
1	Q4	9.757604e+10
2	Q3	8.691892e+10
3	Q1	7.049794e+10

> The data shows that, overall, movies released in the 2nd quarter ranks as number one and generated the most revenue totaling over $100B. That is a stark contrast from the previous graph which showed that the least amount of films were released during that time.

Answer 2

The data indicates that the overall quantity of movie releases do not determine revenue outcome within a quarter. The 2nd quarters had the least releases, but produces the greatest revenue. The 3rd quarters released the most movies, but only beat out one other quarter in terms of revenue.

Conclusions

> Analyzing the data by quarter indicates that on average, within a 10-year period, revenue is not as high during the 3rd quarter of the year. The average revenue is shown as $53M; $33M less than the topped ranked 2nd quarter which averages around $86M.

> The data also suggests that the volume of releases does not necessarily equate to higher revenue. Although the 3rd quarter had the most movies released (roughly 1600), it failed to generate the most revenue. The data actually showed that the inverse occurred. The quarter with the lowest releases, the 2nd quarter, totaled over $100B; the most of any quarter.

> What can be drawn from this is that the quarter in which a movie is released is not the only factor that determines the outcome of greater revenue. This can also be said about the number of releases within a quarter as most of the quarters ranked differently in each analysis. There could be many other factors that could effecting a movie's success such as marketing, popularity, content interest, etc. that were not available or not explored in this report.

> Additionally, a factor that could have directly skewed this data is the missing revenue data that was replaced with the mean during the cleaning phase. There were 296 rows that had that missing data. It had not be grouped into quarters to determine which period had the bulk of missing data, but it is possible that the a large number of releases were in the 2nd quarter and supplied with the mean revenue. In turn, this could have inflated the revenue thus affecting how the 2nd quarter revenue performed in comparison to the other quarters.

Cite

J.D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp.

McKinney, W., & others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56).

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

Thanks for viewing

S.J. Richardson

Github: SQLJamz