

Project: Movie Data Analysis

by Natalia Dudek

Table of Contents

- [Introduction](#)
- [Data Wrangling](#)
- [Exploratory Data Analysis](#)
- [Conclusions](#)

Introduction

In this project I will be analysing movie dataset. In particular, I will be interested in trying to find the most popular movie genre and how popularity of movie genres has been changing over years.

I will also check which genre has the highest release of movies.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Data Wrangling

After looking into data set, I will check which columns we are going to use and which we can remove from our data set. I will also check if there are any duplicated rows and missing values.

General Properties

At first, I will load my data and print out a few lines to see what the dataset looks like.

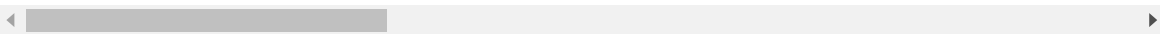
In [2]:

```
df=pd.read_csv('tmdb-movies.csv')
df.head(2)
```

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...	http://
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	Tom Hardy Charlize Theron Hugh Keays-Byrne Nic...	http://w

2 rows × 21 columns



Observation

In the dataset preview above, I can see some columns which I don't need for this analysis (id, imdb_id, homepage, tagline, overview, release_date). I will drop those columns later, in the cleaning section.*

In the 'genres' column we can see multiple genres assigned to one movie in one row. We need to split it to receive one genre in each row.

Next, I need to see how many rows and columns the dataset has.

In [3]:

```
df.shape
```

Out[3]:

(10866, 21)

So we have 10866 movies and 21 columns. Let's have a look at the empty cells for each column:

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
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
```

As we can see above, there is a lot of data missing in columns: 'homepage', 'tagline', 'keywords' and 'production_companies'. I wasn't planning to use those columns and will drop them later anyway.

In my analysis I need genres column, and there is 23 values missing in that column. Let's have a look at this:

In [5]:

```
to_drop=df[df['genres'].isnull()==True]  
to_drop
```

Out[5]:

	id	imdb_id	popularity	budget	revenue	original_title	cast
424	363869	tt4835298	0.244648	0	0	Belli di papÃ	Diego Abatantuono Matilde Gioli Andrea Pisani ...
620	361043	tt5022680	0.129696	0	0	All Hallows' Eve 2	NaN
997	287663	NaN	0.330431	0	0	Star Wars Rebels: Spark of Rebellion	Freddie Prinze Jr. Vanessa Marshall Steve Blum...
1712	21634	tt1073510	0.302095	0	0	Prayers for Bobby	Ryan Kelley Sigourney Weaver Henry Czerny Dan ...
1897	40534	tt1229827	0.020701	0	0	Jonas Brothers: The Concert Experience	Nick Jonas Joe Jonas Kevin Jonas John Lloyd Ta...
2370	127717	tt1525359	0.081892	0	0	Freshman Father	Britt Irvin Merrilyn Gann Barbara Tyson Anthon...
2376	315620	tt1672218	0.068411	0	0	Doctor Who: A Christmas Carol	Matt Smith Karen Gillan Arthur Darvill Michael...
2853	57892	tt0270053	0.130018	0	0	Vizontele	YÃ±Imaz ErdoÃan Demet Akbag Altan Erkekli Cem...
3279	54330	tt1720044	0.145331	0	0	ì¸,ê¸,°ì€ ë¸	Jang Keun-suk Song Ha-yoon Kim Jeong-Nan
4547	123024	tt2305700	0.520520	0	0	London 2012 Olympic Opening Ceremony: Isles of...	Queen Elizabeth II Mike Oldfield Kenneth Brana...
4732	139463	tt2084977	0.235911	0	0	The Scapegoat	Andrew Scott Jodhi May Eileen Atkins Matthew R...
4797	369145	NaN	0.167501	0	0	Doctor Who: The Snowmen	Matt Smith Jenna Coleman Richard E. Grant Ian ...

	id	imdb_id	popularity	budget	revenue	original_title	cast
4890	126909	tt2219564	0.083202	0	0	Cousin Ben Troop Screening	Jason Schwartzman
5830	282848	tt2986512	0.248944	0	0	Doctor Who: The Time of the Doctor	Matt Smith Jenna Coleman
5934	200204	tt2808968	0.067433	0	0	Prada: Candy	Peter Gadiot Rodolphe Pauly L�a Seydoux
6043	190940	tt2797242	0.039080	0	0	Bombay Talkies	Aamir Khan Rani Mukerji Randeep Hooda Saqib Sa...
6530	168891	tt0818519	0.092724	0	0	Saw Rebirth	Whit Anderson Stan Kirsch Jeff Shuter George W...
8234	56804	tt0114844	0.028874	0	0	Viaggi di nozze	Carlo Verdone Claudia Gerini Veronica Pivetti ...
8614	65595	tt0117880	0.273934	0	0	T2 3-D: Battle Across Time	Arnold Schwarzenegger Linda Hamilton Edward Fu...
8878	92208	tt0250593	0.038045	0	0	Mom's Got a Date With a Vampire	Matt O'Leary Laura Vandervoort Myles Jeffrey C...
9307	141859	tt0097446	0.094652	0	0	Goldeneye	Charles Dance Phyllis Logan Patrick Ryecart La...
9799	48847	tt0193716	0.175008	0	0	The Amputee	Catherine E. Coulson David Lynch
10659	4255	tt0065904	0.344172	5000	0	The Party at Kitty and Stud's	Sylvester Stallone Henrietta Holm Nicholas War...

23 rows × 21 columns

Because there is only 23 of those out of 10866, and it looks like those movies aren't very popular (which we can see in 'popularity' column and 'vote_count' column), I am going to drop those rows. But first I will check what are the mean values for 'popularity' and 'vote_count' columns:

In [6]:

```
df.popularity.describe()
```

Out[6]:

```
count      10866.000000
mean         0.646441
std          1.000185
min          0.000065
25%          0.207583
50%          0.383856
75%          0.713817
max          32.985763
Name: popularity, dtype: float64
```

Mean value for 'popularity' is 0.646446 and majority of values are between 0.207575 and 0.713817, which is higher than most of the entries above without specified movie genre.

In [7]:

```
df.vote_count.describe()
```

Out[7]:

```
count      10866.000000
mean       217.389748
std        575.619058
min         10.000000
25%         17.000000
50%         38.000000
75%        145.750000
max        9767.000000
Name: vote_count, dtype: float64
```

Mean value for 'vote_count' is 217.399632 and majority of values are between 17 and 9767, which is higher than most of the entries above without specified movie genre. I will drop those rows later.

Also we need to check if there are any duplicated rows:

In [8]:

```
df[df.duplicated()==True]
```

Out[8]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepag
2090	42194	tt0411951	0.59643	30000000	967000	TEKKEN	Jon Foo Kelly Overton Cary-Hiroyuki Tagawa Ian...	Na

1 rows × 21 columns

There is one duplicated row (which means all the information for each column are duplicated), so I need to drop this row as well.

In [9]:

```
df.describe()
```

Out[9]:

	id	popularity	budget	revenue	runtime	vote_count
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.000000
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.389748
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.619058
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.000000
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.000000
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.000000
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.750000
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.000000

Observation

Drop duplicated row and rows with missing values in 'genre' column.

Data Cleaning

After observing the data and figuring out the questions we are going to analyse, we need to clean the data first.

We need to:

- remove not needed columns;
- remove duplicated and empty rows;
- split string in 'genres' column to get one genre in one row.

Removing not needed columns

As I mentioned above, in my analysis I do not need the following columns: 'id', 'imdb_id', 'homepage', 'keywords', 'tagline', 'cast', 'director', 'overview', 'release_date' and 'production_companies' so am going to clean those columns:

In [10]:

```
df.drop(['id', 'imdb_id', 'homepage', 'keywords', 'tagline', 'cast', 'director', 'overview', 'release_date', 'production_companies'], axis=1, inplace=True)
```

Removing duplicated and empty rows

I will also drop duplicated row, which we found earlier, and after that check if it worked by running `df.duplicated` again:

In [11]:

```
df=df.drop_duplicates()  
df.duplicated().sum()
```

Out[11]:

0

Great, that worked and we do not have duplicated rows anymore.

We also need to drop rows with missing values in 'genres' column:

In [12]:


```
df.drop(to_drop.index, axis=0, inplace=True)
```

Let's double check if it worked

In [13]:

```
df[df['genres'].isnull()==True]
```

Out[13]:

popularity	budget	revenue	original_title	runtime	genres	vote_count	vote_average	rele
								

After dropping those columns and rows, I will check what does it look like now:

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10842 entries, 0 to 10865
Data columns (total 11 columns):
#   Column          Non-Null Count  Dtype
---  -
0   popularity      10842 non-null  float64
1   budget          10842 non-null  int64
2   revenue         10842 non-null  int64
3   original_title  10842 non-null  object
4   runtime         10842 non-null  int64
5   genres          10842 non-null  object
6   vote_count      10842 non-null  int64
7   vote_average    10842 non-null  float64
8   release_year    10842 non-null  int64
9   budget_adj      10842 non-null  float64
10  revenue_adj      10842 non-null  float64
dtypes: float64(4), int64(5), object(2)
memory usage: 1016.4+ KB
```

As we can see we have 11 columns and 10842 movies now.

In [15]:

```
df.isnull().sum()
```

Out[15]:

```
popularity      0
budget          0
revenue         0
original_title  0
runtime         0
genres          0
vote_count      0
vote_average    0
release_year    0
budget_adj      0
revenue_adj     0
dtype: int64
```

Split string in 'genres' column to get one genre in one row

I will first check what data type is the 'genre' column.

In [16]:

```
type(df['genres'][0])
```

Out[16]:

str

'Genres' column is a string.

In the preview above I saw that there are multiple genre types assign to one movie title (separated by '|').
Let's have a look at those:

In [17]:

```
df_m=df[df['genres'].str.contains("\|")]
df_m
```

Out[17]:

	popularity	budget	revenue	original_title	runtime	genres	vo
0	32.985763	150000000	1513528810	Jurassic World	124	Action Adventure Science Fiction Thriller	
1	28.419936	150000000	378436354	Mad Max: Fury Road	120	Action Adventure Science Fiction Thriller	
2	13.112507	110000000	295238201	Insurgent	119	Adventure Science Fiction Thriller	
3	11.173104	200000000	2068178225	Star Wars: The Force Awakens	136	Action Adventure Science Fiction Fantasy	
4	9.335014	190000000	1506249360	Furious 7	137	Action Crime Thriller	
...
10858	0.317824	0	0	The Russians Are Coming, The Russians Are Coming	126	Comedy War	
10859	0.089072	0	0	Seconds	100	Mystery Science Fiction Thriller Drama	
10862	0.065543	0	0	Grand Prix	176	Action Adventure Drama	
10863	0.065141	0	0	Beregis Avtomobilya	94	Mystery Comedy	
10864	0.064317	0	0	What's Up, Tiger Lily?	80	Action Comedy	

8514 rows × 11 columns



There is 8514 movies which has multiple genres type. To be able to check which genres are getting most popular I need to split it by '|', and have only one genre in one row.

Let's check what is the maximum number of genres assign to one movie (row):

In [18]:

```
df_m.genres.str.count('\|').max()
```

Out[18]:

4

Maximum of '|' occurrences is 4, which means there is maximum of 5 genres assign to one movie.

I will create 5 copies of the dataframe:

In [20]:

```
df_1=df_m.copy()  
df_2=df_m.copy()  
df_3=df_m.copy()  
df_4=df_m.copy()  
df_5=df_m.copy()
```

Now I will split string in 'genres' column

In [21]:

```
df_1['genres']=df_1['genres'].str.split("\|").str[0]  
df_2['genres']=df_2['genres'].str.split("\|").str[1]  
df_3['genres']=df_3['genres'].str.split("\|").str[2]  
df_4['genres']=df_4['genres'].str.split("\|").str[3]  
df_5['genres']=df_5['genres'].str.split("\|").str[4]
```

Next, I will drop the original rows with multiple genres in one row:

In [22]:

```
df.drop(df_m.index, inplace=True)
```

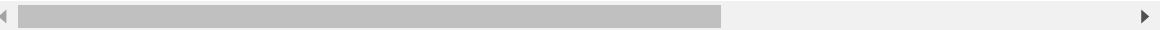
In [23]:

df

Out[23]:

	popularity	budget	revenue	original_title	runtime	genres	vote_count	vote_average
25	4.566713	150000000	682330139	Mission: Impossible - Rogue Nation	131	Action	2349	7.8
26	4.564549	68000000	215863606	Ted 2	115	Comedy	1666	7.2
51	2.814802	35000000	194564672	The Intern	121	Comedy	1255	7.1
55	2.584264	20000000	36606743	Burnt	100	Drama	631	6.8
58	2.557859	15000000	64191523	The Danish Girl	120	Drama	867	6.7
...
10846	0.212716	0	0	Dracula: Prince of Darkness	90	Horror	16	5.5
10852	0.227220	0	0	A Big Hand for the Little Lady	95	Western	11	5.4
10860	0.087034	0	0	Carry On Screaming!	87	Comedy	13	5.4
10861	0.080598	0	0	The Endless Summer	95	Documentary	11	5.4
10865	0.035919	19000	0	Manos: The Hands of Fate	74	Horror	15	5.3

2328 rows × 11 columns



Now we have to add in our newly separated rows:

In [24]:

```
frames = [df_1, df_2, df_3,df_4, df_5]
new_rows = pd.concat(frames)

df=df.append(new_rows, ignore_index=True)
```

In [25]:

df['genres'].isnull().sum()

Out[25]:

17943

We have 17943 empty rows now so we have to drop them:

In [26]:

```
df.drop(df[df['genres'].isnull()==True].index, axis=0, inplace=True)
```

In [27]:

```
df['genres'].isnull().values.any()
```

Out[27]:

False

As we can see above, all the empty rows were dropped and now we don't have any left.

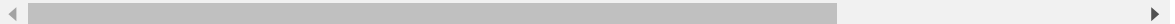
Next I will check if all the original rows with "/" are gone:

In [28]:

```
df[df['genres'].str.contains("\|")]
```

Out[28]:

popularity	budget	revenue	original_title	runtime	genres	vote_count	vote_average	rele
------------	--------	---------	----------------	---------	--------	------------	--------------	------



In [29]:

df

Out[29]:

	popularity	budget	revenue	original_title	runtime	genres	vote_count	vote_
0	4.566713	150000000	682330139	Mission: Impossible - Rogue Nation	131	Action	2349	
1	4.564549	68000000	215863606	Ted 2	115	Comedy	1666	
2	2.814802	35000000	194564672	The Intern	121	Comedy	1255	
3	2.584264	20000000	36606743	Burnt	100	Drama	631	
4	2.557859	15000000	64191523	The Danish Girl	120	Drama	867	
...
44866	0.410366	1377800	0	Batman	105	Crime	99	
44874	0.299911	12000000	20000000	The Sand Pebbles	182	Romance	28	
44880	0.252399	0	0	Khartoum	134	Action	12	
44886	0.202473	0	0	Harper	121	Mystery	14	
44887	0.342791	0	0	Born Free	95	Foreign	15	

26955 rows × 11 columns



There is no multiple genres in 'genres' column in our Dataframe anymore. Our dataframe has 26955 rows now.

Exploratory Data Analysis

Which genres are most popular from year to year?

As we can see on 'Jurassic Word' example, there is one genre assigned to one movie title in each row.

In [30]:

```
df.query('original_title=="Jurassic World"')
```

Out[30]:

	popularity	budget	revenue	original_title	runtime	genres	vote_count	vote
2328	32.985763	150000000	1513528810	Jurassic World	124	Action	5562	
10842	32.985763	150000000	1513528810	Jurassic World	124	Adventure	5562	
19356	32.985763	150000000	1513528810	Jurassic World	124	Science Fiction	5562	
27870	32.985763	150000000	1513528810	Jurassic World	124	Thriller	5562	

There is 20 unique genres.

In [31]:

```
df.genres.nunique()
```

Out[31]:

20

As we have no longer multiple genres in one row, now we can check what genres are getting more popular from year to year.

First, I will group the data by 'realease_year' and then 'genres' to check mean values:

In [32]:

```
year_pop=df.groupby(['release_year','genres']).mean()
year_pop
```

Out[32]:

		popularity	budget	revenue	runtime	vote_count	v
release_year	genres						
1960	Action	0.590724	1.750000e+06	8.113125e+06	137.000000	65.875000	
	Adventure	0.700981	5.500000e+05	9.810000e+05	124.200000	82.000000	
	Comedy	0.396000	7.537500e+05	6.012500e+06	98.625000	44.875000	
	Crime	0.346480	0.000000e+00	0.000000e+00	119.000000	25.500000	
	Drama	0.566305	1.215919e+06	9.461538e+06	133.076923	138.153846	
...	
2015	Science Fiction	2.245603	2.850500e+07	1.118263e+08	94.930233	682.732558	
	TV Movie	0.260574	1.500000e+05	0.000000e+00	86.250000	25.700000	
	Thriller	1.401877	1.242135e+07	4.529045e+07	98.356725	343.403509	
	War	1.284511	2.377778e+07	7.845509e+07	104.000000	384.222222	
	Western	3.178796	4.196667e+07	1.148233e+08	124.666667	1178.666667	

1049 rows × 8 columns



Next I will group by 'release_year' again and find the largest popularity value for each year.

In [33]:

```
year_pop.groupby(['release_year']).popularity.nlargest(1)
```

Out[33]:

release_year	release_year	genres	
1960	1960	Thriller	0.811910
1961	1961	Animation	2.631987
1962	1962	Adventure	0.942513
1963	1963	Animation	2.180410
1964	1964	War	0.930959
1965	1965	Music	0.968850
1966	1966	Animation	0.585717
1967	1967	Animation	1.348805
1968	1968	Mystery	1.519456
1969	1969	Crime	0.948020
1970	1970	Animation	1.127718
1971	1971	Family	1.530722
1972	1972	Crime	1.072768
1973	1973	Animation	0.956526
1974	1974	Mystery	0.702035
1975	1975	Adventure	0.880297
1976	1976	Crime	0.707249
1977	1977	Action	1.419319
1978	1978	Music	0.679805
1979	1979	Action	1.410014
1980	1980	Science Fiction	0.897143
1981	1981	Adventure	0.875815
1982	1982	War	1.143183
1983	1983	Adventure	0.900596
1984	1984	Family	0.823924
1985	1985	Family	0.924311
1986	1986	Adventure	0.798935
1987	1987	History	0.815643
1988	1988	Action	0.599017
1989	1989	Animation	1.177585
1990	1990	Adventure	0.801768
1991	1991	Animation	1.665002
1992	1992	Animation	1.286893
1993	1993	Fantasy	0.918601
1994	1994	Crime	1.297888
1995	1995	Animation	1.467780
1996	1996	Crime	0.976838
1997	1997	Science Fiction	1.140241
1998	1998	War	1.246619
1999	1999	Adventure	1.012306
2000	2000	Adventure	0.854593
2001	2001	Fantasy	1.565260
2002	2002	Fantasy	1.430465
2003	2003	Fantasy	1.747524
2004	2004	Fantasy	1.320568
2005	2005	Fantasy	1.117732
2006	2006	Fantasy	1.023134
2007	2007	Fantasy	0.957349
2008	2008	Adventure	1.008385
2009	2009	Adventure	1.138422
2010	2010	Adventure	1.360319
2011	2011	Western	1.175800
2012	2012	Western	1.732778
2013	2013	Adventure	1.260832
2014	2014	Adventure	2.430526
2015	2015	Adventure	3.283786

Name: popularity, dtype: float64

We have created a list of most popular genres in each year. As we can see above the most popular movie genre in the past 3 years is Adventure (from 2013 to 2015).

Next I will check how the popularity for each genre has been changing from year to year. To do that, let's group our original DataFrame by 'genres' and then by 'release_year' and check mean values:

In [34]:

```
genres_pop=df.groupby(['genres','release_year']).mean()
genres_pop
```

Out[34]:

		popularity	budget	revenue	runtime	vote_count	vot
genres	release_year						
Action	1960	0.590724	1.750000e+06	8.113125e+06	137.000000	65.875000	
	1961	0.365913	2.285714e+06	4.742857e+06	134.000000	30.285714	
	1962	0.708945	3.262500e+06	1.370000e+07	125.375000	100.250000	
	1963	0.951729	7.225000e+06	2.222469e+07	136.000000	140.750000	
	1964	0.813087	7.000000e+05	2.498000e+07	130.600000	145.600000	
...	
Western	2011	1.175800	9.933333e+07	1.403902e+08	108.000000	931.000000	
	2012	1.732778	2.925000e+07	1.063421e+08	108.250000	1884.750000	
	2013	0.665255	8.733333e+07	3.010910e+07	113.333333	579.666667	
	2014	0.673947	2.666667e+06	4.049982e+05	90.833333	227.166667	
	2015	3.178796	4.196667e+07	1.148233e+08	124.666667	1178.666667	

1049 rows × 8 columns

Now I will create an array of unique genres.

In [35]:

```
genre_list=df.genres.unique()
genre_list
```

Out[35]:

```
array(['Action', 'Comedy', 'Drama', 'Thriller', 'Horror', 'Documenta
ry',
      'TV Movie', 'Animation', 'Crime', 'Adventure', 'Family', 'Rom
ance',
      'Science Fiction', 'Music', 'Western', 'Fantasy', 'War', 'For
eign',
      'History', 'Mystery'], dtype=object)
```

Next, create a plots for each genre to see how popularity for each genre has been changing over years.

In [36]:

```
fig, axs = plt.subplots(5,4, figsize=(18, 18), sharex=True, sharey=True)

for i, ax in enumerate(axs.flat):
    ax.plot(genres_pop.popularity[genre_list[i]])
    ax.set_title(genre_list[i])

plt.setp(axs[-1, :], xlabel='Year'); # I want to label only last row
plt.setp(axs[:, 0], ylabel='Popularity'); # just label first column
```



As we can see above, in the past few years Adventure movies are very popular, which we have also seen in the table above. We can also see that back in early 60's Animation movies were very popular.

Which genre has the highest release of movies?

To check which genre has the highest release of movies, I am going to use `value_counts`:

In [37]:

```
genre_release=df.genres.value_counts()  
genre_release
```

Out[37]:

Drama	4760
Comedy	3793
Thriller	2907
Action	2384
Romance	1712
Horror	1637
Adventure	1471
Crime	1354
Family	1231
Science Fiction	1229
Fantasy	916
Mystery	810
Animation	699
Documentary	520
Music	408
History	334
War	270
Foreign	188
TV Movie	167
Western	165

Name: genres, dtype: int64

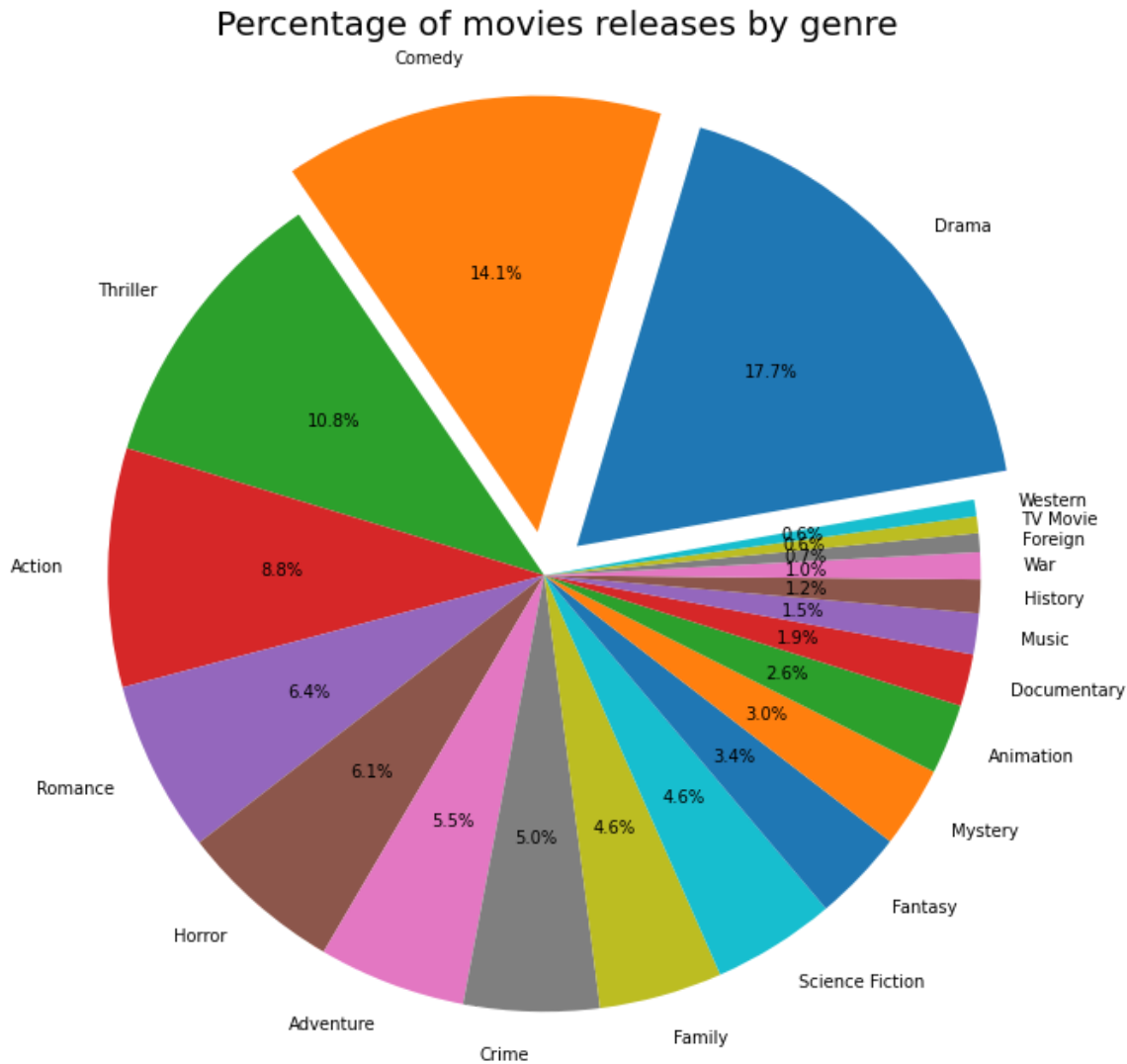
Next we will create the pie chart, and use percents on the visualisation.

In [38]:

```
labels = genre_release.index
sizes = genre_release
explode = (0.1, 0.1, 0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) # only "explode" the
first 2 slices

fig, ax = plt.subplots(figsize = (11,11))
ax.pie(sizes,labels=labels, autopct='%1.1f%%', startangle=10, explode=explode)
ax.axis('equal') # to be sure,that pie is drawn as a circle
ax.set_title('Percentage of movies releases by genre', fontsize=20)

plt.show()
```



As we can see above, Drama genre has the most releases, followed by Comedy genre.

Conclusions

- Adventure movies are the most popular genre;
- Drama and Comedy are the genres which have the most releases;

Limitations

In the original dataset there was many genres assigned to one movie title. In my analysis I had to split the genre column by ' | ', to get one genre per row. Because of that the dataset which I was working on had much more rows and it increased the calculation time.