Project: Investigating the TMDb Movie Dataset

Table of Contents

Introduction

> This dataset contains information about 10,000 movies collected from The Movie Database (TMDb), including user ratings and revenue.

> The Dataset contains:
>
> - 10866 observations/rows
> - 26 features/columns
> - 9 columns with null values
> - 8874 rows that have null values in one or more columns
> - Columns like budget, revenue, budget_adj, revenue_adj contain lots of 0s

In [1]:
```
# Setting up import statements and Loading the dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("darkgrid")
colors = ['Grey', 'Purple', 'Blue', 'Brown', 'Green', 'Orange', 'Red', 'Lime', '(


#Loading dataset into a dataframe using pandas

imdb_df = pd.read_csv("tmdb-movies.csv")
```

Questions

> Questions we would like to answer with this dataset include:
>
> 1. Which genres are most popular from year to year?
> 2. The number of Movies Released Each Year.
> 3. Which is the most popular movie and the least popular movie and what features are associated with popular and less popular movies?
> 4. Has the runtime of movies been declining over the years?
> 5. Is the Movie industry making or loosing money and what is the relationship between budget and popularity?

Data Wrangling

> In this section we will look at the features of the dataset and see if their is a need to clean it.

General Properties of the Tmdb Movie Dataset

In [2]:
```
# Looking at the first few rows of the dataset

imdb_df.head()
```

Out[2]:

	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...	http://www.tl
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D...	http:/.
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle ...	

5 rows × 21 columns

In [3]:
```
# Lets look at the shape of the dataset

imdb_df.shape
```

Out[3]: (10866, 21)

The dataset contains **10866** rows and **21** columns.

In [4]:
```
# Using .info() method to look at dataset information

imdb_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
```

Looks like some columns in the dataset contain null values

In [5]:
```
#looking at the last few rows of the dataset

imdb_df.tail()
```

Out[5]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage
10861	21	tt0060371	0.080598	0	0	The Endless Summer	Michael Hynson|Robert August|Lord 'Tally Ho' B...	NaN
10862	20379	tt0060472	0.065543	0	0	Grand Prix	James Garner|Eva Marie Saint|Yves Montand|Tosh...	NaN
10863	39768	tt0060161	0.065141	0	0	Beregis Avtomobilya	Innokentiy Smoktunovskiy|Oleg Efremov|Georgi Z...	NaN
10864	21449	tt0061177	0.064317	0	0	What's Up, Tiger Lily?	Tatsuya Mihashi|Akiko Wakabayashi|Mie Hama|Joh...	NaN
10865	22293	tt0060666	0.035919	19000	0	Manos: The Hands of Fate	Harold P. Warren|Tom Neyman|John Reynolds|Dian...	NaN

5 rows × 21 columns

The dataset also has columns with zero values

In [6]:
```
#Number of nulls in each column

imdb_df.isna().sum()
```

Out[6]:
```
id                         0
imdb_id                   10
popularity                 0
budget                     0
revenue                    0
original_title             0
cast                      76
homepage                7930
director                  44
tagline                 2824
keywords                1493
overview                   4
runtime                    0
genres                    23
production_companies    1030
release_date               0
vote_count                 0
vote_average               0
release_year               0
budget_adj                 0
revenue_adj                0
dtype: int64
```

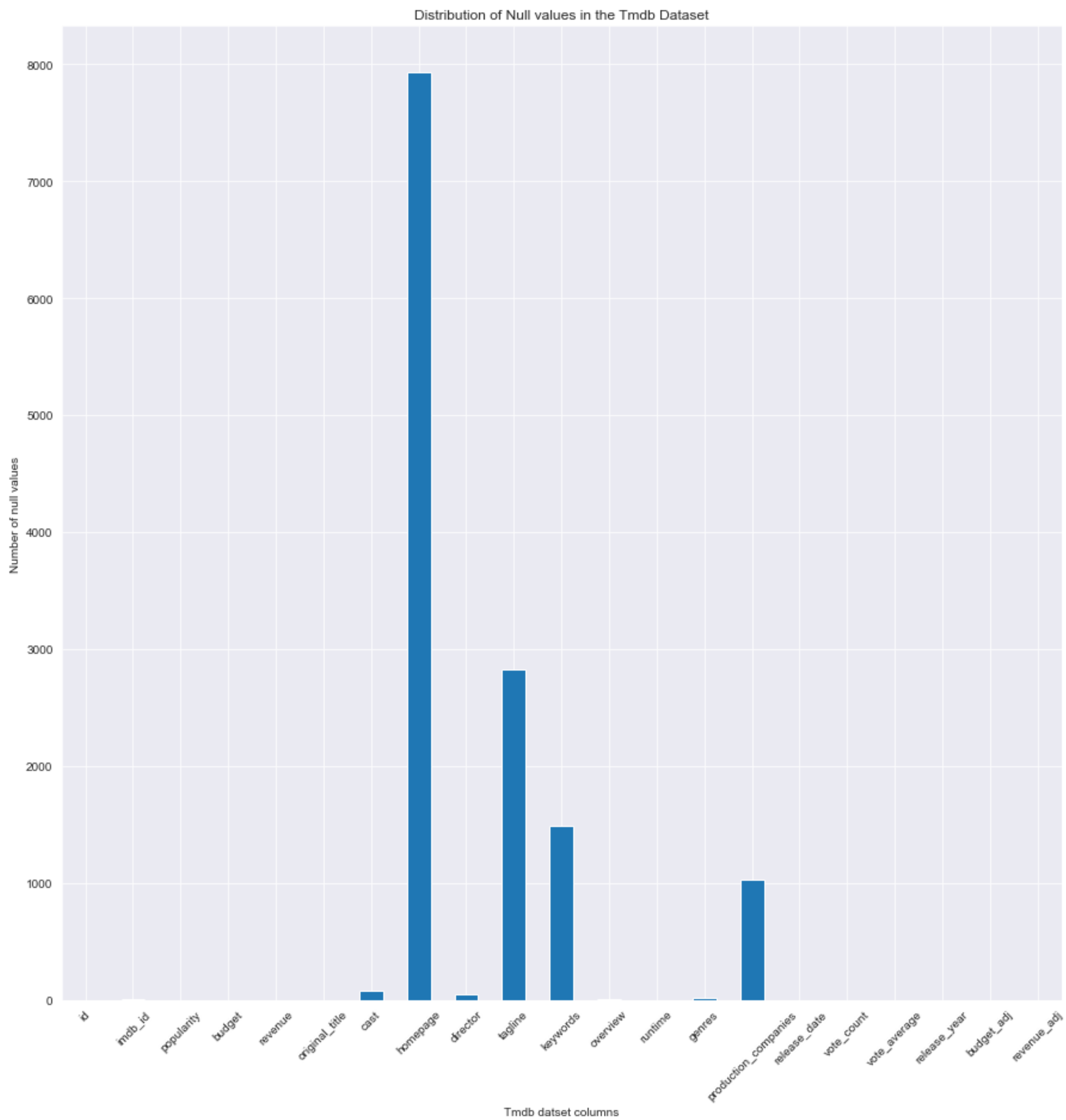
In [7]:
```
# total number of columns with null values

imdb_df.isna().any().sum()
```

Out[7]:
```
9
```

In [8]:
```
# list of columns with null values

list(imdb_df.columns[imdb_df.isna().any()])
```

Out[8]:
```
['imdb_id',
 'cast',
 'homepage',
 'director',
 'tagline',
 'keywords',
 'overview',
 'genres',
 'production_companies']
```

In [9]:
```
#Visualizing the distribution of null values in each column

imdb_df.isna().sum().plot(kind='bar', figsize=(15,15))
plt.title("Distribution of Null values in the Tmdb Dataset")
plt.xticks(rotation=45)
plt.xlabel("Tmdb datset columns")
plt.ylabel("Number of null values")
plt.show()
```



> The homepage column has the highest number of null values followed by tagline and keywords

In [10]:
```
#Number of rows with missing data

imdb_df.isna().any(axis=1).sum()
```

Out[10]:
```
8874
```

Conclusion

Dataset Information

> Using the .info() method, we see that there are columns with null values in the dataset.

> Another observation is the release_date column has a type of "String". It should be changed to datetime object.

> Further inspection tells us that there are 9 columns with null values and the most affected is the homepage column with 7930 nulls out of 10866 observations. The distribution of nulls was visualized with a bar chart as shown above.

> we also see that there 8874 rows with null values in one or more columns

> We will need to determine which columns to keep for our analysis and whether to discard those with nulls or fill them with aggregate data.

Data Cleaning

Data Cleaning Steps to be carried out

Columns to Drop: We will be dropping columns we have identified as "not important to our analysis or questions". They include:

- imdb_id
- cast
- homepage
- tagline
- keywords
- overview
- production_companies
- revenue_adj
- budget_adj

Dealing With nulls: Since we identified that there are 8874 rows in the Tmdb dataset that contain null values as a result of the homepage column having over 7000 null values. This will be dealt with after dropping the homepage column. After which, we will have to drop null values from the columns we are keeping.

Columns with zeros (0s): we will replace zeros with nans (null values) and remove them.

Changing data types: The release_date column type will be changed to datetime.

Duplicates: duplicates will be identified and dropped.

Dropping Unwanted Columns

In [11]:
```
# Dropping unwanted columns

imdb_df.drop(['imdb_id', 'cast', 'homepage', 'tagline', 'keywords', 'overview', '
```

In [12]:
```
# Verifying that specified columns have been dropped
imdb_df.head(2)
```

Out[12]:

	id	popularity	budget	revenue	original_title	director	runtime	ge
0	135397	32.985763	150000000	1513528810	Jurassic World	Colin Trevorrow	124	Action|Adventure|Sci Fiction|Th
1	76341	28.419936	150000000	378436354	Mad Max: Fury Road	George Miller	120	Action|Adventure|Sci Fiction|Th

In [13]:
```
# The new shape of dataset after dropping columns

imdb_df.shape
```

Out[13]:
```
(10866, 12)
```

Dealing with nulls and 0s

In [14]:
```
# Replacing 0s with nan using Numpy's np.nan function

imdb_df.replace(0, np.nan, inplace=True)
```

In [15]:
```
# Number of null values after replacing 0s with nan in the remaining columns

imdb_df.isna().sum()
```

Out[15]:
```
id                   0
popularity           0
budget            5696
revenue           6016
original_title       0
director            44
runtime             31
genres              23
release_date         0
vote_count           0
vote_average         0
release_year         0
dtype: int64
```

In [16]:
```
# Dropping Nulls

imdb_df = imdb_df.dropna()
```

In [17]:
```
# New shape of dataset after dropping nulls

imdb_df.shape
```

Out[17]:
```
(3854, 12)
```

In [18]:
```
# verifying that all the null values have been removed

imdb_df.isna().sum()
```

Out[18]:
```
id                0
popularity        0
budget            0
revenue           0
original_title    0
director          0
runtime           0
genres            0
release_date      0
vote_count        0
vote_average      0
release_year      0
dtype: int64
```

In [19]:
```
# verifying that all the null values have been removed

imdb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3854 entries, 0 to 10848
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              3854 non-null   int64  
 1   popularity      3854 non-null   float64
 2   budget          3854 non-null   float64
 3   revenue         3854 non-null   float64
 4   original_title  3854 non-null   object 
 5   director        3854 non-null   object 
 6   runtime         3854 non-null   float64
 7   genres          3854 non-null   object 
 8   release_date    3854 non-null   object 
 9   vote_count      3854 non-null   int64  
 10  vote_average    3854 non-null   float64
 11  release_year    3854 non-null   int64  
dtypes: float64(5), int64(3), object(4)
memory usage: 391.4+ KB
```

Changing release_date Data Type

In [20]:
```
# converting the release_date column type from string to datetime using pandas to

imdb_df['release_date'] = pd.to_datetime(imdb_df['release_date'])
```

In [21]:
```
# checking to see if the conversion was successful

imdb_df.dtypes
```

Out[21]:
```
id                           int64
popularity                 float64
budget                     float64
revenue                    float64
original_title              object
director                    object
runtime                    float64
genres                      object
release_date        datetime64[ns]
vote_count                   int64
vote_average               float64
release_year                 int64
dtype: object
```

Duplicates

In [22]:
```
#Number of duplicates in the dataset

imdb_df.duplicated().sum()
```

Out[22]: 1

In [23]:
```
# looking at the duplicated row

imdb_df[imdb_df.duplicated()]
```

Out[23]:

	id	popularity	budget	revenue	original_title	director	runtime	
0	42194	0.59643	30000000.0	967000.0	TEKKEN	Dwight H. Little	92.0	Crime|Drama|Action|Thriller|

In [24]:
```
# Taking a closer look at the duplicated rows

imdb_df[imdb_df['id'] == 42194]
```

Out[24]:

	id	popularity	budget	revenue	original_title	director	runtime	
2089	42194	0.59643	30000000.0	967000.0	TEKKEN	Dwight H. Little	92.0	Crime|Drama|Action|Tl
2090	42194	0.59643	30000000.0	967000.0	TEKKEN	Dwight H. Little	92.0	Crime|Drama|Action|Tl

In [25]:
```
# Drop duplicate
imdb_df.drop_duplicates(inplace=True)
```

In [26]:
```
# verifying that duplicate has been removed
imdb_df.duplicated().sum()
```

Out[26]: 0

In [27]:
```
# Final Shape: Number of rows and columns after cleanning the dataset

imdb_df.shape
```

Out[27]: (3853, 12)

Conclusion

After dropping unwanted columns, nulls, duplicates and changing the data type of the release_date column to datetime, the dataset now has a dimension of **3853** rows and **13** columns.

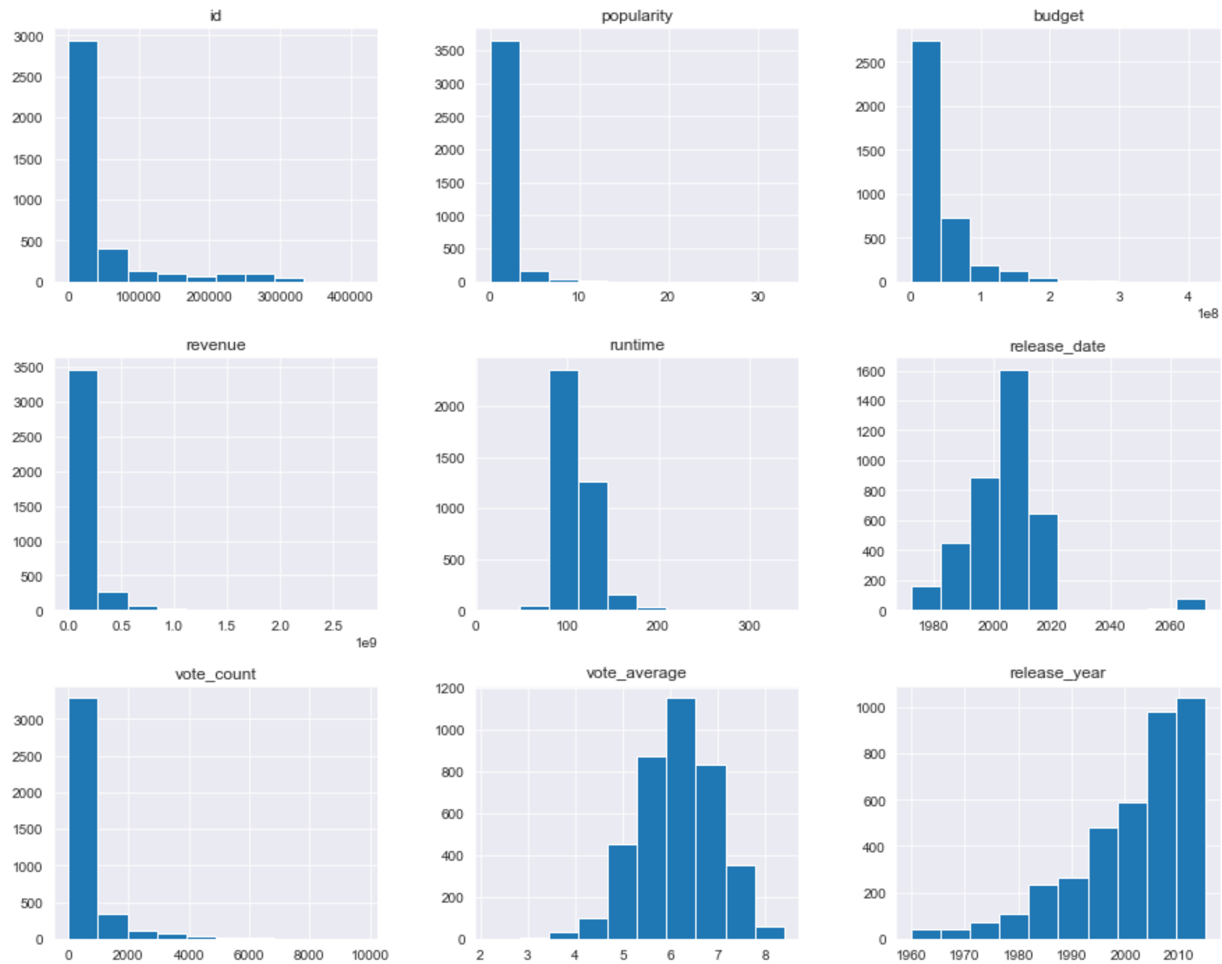
Exploratory Data Analysis

In [28]:
```
# Looking the descriptive information about the numerical columns of the Tmdb dat

imdb_df.describe()
```

Out[28]:

	id	popularity	budget	revenue	runtime	vote_count	vote_av
count	3853.000000	3853.000000	3.853000e+03	3.853000e+03	3853.000000	3853.000000	3853.0
mean	39894.523488	1.191825	3.721227e+07	1.077117e+08	109.208928	527.854399	6.1
std	67230.100737	1.475258	4.221035e+07	1.765554e+08	19.912913	880.031643	0.7
min	5.000000	0.001117	1.000000e+00	2.000000e+00	15.000000	10.000000	2.2
25%	6073.000000	0.462609	1.000000e+07	1.360940e+07	95.000000	71.000000	5.7
50%	11321.000000	0.797723	2.400000e+07	4.480678e+07	106.000000	204.000000	6.2
75%	38575.000000	1.368403	5.000000e+07	1.242721e+08	119.000000	580.000000	6.7
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	338.000000	9767.000000	8.4

In [29]:
```
# Plotting the histogram of all numerical columns in the Tmdb dataset

imdb_df.hist(figsize=(15,12));
```



Conclusion

From the .describe() method we see that the maximum popularity score is 32.99, maximum runtime of 900 minutes (unit is not specified, I am assuming).

We also see the min, max, mean values for the budget and revenue columns

These histograms show mostly right skewed distributions with the exception of vote_average and release year that are left skewed.

Research Questions

Question 1: Which genres are most popular from year to year?

In [30]:
```
# grouping the imdb_df dataframe by release_year and genres then counting the id
mov = imdb_df.groupby(['release_year', 'genres'])['id'].count()

mov
```

Out[30]:
```
release_year  genres
1960          Action|Adventure|Western                  1
              Action|Drama|History                      1
              Comedy|Drama|Romance                      1
              Comedy|Romance                            1
              Drama|Horror|Thriller                     1
                                                       ..
2015          Thriller|Drama|Adventure|Action|History   1
              Thriller|Horror                           3
              Thriller|Mystery                          1
              War|Adventure|Science Fiction             1
              Western|Drama|Adventure|Thriller          1
Name: id, Length: 2723, dtype: int64
```

In [31]:
```
# converting the mov series object to a dataframe object
mov_df = pd.DataFrame(mov)

#looking at the last 30 rows of the mov_df dataframe
mov_df.tail(30)
```

Out[31]:

		id
release_year	**genres**	
2015	**Fantasy|Action|Adventure**	1
	Fantasy|Comedy|Animation|Science Fiction|Family	1
	Fantasy|Drama|Romance	1
	Fantasy|Thriller	1
	History|Drama	2
	Horror	2
	Horror|Comedy|Fantasy	1
	Horror|Thriller	5
	Mystery|Crime|Action|Thriller|Drama	1
	Mystery|Drama	1
	Mystery|Horror	1
	Mystery|Thriller|Fantasy|Horror|Drama	1
	Romance|Comedy	1
	Romance|Comedy|Crime|Drama	1
	Romance|Drama	3
	Romance|Fantasy|Family|Drama	1
	Science Fiction|Action|Adventure	1
	Science Fiction|Action|Thriller|Adventure	1
	Science Fiction|Fantasy|Action|Adventure	1
	Science Fiction|Mystery|Thriller	1
	Science Fiction|Thriller	1
	Thriller	1
	Thriller|Action|Crime	1
	Thriller|Comedy|Drama|Romance|Science Fiction	1
	Thriller|Drama	1
	Thriller|Drama|Adventure|Action|History	1
	Thriller|Horror	3
	Thriller|Mystery	1
	War|Adventure|Science Fiction	1
	Western|Drama|Adventure|Thriller	1

In [32]:
```
# Visualizing the mov_df dataframe

#mov_df.plot(kind='bar', figsize=(15, 15));
```

visualizing the result of the groupby() and count() operation on the release_year and genres columns doesn' t paint a clear picture.

> Not sure if it is as a result of the nature of values in the genres column or we are using the wrong plot.

> So we are forced to ask another question. **Which genre is the most popular?**

Which genre is the most popular

In [33]:
```
# lets try and find out the most popular genre

imdb_df['genres'].value_counts()
```

Out[33]:
```
Drama                                    245
Comedy                                   233
Drama|Romance                            107
Comedy|Romance                           104
Comedy|Drama|Romance                      91
                                        ...
Drama|Thriller|Crime|Mystery|Romance       1
Comedy|Romance|Science Fiction             1
Horror|Thriller|Mystery|Fantasy            1
Fantasy|Animation|Comedy|Family            1
Action|Adventure|Drama|War|Romance         1
Name: genres, Length: 1053, dtype: int64
```

In [34]:
```
# Creating a plot to visualize the most popular genre

#imdb_df['genres'].value_counts().plot(kind='bar', figsize=(15,15));
```

> ### Still not a clear picture. The genres column needs some work.

In [35]:
```
#lets take a closer look
imdb_df['genres'].head()
```

Out[35]:
```
0    Action|Adventure|Science Fiction|Thriller
1    Action|Adventure|Science Fiction|Thriller
2           Adventure|Science Fiction|Thriller
3     Action|Adventure|Science Fiction|Fantasy
4                        Action|Crime|Thriller
Name: genres, dtype: object
```

In [36]:
```
# Number of unique entries in the genres column
imdb_df['genres'].nunique()
```

Out[36]:
```
1053
```

data in the genres column represents all the genres that each movie falls into which could be more than one genre separated by pipe |

there are 1053 unique genres in the genres column. That's a lot.

The genres column needs to be featured engineered to represent the dominant genre for each movie.

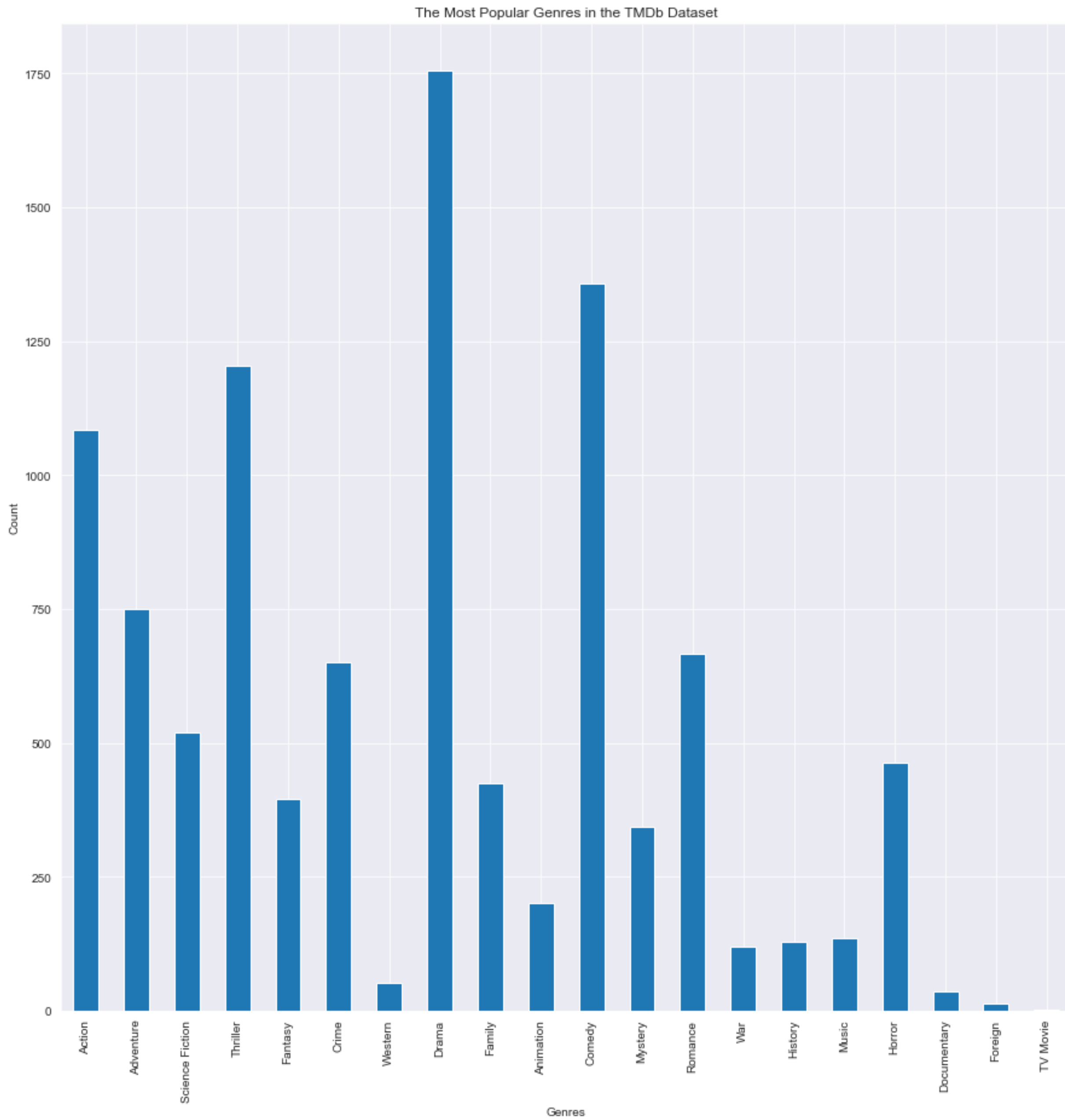
> This is not possible since we don't know which genre is dominant for those movies that belong to more than one genre.

What we can do right now is to break down the values in the genres column and count the number of occurrences using a dictionary

In [37]:
```
# counting the genres using a dictionary

genres_list = list(imdb_df['genres']) #converting the genres column into a list c
genres_dict = {} # empty dictionary

for genres in genres_list:
    for genre in genres.split("|"):
        if genre not in genres_dict:
            genres_dict[genre] = 1
        else:
            genres_dict[genre] += 1

print(genres_dict)
```

```
{'Action': 1085, 'Adventure': 749, 'Science Fiction': 519, 'Thriller': 1204, 'F
antasy': 396, 'Crime': 651, 'Western': 52, 'Drama': 1755, 'Family': 425, 'Anima
tion': 201, 'Comedy': 1357, 'Mystery': 344, 'Romance': 666, 'War': 119, 'Histor
y': 129, 'Music': 136, 'Horror': 463, 'Documentary': 35, 'Foreign': 12, 'TV Mov
ie': 1}
```

In [38]:
```
# converting genres dictionary counter into a pandas series object and plotting i

pd.Series(genres_dict).plot(kind='bar', figsize=(15, 15))
plt.xlabel("Genres")
plt.ylabel("Count")
plt.title("The Most Popular Genres in the TMDb Dataset")
plt.show()
```



Conclusion

From the .value_counts() operation and the visualization above, the most popular genre is Drama.

This is followed by Comedy, Thriller and Action as seen from the visualization.

Drama = 1729,

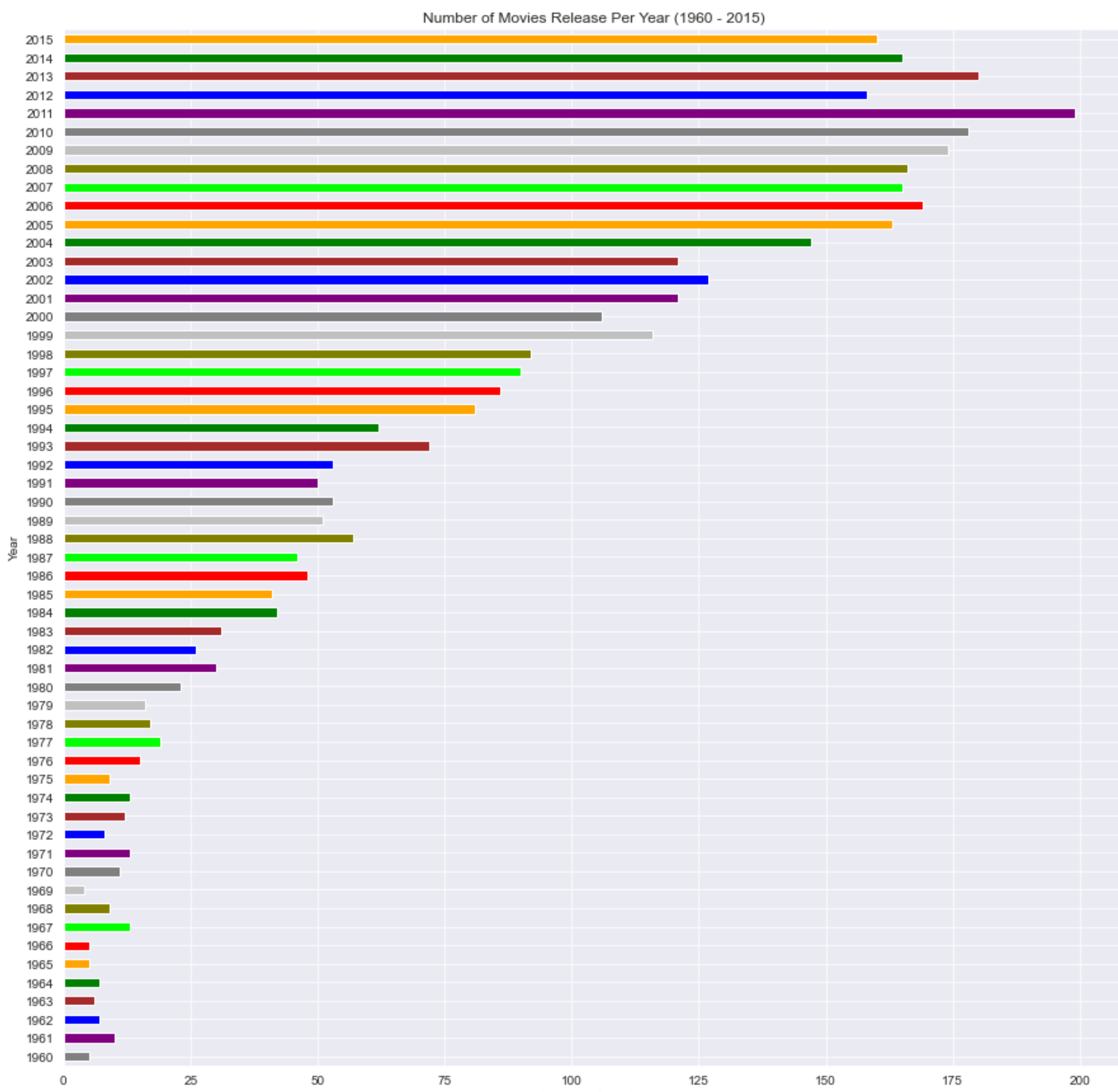
Comedy = 1335,

Thriller = 1194,

Action = 1076

Question 2: The number of Movies Released Each Year

In [39]:
```
# counting the number of movies released each year and visualizing it.

movies_per_year = imdb_df['release_year'].value_counts()

movies_per_year.sort_index().plot(kind='barh', color = colors, figsize=(15, 15))
plt.title("Number of Movies Release Per Year (1960 - 2015)")
plt.ylabel("Year")
plt.xlabel("Number of movies")
plt.show()
```



Conclusion

From the horizontal bar chat plotted above, the top three years with the highest number of movie releases are :

1. 2011
2. 2013
3. 2010

Question 3: Which is the most popular movie and the least popular movie and what features are associated with popular and less popular movies.

In [40]:
```
# getting the maximum value from the popularity column

max_pop = imdb_df['popularity'].max()

max_pop
```

Out[40]: 32.985763

In [41]:
```
# getting the minimum value from the popularity column

min_pop = imdb_df['popularity'].min()

min_pop
```

Out[41]: 0.001117

In [42]:
```
# The most popular movie

imdb_df.query('popularity == 32.985763')
```

Out[42]:

	id	popularity	budget	revenue	original_title	director	runtime	
0	135397	32.985763	150000000.0	1.513529e+09	Jurassic World	Colin Trevorrow	124.0	Action|Adventure Fictio

In [43]:
```
#The least popular movie

imdb_df.query('popularity == 0.001117')
```

Out[43]:

	id	popularity	budget	revenue	original_title	director	runtime	genres	releas
7268	1392	0.001117	350000.0	3515061.0	Born into Brothels	Zana Briski|Ross Kauffman	85.0	Documentary	2004

In [44]:
```
#getting the average popularity value

mean = imdb_df['popularity'].mean()
mean
```

Out[44]: 1.1918250228393457

In [45]:
```
# subsetting the imdb_df dataframe to get rows of popular and less popular movies

pop_movies = imdb_df[imdb_df['popularity'] >= mean]
less_pop = imdb_df[imdb_df['popularity'] < mean]
```

In [46]:
```
# looking at the first three rows of popular movies

pop_movies.head(3)
```

Out[46]:

	id	popularity	budget	revenue	original_title	director	runtime	
0	135397	32.985763	150000000.0	1.513529e+09	Jurassic World	Colin Trevorrow	124.0	Action|Adventure|S Fiction|
1	76341	28.419936	150000000.0	3.784364e+08	Mad Max: Fury Road	George Miller	120.0	Action|Adventure|S Fiction|
2	262500	13.112507	110000000.0	2.952382e+08	Insurgent	Robert Schwentke	119.0	Adventure|S Fiction|

In [47]:
```
#looking at the first three rows of less popular movies

less_pop.head(3)
```

Out[47]:

	id	popularity	budget	revenue	original_title	director	runtime	ge
136	277685	1.191138	1000000.0	62882090.0	Unfriended	Levan Gabriadze	82.0	Horror|T
137	268920	1.178831	35000000.0	51680201.0	Hot Pursuit	Anne Fletcher	87.0	Action|Crime|Co
138	280092	1.164724	10000000.0	104303851.0	Insidious: Chapter 3	Leigh Whannell	97.0	Drama|Horror|T

In [48]:
```
# the features associated with popular movies

pop_movies.describe()
```

Out[48]:

	id	popularity	budget	revenue	runtime	vote_count	vote_av
count	1199.000000	1199.000000	1.199000e+03	1.199000e+03	1199.000000	1199.000000	1199.0(
mean	55337.562135	2.498335	6.291791e+07	2.344117e+08	113.848207	1278.513761	6.5
std	84331.756717	2.080194	5.608043e+07	2.531090e+08	20.689459	1227.370123	0.7
min	5.000000	1.193916	2.700000e+04	2.500000e+02	63.000000	10.000000	3.7(
25%	1588.500000	1.444735	2.025000e+07	7.481606e+07	98.000000	500.500000	6.0(
50%	10140.000000	1.885979	4.500000e+07	1.557601e+08	110.000000	851.000000	6.5(
75%	72147.500000	2.729929	9.000000e+07	3.044872e+08	125.000000	1641.500000	7.1(
max	336004.000000	32.985763	3.800000e+08	2.781506e+09	201.000000	9767.000000	8.4(

In [49]:
```
# features of less popular movies

less_pop.describe()
```

Out[49]:

	id	popularity	budget	revenue	runtime	vote_count	vote_av
count	2654.000000	2654.000000	2.654000e+03	2.654000e+03	2654.000000	2654.000000	2654.0
mean	32917.807837	0.601582	2.559920e+07	5.047235e+07	107.113037	188.728335	6.0
std	56521.491259	0.293266	2.707802e+07	7.614461e+07	19.191519	272.941151	0.7
min	16.000000	0.001117	1.000000e+00	2.000000e+00	15.000000	10.000000	2.2
25%	8842.250000	0.369628	7.000000e+06	8.038173e+06	95.000000	46.000000	5.5
50%	11480.500000	0.577669	1.750000e+07	2.577330e+07	104.000000	112.000000	6.0
75%	27388.000000	0.833072	3.500000e+07	6.362171e+07	116.000000	234.750000	6.6
max	417859.000000	1.191235	4.250000e+08	1.123747e+09	338.000000	4368.000000	8.4

Conclusion

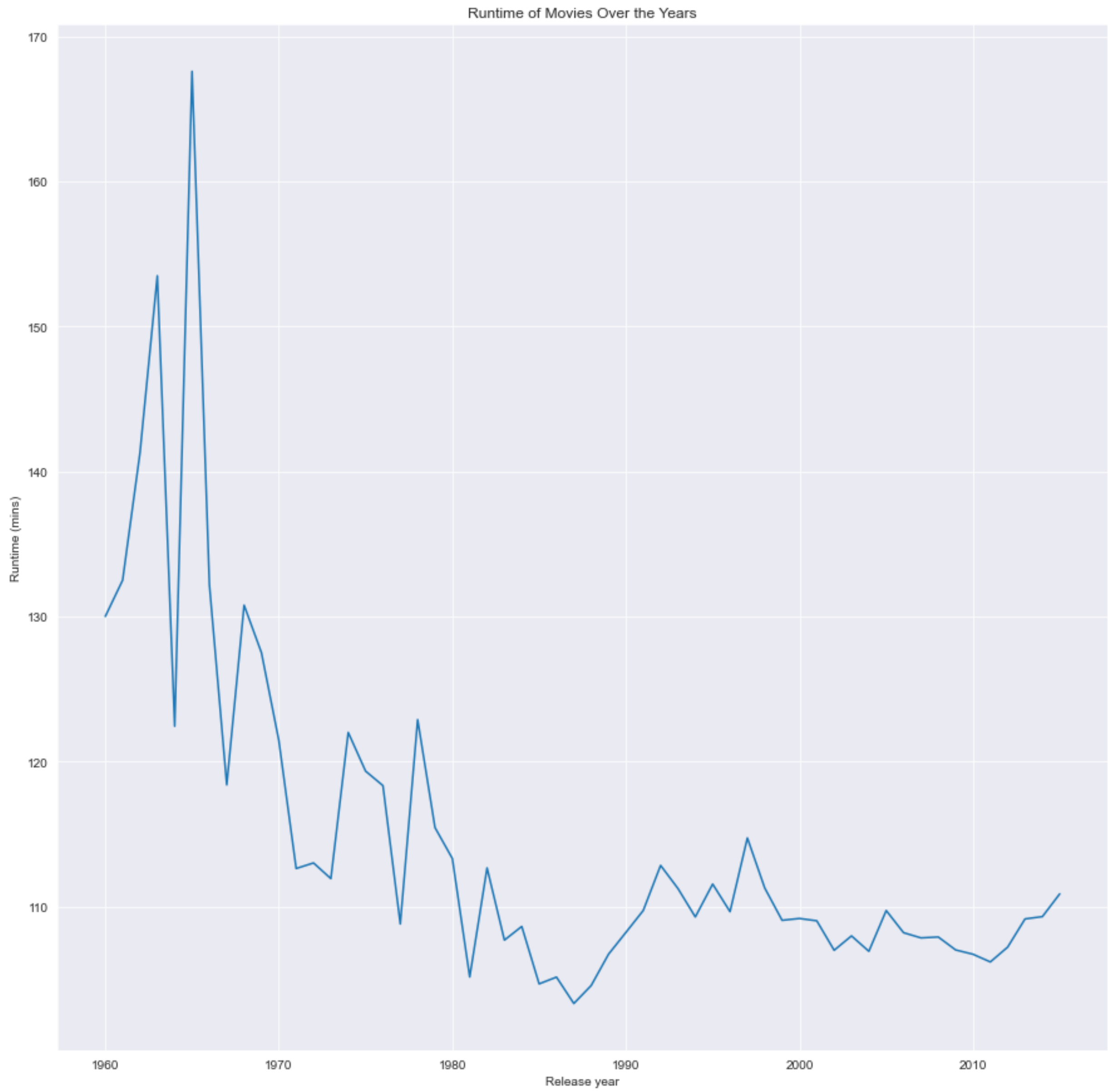
The features show that the average budget of popular movies is higher than that of less popular movies. The other features show little differences between the two categories.

Yes money is an important feature that is required to make a popular movie. What other features are important; story, director, cast??? This is an important question to pose when investigating further.

Question 4: Has the runtime of movies been declining over the

years

In [50]:
```
#Grouping by year and plotting the average runtime for each year

imdb_df.groupby('release_year')['runtime'].mean().plot(figsize=(15,15))
plt.xlabel("Release year")
plt.ylabel("Runtime (mins)")
plt.title("Runtime of Movies Over the Years")
plt.show()
```



This looks like the average runtime of movies have been declining over the years. Confirming with a scatter plot

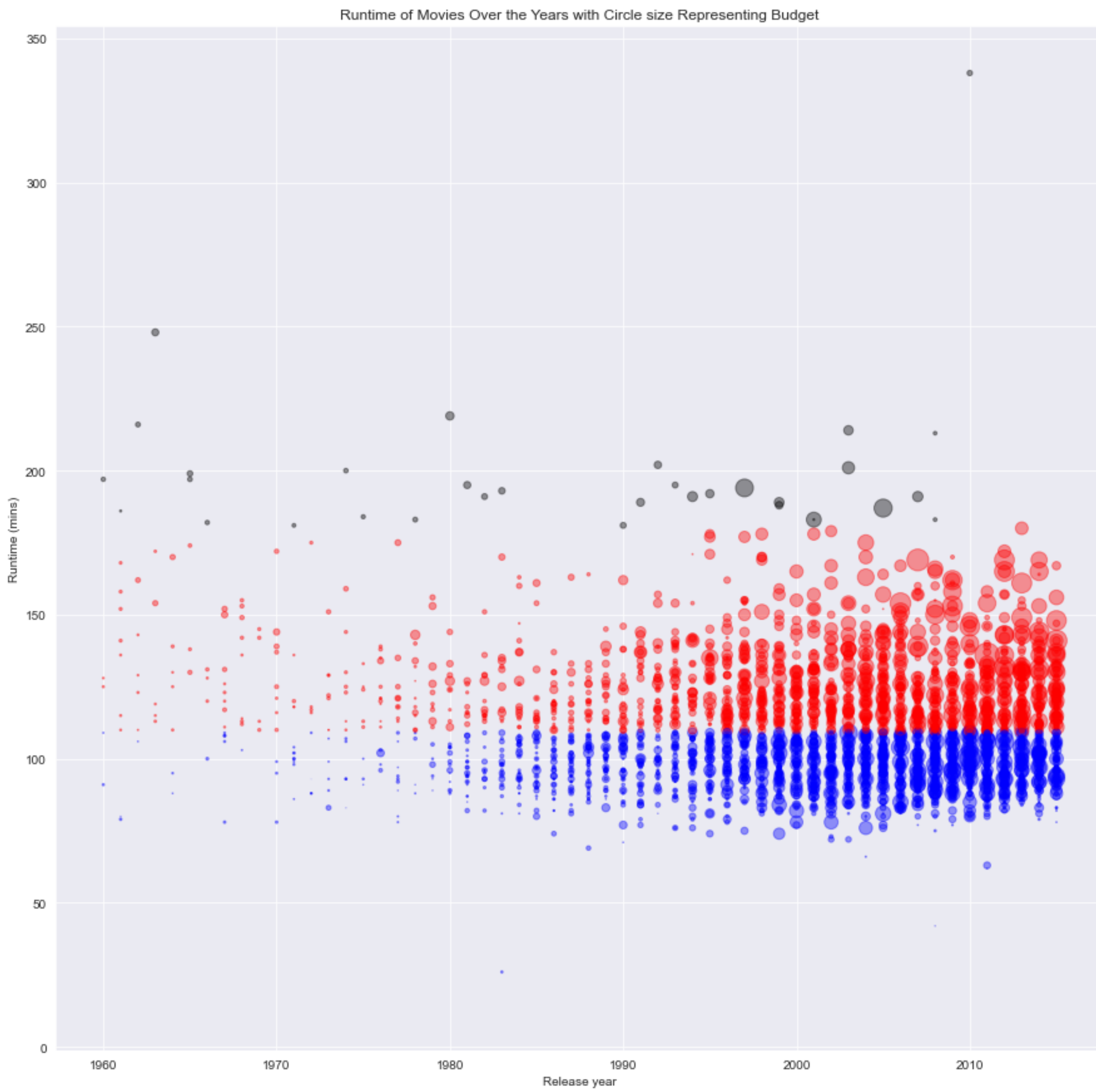
In [51]:
```
#plotting a scatter plot of runtime as a function of release_year

imdb_df.plot(kind='scatter', x='release_year', y='runtime', figsize=(15,15))
plt.xlabel("Release year")
plt.ylabel("Runtime (mins)")
plt.title("Runtime of Movies Over the Years")
plt.show()
```

Movie runtime doesn't appear to be declining. can we make the plot a little clearer? seems there are movies with short runtime.

In [52]:
```
# Comparing runtime with average runtime to create a list of colors to be  used c

avg_runtime = imdb_df['runtime'].mean() #average run time
colors = [] # empty list

for lab, row in imdb_df.iterrows():
    if row['runtime'] < avg_runtime:
        colors.append('blue')
    elif avg_runtime <= row['runtime'] <= 180:
        colors.append('red')
    else:
        colors.append('black')

# printing a list of the first fifty colors
print(colors[:50])

s = imdb_df['budget'] / 1000000 # dividing budget values by one million to be use
```

```
['red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'blue', 'blue', 'red',
'red', 'blue', 'blue', 'red', 'red', 'blue', 'red', 'red', 'red', 'red', 'red',
'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'red', 'b
lue', 'red', 'blue', 'red', 'red', 'red', 'blue', 'red', 'red', 'red', 'blue',
'blue', 'blue', 'red', 'red', 'red', 'red']
```

In [53]:
```
# Plotting with color and size

imdb_df.plot(kind='scatter', x='release_year', y='runtime', s=s, color=colors, al
plt.xlabel("Release year")
plt.ylabel("Runtime (mins)")
plt.title("Runtime of Movies Over the Years with Circle size Representing Budget'
plt.show()
```



Movies with runtime > 3 hrs = Black

Movies with runtime > average run time <= 3 hrs = Red

Movies with runtime < average run time = Blue

Conclusion

From the line plot, we see that the average runtime of movies have been declining over the years. But after taking a closer look with a scatter plot we see that that is not the case.

From the scatter plots we see that movie lengths have practically remained the same over the years and that most movies with long runtime have large budgets.

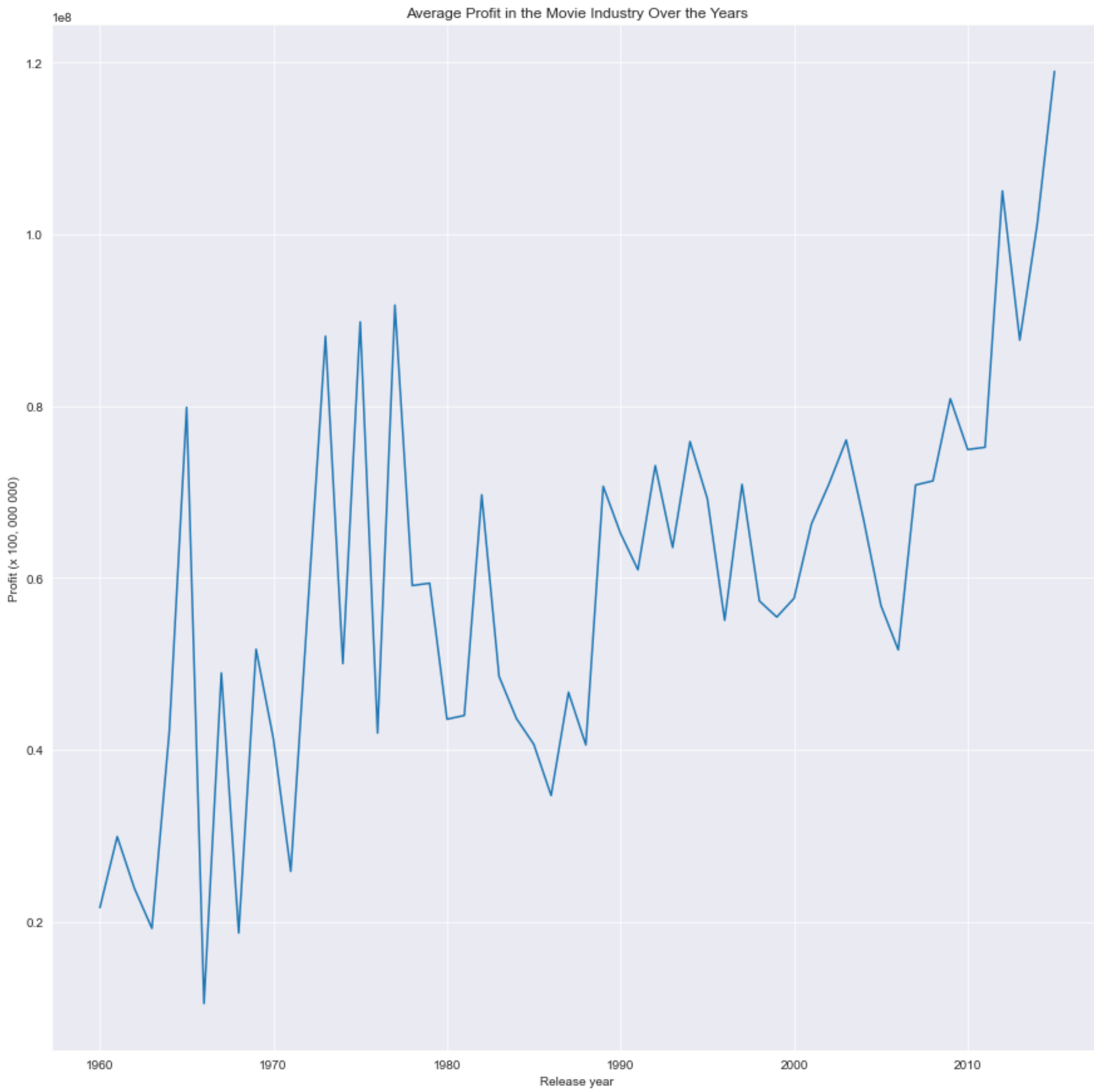
Question 5: Is the Movie industry making or loosing money and what is the relationship between budget and popularity?

In [54]:
```
# calculating and adding the profit column to the imdb_df dataframe
imdb_df['profit'] = imdb_df['revenue'] - imdb_df['budget']

imdb_df.head(3)
```

Out[54]:

opularity	budget	revenue	original_title	director	runtime	genres	releas
2.985763	150000000.0	1.513529e+09	Jurassic World	Colin Trevorrow	124.0	Action|Adventure|Science Fiction|Thriller	201
8.419936	150000000.0	3.784364e+08	Mad Max: Fury Road	George Miller	120.0	Action|Adventure|Science Fiction|Thriller	201
3.112507	110000000.0	2.952382e+08	Insurgent	Robert Schwentke	119.0	Adventure|Science Fiction|Thriller	201

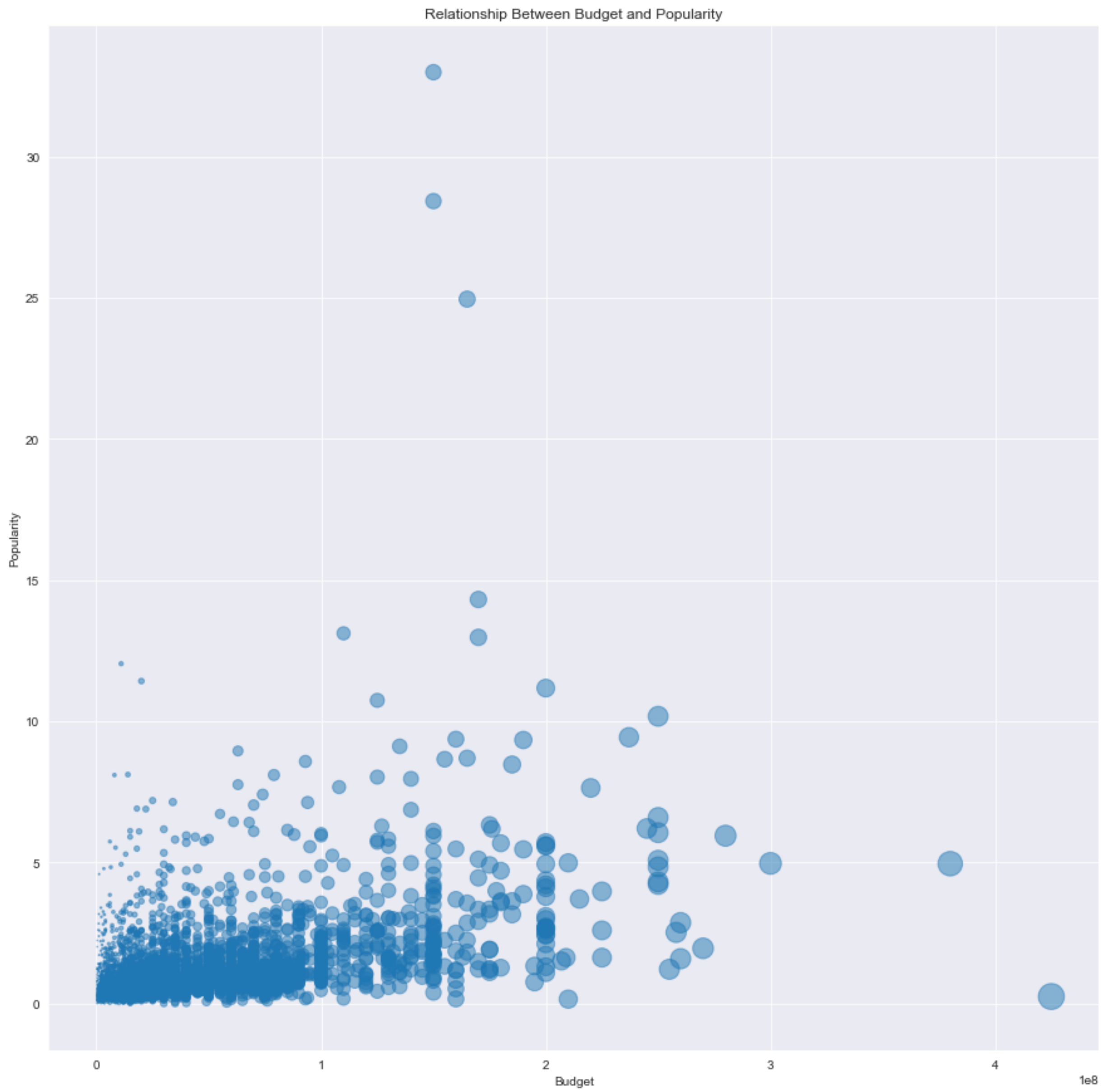
In [55]:
```
# Grouping by year and calculating the average profit over the years. Then visual

imdb_df.groupby('release_year')['profit'].mean().plot(figsize=(15, 15))
plt.xlabel("Release year")
plt.ylabel("Profit (x 100, 000 000)")
plt.title("Average Profit in the Movie Industry Over the Years")
plt.show()
```



The line plot shows that the movie industry has been making profit over the years

Relationship between budget and popularity

In [56]:
```
# creating a scatter plot of popularity as a function of budget

imdb_df.plot(kind='scatter', x='budget', y='popularity', s=s, alpha=0.5, figsize=
plt.xlabel("Budget")
plt.ylabel("Popularity")
plt.title("Relationship Between Budget and Popularity")
plt.show()
```



The scatter plot shows that a relationship exists between budget and poularity.

Conclusion

The line plot shows that profit in the movie industry has been increasing over the years and not declining.

Their exists a relationship between budget and popularity however, at this point in time it is not clear how weak or strong that relationship is.

```
<a id='conclusions'></a>
## Conclusions
<br>
<b> It is important to state that the methods and approaches used to answer the research questions in this project and the result obtained are tentative and subject to confirmation by someone who is a qualified data analyst.
 <br>
<br>
In conclusion, we were able:

> To identify the most popular genre, we made two different attempts:
> > **Attempt 1**: we used the groupby() function on the release_year and genres columns and using the count() function to count the number of movie id for each year. <b>The result of this operation showed that "Drama" is the most popular genre.</b> <br>
<br>
> > **Attempt 2**: Given that values in the genres column consists of multiple genres for movies that belong to more than one genre separated by pipe (|), We had to separate these combined genres into unique values and count the number of their occurrence using a dictionary. <b> The result also showed that "Drama" is the most popular genre</b>

> To identify the year with highest number of movie releases. <br/>
> > The groupby() function was used on the release_year column with the value_counts() function to get the number movies release each year. The result was then sorted by index in descending order and plotted using an horizontal bar chart. <b> 2011 was shown as the year with highest number of movie releases </b>


> To identify the most popular movie and the least popular movie and what features are associated with popular and less popular movies.<br/>
> >**The most Popular and least popular movie**: The maximum and minimum popularity values were obtained using the max() and min() functions on the popularity column of the dataframe. These values were then used filter for the most popular movie and least popular movie respectively. <br/>
<br>
> > **Features of popular and less popular movies**: The average popularity value was calculated using the mean() function which was used to create two dataframe subsets of popular movies and less popular movies. The .describe() function was then used to print summary statistics of both subsets. <b>The features show that the average budget of popular movies is higher than that of less popular movies. The other features show little differences between the two categories.</b>

> To determine that the runtime of movies haven't been declining over the years.
<br>
> > We grouped the dataset by the release_year column and calculated the average runtime each year which we plotted using a line plot. <b>The line plot showed that the average movie runtime has been declining over the years.</b>
<br>
```


> > Confirming the decline of movie runtime over the years, a scatter plot was created. The visualization showed that movie runtime hasn't being declining over the years The visualization was made clearer with color added to the plot based on a filter of average runtime together with the size attribute based on budget divided by 1000000.

> To determine if the movie industry has being making or loosing money over the years and the relationship between budget and popularity.

> > The profit made by each movie was calculated and added to the dataframe as new column. The dataframe was then grouped by release_year column with the average profit calculated. The result was then visualized using a line plot. The result showed that the movie industry has been making profit over the years

> >**Budget and Popularity**: The popularity column was plotted as a function of the budget column using a scatter plot with the size attribute of the scatter plot set to budget/1000000. The result showed that there is a relationship between budget and popularity. However, we were unable to determine how strong this relationship is as at this time.

References

1. https://github.com/deepak525/Investigate_TMDb_Movies
2. https://github.com/deepak525/Investigate_TMDb_Movies/blob/master/investigate_the_TBMb_D
3. https://www.kaggle.com/code/deepak525/investigate-tmdb-movie-dataset/notebook
4. https://github.com/deepak525/Investigate_TMDb_Movies/blob/master/investigate_the_TBMb_D
5. https://matplotlib.org/3.5.0/gallery/shapes_and_collections/scatter.html#sphx-glr-gallery-shapes-and-collections-scatter-py