Final Project Submission ¶

Please fill out:

- Student name: Guofa Shou
- Student pace: self paced
- Scheduled project review date/time:
- Instructor name: Jeff Herman
- Blog post URL:

The code is prepared for the project:

Microsoft sees all the big companies creating original video content, and they want to get in on the fun. They have decided to create a new movie studio, but the problem is they don’t know anything about creating movies. They have hired you to help them better understand the movie industry. Your team is charged with doing data analysis and creating a presentation that explores what type of films are currently doing the best at the box office. You must then translate those findings into actionable insights that the CEO can use when deciding what type of films they should be creating.

Loading the data provided in zippedData from several mainstream moive-related sources, and get familiar about the information in each file

In [69]:

```
# Imported necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [70]:

```
# get the files in the zippedData folder
import os
def get_sorted_files(Directory):
    filenamelist = []
    for root, dirs, files in os.walk(Directory):
        for name in files:
            fullname = os.path.join(root, name)
            filenamelist.append(fullname)
    return sorted(filenamelist)

moiveFileList = get_sorted_files("zippedData")
print ("Number of items in the moiveFileList = ", len(moiveFileList))
ii = 0
for moivefile in moiveFileList:
    ii += 1
    print(str(ii) +':' + moivefile)
del ii
```

```
Number of items in the moiveFileList =  11
1:zippedData\bom.movie_gross.csv.gz
2:zippedData\imdb.name.basics.csv.gz
3:zippedData\imdb.title.akas.csv.gz
4:zippedData\imdb.title.basics.csv.gz
5:zippedData\imdb.title.crew.csv.gz
6:zippedData\imdb.title.principals.csv.gz
7:zippedData\imdb.title.ratings.csv.gz
8:zippedData\rt.movie_info.tsv.gz
9:zippedData\rt.reviews.tsv.gz
10:zippedData\tmdb.movies.csv.gz
11:zippedData\tn.movie_budgets.csv.gz
```

Summary of files in zippedData folder

There are 11 files in the zippedData folder and they are from five sources: bom, imdb, rt, tmdb, and tn. Among them, bom,tmdb and tn have only one single file, imdb have 6 files and rt have 2 files

In [71]:

```
# Loading data from the files in the folder zippedData
# First use the file names to set the variable names
moiveVar = []
for moiveFile in moiveFileList:
    moivefilesplit = moiveFile.split(".")
    moivevartmp = ""
    for ii in range(len(moivefilesplit)-2):
        if ii == 0:
            moivevartmp += moivevartmp + moivefilesplit[ii]
        else:
            moivevartmp += moivevartmp + moivefilesplit[ii].title()
    moiveVar.append(moivevartmp.split("\\")[-1])
# get the variable names and check whether it is consistent with file names
for ii in range(len(moiveFileList)):
    print(str(ii) + ':' + moiveVar[ii] + ' -- ' + moiveFileList[ii])
```

```
0:bomMovie_Gross -- zippedData\bom.movie_gross.csv.gz
1:imdbNameBasics -- zippedData\imdb.name.basics.csv.gz
2:imdbTitleAkas -- zippedData\imdb.title.akas.csv.gz
3:imdbTitleBasics -- zippedData\imdb.title.basics.csv.gz
4:imdbTitleCrew -- zippedData\imdb.title.crew.csv.gz
5:imdbTitlePrincipals -- zippedData\imdb.title.principals.csv.gz
6:imdbTitleRatings -- zippedData\imdb.title.ratings.csv.gz
7:rtMovie_Info -- zippedData\rt.movie_info.tsv.gz
8:rtReviews -- zippedData\rt.reviews.tsv.gz
9:tmdbMovies -- zippedData\tmdb.movies.csv.gz
10:tnMovie_Budgets -- zippedData\tn.movie_budgets.csv.gz
```

In [72]:

```
# load the data for each file
bomMovie_Gross = pd.read_csv(moiveFileList[0])
imdbNameBasics = pd.read_csv(moiveFileList[1])
imdbTitleAkas = pd.read_csv(moiveFileList[2])
imdbTitleBasics = pd.read_csv(moiveFileList[3])
imdbTitleCrew = pd.read_csv(moiveFileList[4])
imdbTitlePrincipals = pd.read_csv(moiveFileList[5])
imdbTitleRatings = pd.read_csv(moiveFileList[6])
rtMovie_Info = pd.read_csv(moiveFileList[7],encoding= 'unicode_escape',sep='\t')
rtReviews = pd.read_csv(moiveFileList[8], encoding= 'unicode_escape',sep='\t')
tmdbMovies = pd.read_csv(moiveFileList[9])
tnMovie_Budgets = pd.read_csv(moiveFileList[10])
```

In [73]:

```
# check the information for each one
# for bom: it has domestic_gross and foreign_gross data
bomMovie_Gross.info() # 3387 entries
bomMovie_Gross.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

Out[73]:

	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
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

In [74]:

```
# For imdb: it has averagerating in imdbTitleRatings
imdbNameBasics.info() #606648 entries
imdbNameBasics.head()
imdbTitleAkas.info() #606648 entries
imdbTitleAkas.head()
imdbTitleBasics.info()
imdbTitleBasics.head()
imdbTitleCrew.info() #606648 entries
imdbTitleCrew.head()
imdbTitlePrincipals.info() #606648 entries
imdbTitlePrincipals.head()
imdbTitleRatings.info() # 606648 entries, 8 columns
imdbTitleRatings.head()

# Get columns for five files
print('imdbNameBasics columns:',imdbNameBasics.columns)
print('imdbTitleAkas columns:',imdbTitleAkas.columns)
print('imdbTitleBasics columns:', imdbTitleBasics.columns)
print('imdbTitleCrew columns:',imdbTitleCrew.columns)
print('imdbTitlePrincipals columns:',imdbTitlePrincipals.columns)
print('imdbTitleRatings columns:',imdbTitleRatings.columns)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 606648 entries, 0 to 606647
Data columns (total 6 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   nconst              606648 non-null  object 
 1   primary_name        606648 non-null  object 
 2   birth_year          82736 non-null   float64
 3   death_year          6783 non-null    float64
 4   primary_profession  555308 non-null  object 
 5   known_for_titles    576444 non-null  object 
dtypes: float64(2), object(4)
memory usage: 27.8+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331703 entries, 0 to 331702
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   title_id           331703 non-null  object 
 1   ordering           331703 non-null  int64  
 2   title              331703 non-null  object 
 3   region             278410 non-null  object 
 4   language           41715 non-null   object 
 5   types              168447 non-null  object 
 6   attributes         14925 non-null   object 
 7   is_original_title  331678 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 20.2+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 3 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   tconst     146144 non-null  object
 1   directors  140417 non-null  object
 2   writers    110261 non-null  object
dtypes: object(3)
memory usage: 3.3+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1028186 entries, 0 to 1028185
Data columns (total 6 columns):
 #   Column      Non-Null Count    Dtype 
---  ------      --------------    ----- 
 0   tconst      1028186 non-null  object
 1   ordering    1028186 non-null  int64 
 2   nconst      1028186 non-null  object
 3   category    1028186 non-null  object
 4   job         177684 non-null   object
 5   characters  393360 non-null   object
dtypes: int64(1), object(5)
memory usage: 47.1+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73856 entries, 0 to 73855
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   tconst         73856 non-null  object 
 1   averagerating  73856 non-null  float64
 2   numvotes       73856 non-null  int64  
dtypes: float64(1), int64(1), object(1)
memory usage: 1.7+ MB
imdbNameBasics columns: Index(['nconst', 'primary_name', 'birth_year', 'deat
h_year',
       'primary_profession', 'known_for_titles'],
      dtype='object')
imdbTitleAkas columns: Index(['title_id', 'ordering', 'title', 'region', 'la
nguage', 'types',
       'attributes', 'is_original_title'],
      dtype='object')
imdbTitleBasics columns: Index(['tconst', 'primary_title', 'original_title',
'start_year',
       'runtime_minutes', 'genres'],
      dtype='object')
imdbTitleCrew columns: Index(['tconst', 'directors', 'writers'], dtype='obje
ct')
imdbTitlePrincipals columns: Index(['tconst', 'ordering', 'nconst', 'categor
y', 'job', 'characters'], dtype='object')
imdbTitleRatings columns: Index(['tconst', 'averagerating', 'numvotes'], dty
pe='object')
```

In [75]:

```
# For RT: in rtMovie_Info, it has box_office for not all movies
#          in rtReviews, it has rating information
rtMovie_Info.info() # 5782 entries, 6 columns
rtMovie_Info.head()
# rtReviews.info() # 5782 entries, 8 columns
# rtReviews.head()
# get columns for both files
# print('rtMovie_Info columns:', rtMovie_Info.columns)
# print('rtReviews columns:', rtReviews.columns)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1560 entries, 0 to 1559
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            1560 non-null   int64 
 1   synopsis      1498 non-null   object
 2   rating        1557 non-null   object
 3   genre         1552 non-null   object
 4   director      1361 non-null   object
 5   writer        1111 non-null   object
 6   theater_date  1201 non-null   object
 7   dvd_date      1201 non-null   object
 8   currency      340 non-null    object
 9   box_office    340 non-null    object
 10  runtime       1530 non-null   object
 11  studio        494 non-null    object
dtypes: int64(1), object(11)
memory usage: 146.4+ KB
```

Out[75]:

	id	synopsis	rating	genre	director	writer	theater_date
0	1	This gritty, fast-paced, and innovative police...	R	Action and Adventure|Classics|Drama	William Friedkin	Ernest Tidyman	Oct 9, 1971
1	3	New York City, not-too-distant-future: Eric Pa...	R	Drama|Science Fiction and Fantasy	David Cronenberg	David Cronenberg|Don DeLillo	Aug 17, 2012
2	5	Illeana Douglas delivers a superb performance ...	R	Drama|Musical and Performing Arts	Allison Anders	Allison Anders	Sep 13, 1996
3	6	Michael Douglas runs afoul of a treacherous su...	R	Drama|Mystery and Suspense	Barry Levinson	Paul Attanasio|Michael Crichton	Dec 9, 1994
4	7	NaN	NR	Drama|Romance	Rodney Bennett	Giles Cooper	NaN

In [76]:

```
# For tmdb, it does not have box information but has the vote information
tmdbMovies.info() # 26517 entries, 10 columns
tmdbMovies.head()
```

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

Out[76]:

	Unnamed: 0	genre_ids	id	original_language	original_title	popularity	release_date	t
0	0	[12, 14, 10751]	12444	en	Harry Potter and the Deathly Hallows: Part 1	33.533	2010-11-19	Ha Po and Deat Hallo Pa
1	1	[14, 12, 16, 10751]	10191	en	How to Train Your Dragon	28.734	2010-03-26	Hov Tr Y Drag
2	2	[12, 28, 878]	10138	en	Iron Man 2	28.515	2010-05-07	Iron M
3	3	[16, 35, 10751]	862	en	Toy Story	28.005	1995-11-22	St
4	4	[28, 878, 12]	27205	en	Inception	27.920	2010-07-16	Incept

In [77]:

```
# For rt, it has budget, domestic_gross and worldwide_gross
tnMovie_Budgets.info() # 5782 entries
tnMovie_Budgets.head()
```

```
<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
```

Out[77]:

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

Based on the information obtained from each file above, there are two key types of information, i.e., gross/box data and voting/rating data, will be used to identify which categories will be suggested to be the focus of new studio in Microsoft. gross/box data can be obtained from tnMoive,rtMovie_Info, or bomMovie_Gross voting/rating data can be obtained from tmdbMovies, imdbTitleRatings, or rtReviews

Three questions:

1) What's the general performance of movie industry in terms of box office for recent years?

2) What types of films have large number of box office?

3) What types of films have good ratings?

Question 1: What's the general performance of movie industry in terms of box office for recent years?

I am going to analyze the data from bomMovie_Gross and tnMovie_Budgets.

For bomMoive_Gross, I am going to plot the domestic and foreign gross for each year to check the trend for the moive industry

In [78]:

```
# take a look the summary of each column
bomMovie_Gross.info()
bomMovie_Gross.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   title           3387 non-null   object 
 1   studio          3382 non-null   object 
 2   domestic_gross  3359 non-null   float64
 3   foreign_gross   2037 non-null   object 
 4   year            3387 non-null   int64  
dtypes: float64(1), int64(1), object(3)
memory usage: 132.4+ KB
```

Out[78]:

	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
4	Shrek Forever After	P/DW	238700000.0	513900000	2010

From the summary above, I observed:

1) There are 3387 total entries

2) domestic_gross only have 3359 entires, which means there are 3387-3359 NAN

3) foreign_gross only have 2037 entries, which means there are 3387-2037 NAN

4) foreign_gross is not in the type of float64 and need be converted to float64

In [79]:

```
# deal with foreign_gross
# convert string to float, however, it has errors since there are some values with ',',
# therefore, firstly remove the ',' in the corresponding values
for ii in range(len(bomMovie_Gross.foreign_gross)):
    strval = bomMovie_Gross.foreign_gross[ii]
    if not isinstance(strval,float):
        #print(str(ii) + ':' + strval)
        if len((strval)) > 0 :
            if "," in strval:
                print(str(ii) + ':' + strval)
                bomMovie_Gross.foreign_gross[ii] = strval.replace(",","")
```

```
1872:1,131.6
1873:1,019.4
1874:1,163.0
2760:1,010.0
3079:1,369.5

<ipython-input-79-ca3113c4e3af>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  bomMovie_Gross.foreign_gross[ii] = strval.replace(",","")
```

In [80]:

```
# convert string to float and assign it for a new column
bomMovie_Gross['foreign_grossfloat'] = bomMovie_Gross.foreign_gross.map(lambda x: float(x))
bomMovie_Gross.info()
# add one more column for total gross
bomMovie_Gross['total_gross'] = bomMovie_Gross.foreign_grossfloat + bomMovie_Gross.domestic
bomMovie_Gross.info()
# replace nan as 0 for each item
bomMovie_Gross['domestic_gross0'] = bomMovie_Gross['domestic_gross'].fillna(0)
bomMovie_Gross['foreign_grossfloat0'] = bomMovie_Gross['foreign_grossfloat'].fillna(0)
bomMovie_Gross['total_gross0'] = bomMovie_Gross['total_gross'].fillna(0)
bomMovie_Gross.info()
bomMovie_Gross['year'].unique()
# for sum of each year:2010-2018 (obtained from the unique)
bomMovie_Gross_yearsum = bomMovie_Gross.groupby(['year']).sum()
bomMovie_Gross_yearsum.info()
bomMovie_Gross_yearsum.head()
# it seems it is unnecessary to fillna, since the sum is same before and after
# for mean of each year, since every year the number of movies will be different
bomMovie_Gross_yearmean = bomMovie_Gross.groupby(['year']).mean()
print("*************** bomMovie_gross_yearmean *********")
bomMovie_Gross_yearmean.info()
bomMovie_Gross_yearmean.head()
# for mean, the values are different before and after replacing nan to zero, I will use the
```

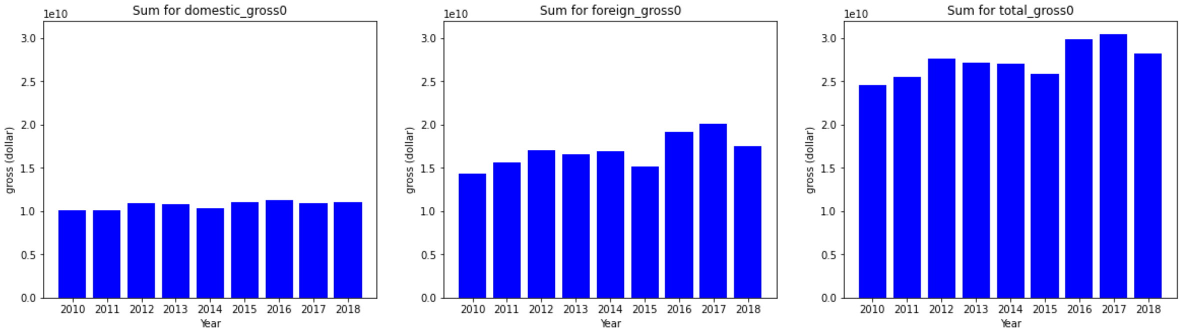
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   title               3387 non-null   object 
 1   studio              3382 non-null   object 
 2   domestic_gross      3359 non-null   float64
 3   foreign_gross       2037 non-null   object 
 4   year                3387 non-null   int64  
 5   foreign_grossfloat  2037 non-null   float64
dtypes: float64(2), int64(1), object(3)
memory usage: 158.9+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 7 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  
 5   foreign_grossfloat  2037 non-null   float64
 6   total_gross         2009 non-null   float64
dtypes: float64(3), int64(1), object(3)
memory usage: 185.4+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3387 entries, 0 to 3386
Data columns (total 10 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  
 5   foreign_grossfloat   2037 non-null   float64
 6   total_gross          2009 non-null   float64
 7   domestic_gross0      3387 non-null   float64
 8   foreign_grossfloat0  3387 non-null   float64
 9   total_gross0         3387 non-null   float64
dtypes: float64(6), int64(1), object(3)
memory usage: 264.7+ KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9 entries, 2010 to 2018
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   domestic_gross       9 non-null      float64
 1   foreign_grossfloat   9 non-null      float64
 2   total_gross          9 non-null      float64
 3   domestic_gross0      9 non-null      float64
 4   foreign_grossfloat0  9 non-null      float64
 5   total_gross0         9 non-null      float64
dtypes: float64(6)
memory usage: 504.0 bytes
*************** bomMovie_gross_yearmean *********
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9 entries, 2010 to 2018
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   domestic_gross       9 non-null      float64
 1   foreign_grossfloat   9 non-null      float64
 2   total_gross          9 non-null      float64
 3   domestic_gross0      9 non-null      float64
 4   foreign_grossfloat0  9 non-null      float64
 5   total_gross0         9 non-null      float64
dtypes: float64(6)
memory usage: 504.0 bytes
```

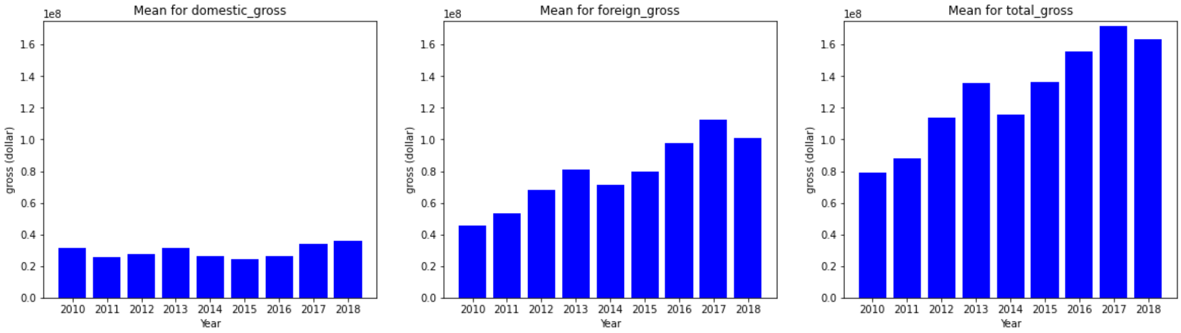
Out[80]:

	domestic_gross	**foreign_grossfloat**	**total_gross**	**domestic_gross0**	**foreign_grossfloat0**
year					
2010	3.144559e+07	4.577789e+07	7.937058e+07	3.096624e+07	4.382396e+07
2011	2.535052e+07	5.348459e+07	8.791040e+07	2.522345e+07	3.927565e+07
2012	2.767584e+07	6.815155e+07	1.139953e+08	2.719151e+07	4.259472e+07
2013	3.128212e+07	8.103607e+07	1.356955e+08	3.083523e+07	4.746398e+07
2014	2.643923e+07	7.131079e+07	1.158318e+08	2.617149e+07	4.296701e+07

In [81]:

```
# barplot figure for the sum of three different types of gross, using the values before or
fig, (ax1, ax2, ax3)  = plt.subplots(1, 3, figsize = (20, 5))
ax1.bar(bomMovie_Gross_yearsum.index,bomMovie_Gross_yearsum['domestic_gross0'], color='blue
ax1.set_xlabel('Year')
ax1.set_ylabel('gross (dollar)')
ax1.set_title('Sum for domestic_gross0')
ax1.set_xticks(bomMovie_Gross_yearsum.index)
ax1.set_ylim([0,3.2e10])
ax2.bar(bomMovie_Gross_yearsum.index,bomMovie_Gross_yearsum['foreign_grossfloat0'], color='
ax2.set_xlabel('Year')
ax2.set_ylabel('gross (dollar)')
ax2.set_title('Sum for foreign_gross0')
ax2.set_xticks(bomMovie_Gross_yearsum.index)
ax2.set_ylim([0,3.2e10])
ax3.bar(bomMovie_Gross_yearsum.index,bomMovie_Gross_yearsum['total_gross0'], color='blue')
ax3.set_xlabel('Year')
ax3.set_ylabel('gross (dollar)')
ax3.set_title('Sum for total_gross0')
ax3.set_xticks(bomMovie_Gross_yearsum.index)
ax3.set_ylim([0,3.2e10])
plt.show()

# barplot figure for the mean of three different types of gross, will use the values before
fig, (ax1, ax2, ax3)  = plt.subplots(1, 3, figsize = (20, 5))
ax1.bar(bomMovie_Gross_yearmean.index,bomMovie_Gross_yearmean['domestic_gross'], color='blu
ax1.set_xlabel('Year')
ax1.set_ylabel('gross (dollar)')
ax1.set_title('Mean for domestic_gross')
ax1.set_xticks(bomMovie_Gross_yearmean.index)
ax1.set_ylim([0,1.75e8])
ax2.bar(bomMovie_Gross_yearmean.index,bomMovie_Gross_yearmean['foreign_grossfloat'], color=
ax2.set_xlabel('Year')
ax2.set_ylabel('gross (dollar)')
ax2.set_title('Mean for foreign_gross')
ax2.set_xticks(bomMovie_Gross_yearmean.index)
ax2.set_ylim([0,1.75e8])
ax3.bar(bomMovie_Gross_yearmean.index,bomMovie_Gross_yearmean['total_gross'], color='blue')
ax3.set_xlabel('Year')
ax3.set_ylabel('gross (dollar)')
ax3.set_title('Mean for total_gross')
ax3.set_xticks(bomMovie_Gross_yearmean.index)
ax3.set_ylim([0,1.75e8])

plt.show()
```





Observations from figures of the sum and mean values:

1) From figures of sum values: a. the domestic gross is quite stable, while foreign gross has an increased trend b. the foreign gross has around 1.5 times as compared to domestic gross, thus the trend in total gross is similar as foreign gross

2) From figures of mean values: a. there are increased trend in both domestic and foreign gross, while the slope is much high in foreign gross b. again the mean values for each moive are around 2 times higher in foreign gross than in domestic gross, thus the trend in total gross is similar as foreign gross

In [82]:

```
# look the data from tnMovie_Budgets, which has the unique information regarding budgets,
# which is also important if profit rather than gross will be examined
tnMovie_Budgets.info()
tnMovie_Budgets.head()
```

```
<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
```

Out[82]:

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

1) tnMovie_Budgets have all the values for each entry but need to convert the values from string to float for production_budget, domestic_gross, and worldwide_gross

2) need to extract year from release_date if I want to examine year values

In [83]:

```
# convert string to float and assign it for a new column, again it need to remove ',' and '
# find a new way to do it
tnMovie_Budgets['production_budgetpure'] = tnMovie_Budgets['production_budget'].str.replace
tnMovie_Budgets['domestic_grosspure'] = tnMovie_Budgets['domestic_gross'].str.replace(r'\D'
tnMovie_Budgets['worldwide_grosspure'] = tnMovie_Budgets['worldwide_gross'].str.replace(r'\
tnMovie_Budgets.head()

# get year from release_data
tnMovie_Budgets['release_year'] = pd.DatetimeIndex(tnMovie_Budgets['release_date']).year
tnMovie_Budgets.head()
tnMovie_Budgets['release_year'].unique()

# calculate domestic profit and worldwide profit and also the rate
tnMovie_Budgets['domestic_profit'] = tnMovie_Budgets['domestic_grosspure'] - tnMovie_Budget
tnMovie_Budgets['worldwide_profit'] = tnMovie_Budgets['worldwide_grosspure'] - tnMovie_Budg
tnMovie_Budgets['domestic_profit_perc'] = tnMovie_Budgets['domestic_profit'] / tnMovie_Budg
tnMovie_Budgets['worldwide_profit_perc'] = tnMovie_Budgets['worldwide_profit'] / tnMovie_Bu

# calculate sum and mean for year, of course, the movies released early will be watched mor
tnMovie_Budgets_sum = tnMovie_Budgets.groupby(['release_year']).sum()
tnMovie_Budgets_sum.info()
tnMovie_Budgets_sum.head()
tnMovie_Budgets_mean = tnMovie_Budgets.groupby(['release_year']).mean()
tnMovie_Budgets_mean.info()
tnMovie_Budgets_mean.tail(10)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 1915 to 2020
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     96 non-null     int64  
 1   production_budgetpure  96 non-null     float64
 2   domestic_grosspure     96 non-null     float64
 3   worldwide_grosspure    96 non-null     float64
 4   domestic_profit        96 non-null     float64
 5   worldwide_profit       96 non-null     float64
 6   domestic_profit_perc   96 non-null     float64
 7   worldwide_profit_perc  96 non-null     float64
dtypes: float64(7), int64(1)
memory usage: 6.8 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 1915 to 2020
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     96 non-null     float64
 1   production_budgetpure  96 non-null     float64
 2   domestic_grosspure     96 non-null     float64
 3   worldwide_grosspure    96 non-null     float64
 4   domestic_profit        96 non-null     float64
 5   worldwide_profit       96 non-null     float64
 6   domestic_profit_perc   96 non-null     float64
 7   worldwide_profit_perc  96 non-null     float64
dtypes: float64(8)
memory usage: 6.8 KB
```

Out[83]:

	id	production_budgetpure	domestic_grosspure	worldwide_grosspure	don
release_year					
2011	52.141732	3.524784e+07	3.856479e+07	9.714671e+07	3
2012	51.336170	3.611547e+07	4.479053e+07	1.156729e+08	8
2013	49.701681	3.739238e+07	4.482367e+07	1.107553e+08	7
2014	49.188235	3.040552e+07	3.945958e+07	9.911344e+07	9
2015	51.260355	2.616029e+07	3.193948e+07	8.225145e+07	5
2016	49.643836	4.097370e+07	5.042387e+07	1.313614e+08	9
2017	51.422619	5.003073e+07	6.222259e+07	1.692240e+08	1
2018	53.286713	4.813886e+07	7.378870e+07	1.824786e+08	2
2019	51.791045	5.273896e+07	4.280029e+07	9.965411e+07	-9
2020	45.666667	9.400000e+07	0.000000e+00	0.000000e+00	-9

In [84]:

```
tnMovie_Budgets_sum.index[-11:-1]
```

Out[84]:

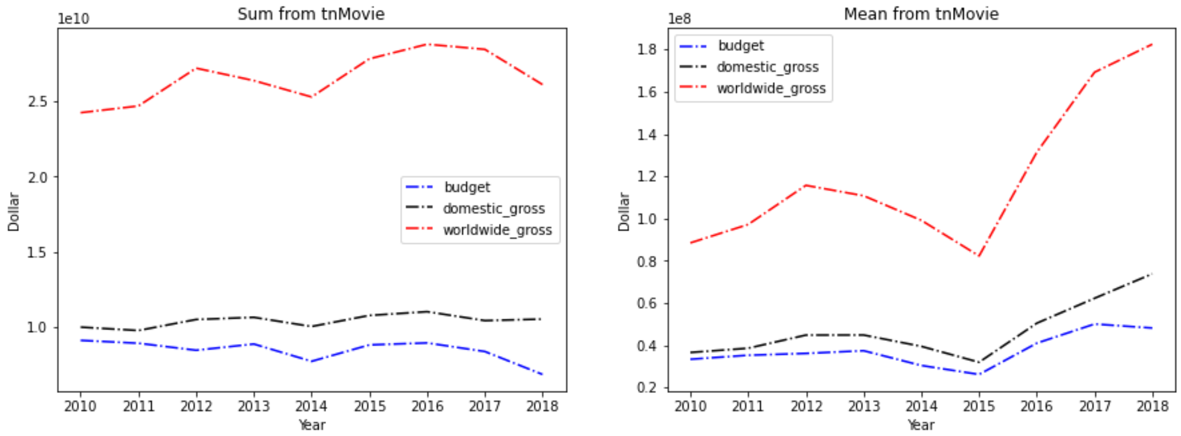
```
Int64Index([2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019], dty
pe='int64', name='release_year')
```

In [85]:

```
# To be consistent with the data from bomMovie_Gross, I only plot the data from 2010 to 201
# 1) curve plot of sum and mean for budget, domestic_gross and worldwide_gross
fig, (ax1, ax2)  = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['production_budgetpure'][-11
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['domestic_grosspure'][-11:-2
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['worldwide_grosspure'][-11:-
ax1.legend()
ax1.set_xlabel('Year')
ax1.set_ylabel('Dollar')
ax1.set_title('Sum from tnMovie')
ax1.set_xticks(tnMovie_Budgets_sum.index[-11:-2])

ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['production_budgetpure'][-
ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['domestic_grosspure'][-11:
ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['worldwide_grosspure'][-11
ax2.legend()
ax2.set_xlabel('Year')
ax2.set_ylabel('Dollar')
ax2.set_title('Mean from tnMovie')
ax2.set_xticks(tnMovie_Budgets_sum.index[-11:-2])
```

Out[85]:

```
[<matplotlib.axis.XTick at 0x20204ee60d0>,
 <matplotlib.axis.XTick at 0x20204ee60a0>,
 <matplotlib.axis.XTick at 0x20204eb3be0>,
 <matplotlib.axis.XTick at 0x20204f376a0>,
 <matplotlib.axis.XTick at 0x20204f37bb0>,
 <matplotlib.axis.XTick at 0x20204f40100>,
 <matplotlib.axis.XTick at 0x20204f372e0>,
 <matplotlib.axis.XTick at 0x20204f402b0>,
 <matplotlib.axis.XTick at 0x20204f407c0>]
```



From sum values:

1) total budget is quite stable, but drops a lot in 2018 and 2019 2) domestic gross is also stable, but also drops a lot in 2018 and 2019 3) worldwide gross has the same observations as domestic gross 4) domestic gross always have larger values than budget, and worldwide gross is around 2.5 times than budget and domestic gross

From mean values:

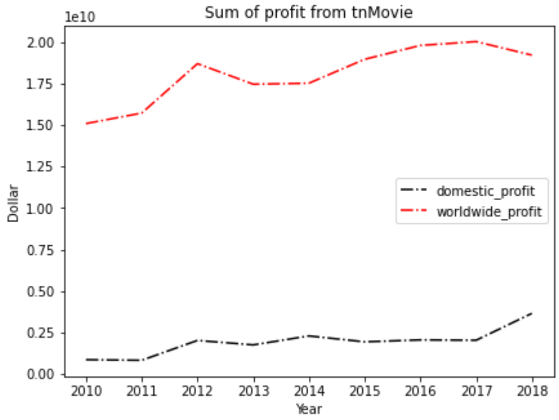
1) all three have a increase and decrease pattern in two intervals: from 2010 to 2015, and 2015 to 2019

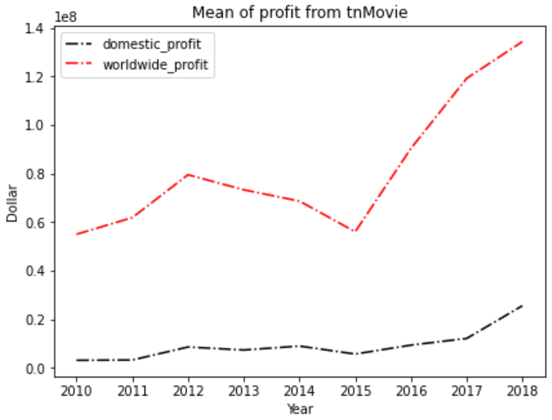
In [86]:

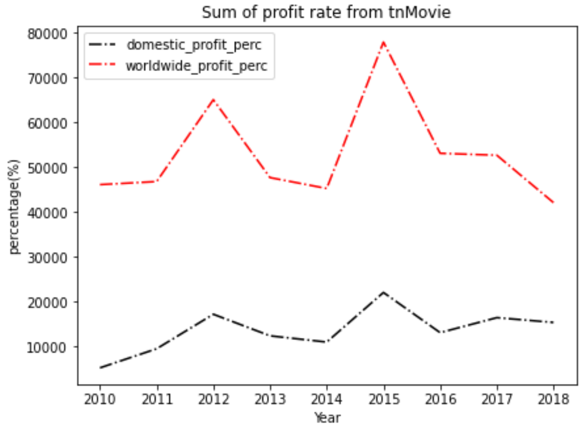
```
# 2) curve plot of sum and mean for profit and profit rate
fig, (ax1, ax2)  = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['domestic_profit'][-11:-2],
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['worldwide_profit'][-11:-2],
ax1.legend()
ax1.set_xlabel('Year')
ax1.set_ylabel('Dollar')
ax1.set_title('Sum of profit from tnMovie')
ax1.set_xticks(tnMovie_Budgets_sum.index[-11:-2])

ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['domestic_profit'][-11:-2]
ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['worldwide_profit'][-11:-2
ax2.legend()
ax2.set_xlabel('Year')
ax2.set_ylabel('Dollar')
ax2.set_title('Mean of profit from tnMovie')
ax2.set_xticks(tnMovie_Budgets_sum.index[-11:-2])

fig, (ax1, ax2)  = plt.subplots(1, 2, figsize = (15, 5))
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['domestic_profit_perc'][-11:
ax1.plot(tnMovie_Budgets_sum.index[-11:-2],tnMovie_Budgets_sum['worldwide_profit_perc'][-11
ax1.legend()
ax1.set_xlabel('Year')
ax1.set_ylabel('percentage(%)')
ax1.set_title('Sum of profit rate from tnMovie')
ax1.set_xticks(tnMovie_Budgets_sum.index[-11:-2])

ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['domestic_profit_perc'][-1
ax2.plot(tnMovie_Budgets_mean.index[-11:-2],tnMovie_Budgets_mean['worldwide_profit_perc'][-
ax2.legend()
ax2.set_xlabel('Year')
ax2.set_ylabel('percentage(%)')
ax2.set_title('Mean of profit rate from tnMovie')
ax2.set_xticks(tnMovie_Budgets_sum.index[-11:-2])
```

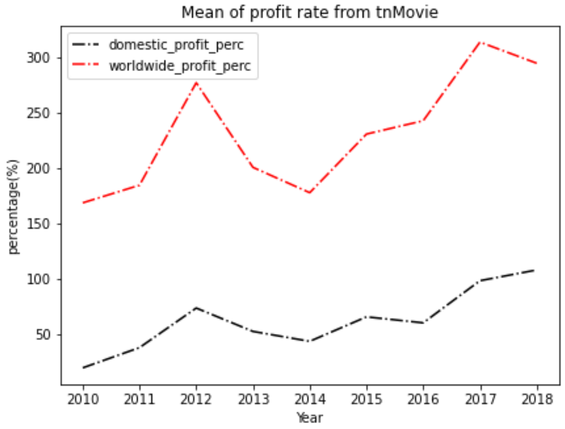
Out[86]:

```
[<matplotlib.axis.XTick at 0x202056faca0>,
 <matplotlib.axis.XTick at 0x202056fac70>,
 <matplotlib.axis.XTick at 0x202056fa3d0>,
 <matplotlib.axis.XTick at 0x20205743850>,
 <matplotlib.axis.XTick at 0x20205743d60>,
 <matplotlib.axis.XTick at 0x2020574b2b0>,
 <matplotlib.axis.XTick at 0x20205743a90>,
 <matplotlib.axis.XTick at 0x2020573c880>,
 <matplotlib.axis.XTick at 0x2020574ba90>]
```









From profit and profit rate figures, I only focused on the mean ones.

1) From mean profit, the domestic profit has an increased trend till 2018 and 2019, while worldwide profit has an increase and decrease pattern in two intervals: 2010-2015 and 2015-2019. Worldwide profit is much higher than domestic profit 2) From mean profit rate, both have an increase and decrease pattern in two intervals: 2010-2014 and 2014-2019. Worldwide profit rate is much higher than domestic profit rate

For 2018 and 2019, lots of data probably still missing

End of Question 1: What's the general performance of movie industry in terms of box office for recent years?

Answer: the movie industy is still developing very well, and lots of profits could be achieved, especially from worldwide

---

Question 2: What types of films have large number of box office?

imdbTitleBasics and tnMovie_Budgets will be examined together since the first one has the information for movies categories, and the last one has the information for box office and budget

In [87]:

```
# refresh for imdb related files
# imdbNameBasics.info() #606648 entries
# imdbNameBasics.head()
# imdbTitleAkas.info() #606648 entries
# imdbTitleAkas.head()
imdbTitleBasics.info()
imdbTitleBasics.head()
# imdbTitleCrew.info() #606648 entries
# imdbTitleCrew.head()
# imdbTitlePrincipals.info() #606648 entries
# imdbTitlePrincipals.head()
# imdbTitleRatings.info() # 606648 entries, 8 columns
# imdbTitleRatings.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

Out[87]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

From imdbTitleBasics, it has the genres columns indicating the categories of the movie and one movie might have more than one types. In addtion, there are also 146144-140736 movies have Null in genres. Therefore, I will only get the movies with the genres information

In [88]:

```
imdbTitleBasicsWithGenres = imdbTitleBasics[pd.isna(imdbTitleBasics['genres']) == False]
imdbTitleBasicsWithGenres.info()
imdbTitleBasicsWithGenres.head()
imdbTitleBasicsWithGenres['start_year'].unique()
# only select movies released between 2010 and 2019
imdbTitleBasicsWithGenres = imdbTitleBasicsWithGenres[(imdbTitleBasicsWithGenres['start_yea
imdbTitleBasicsWithGenres.info()
imdbTitleBasicsWithGenres.head()
imdbTitleBasicsWithGenres['start_year'].unique()
imdbTitleBasicsWithGenres.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 140736 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           140736 non-null  object 
 1   primary_title    140736 non-null  object 
 2   original_title   140734 non-null  object 
 3   start_year       140736 non-null  int64  
 4   runtime_minutes  112233 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.5+ MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 139722 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           139722 non-null  object 
 1   primary_title    139722 non-null  object 
 2   original_title   139720 non-null  object 
 3   start_year       139722 non-null  int64  
 4   runtime_minutes  112145 non-null  float64
 5   genres           139722 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 7.5+ MB
```

Out[88]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

In [89]:

```
# combining gross data from tnMoive_Budgets and genres information from imdbTitleBasicsWith
# To do so,I will use SQL in pandas
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())

# select primary_title, original_title, start_year, and genres from imdbTitleBasicsWithGenr
# and production_budgetpure, domestic_grosspure, worldwide_grosspure, and their profit and

q1 = """SELECT
        imdb.genres, imdb.start_year, tnm.movie,
        tnm.production_budgetpure, tnm.domestic_grosspure, tnm.worldwide_grosspure,
        tnm.domestic_profit, tnm.worldwide_profit, tnm.domestic_profit_perc, tnm.worldwide_
     FROM
        imdbTitleBasicsWithGenres imdb
     INNER JOIN
        tnMovie_Budgets tnm
           ON (imdb.primary_title = tnm.movie or imdb.original_title = tnm.movie )
           and imdb.start_year = tnm.release_year
     ORDER bY worldwide_profit desc;"""
imdbTitleGenrestnMovieJoin = pysqldf(q1)
```

In [90]:

```
imdbTitleGenrestnMovieJoin.info()
imdbTitleGenrestnMovieJoin.head(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1571 entries, 0 to 1570
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   genres                 1571 non-null   object 
 1   start_year             1571 non-null   int64  
 2   movie                  1571 non-null   object 
 3   production_budgetpure  1571 non-null   float64
 4   domestic_grosspure     1571 non-null   float64
 5   worldwide_grosspure    1571 non-null   float64
 6   domestic_profit        1571 non-null   float64
 7   worldwide_profit       1571 non-null   float64
 8   domestic_profit_perc   1571 non-null   float64
 9   worldwide_profit_perc  1571 non-null   float64
dtypes: float64(7), int64(1), object(2)
memory usage: 122.9+ KB
```

Out[90]:

	genres	start_year	movie	production_budgetpure	domestic_gr
0	Action,Adventure,Sci-Fi	2018	Avengers: Infinity War	300000000.0	6788
1	Action,Adventure,Sci-Fi	2015	Jurassic World	215000000.0	6522
2	Action,Crime,Thriller	2015	Furious 7	190000000.0	3530
3	Action,Adventure,Sci-Fi	2012	The Avengers	225000000.0	6232
4	Action,Adventure,Sci-Fi	2018	Black Panther	200000000.0	7000
5	Action,Adventure,Sci-Fi	2018	Jurassic World: Fallen Kingdom	170000000.0	4177
6	Adventure,Animation,Comedy	2013	Frozen	150000000.0	4007
7	Family,Fantasy,Musical	2017	Beauty and the Beast	160000000.0	5040
8	Adventure,Animation,Comedy	2015	Minions	74000000.0	3360
9	Action,Adventure,Sci-Fi	2015	Avengers: Age of Ultron	330600000.0	4590
10	Action,Adventure,Animation	2018	Incredibles 2	200000000.0	6085
11	Action,Adventure,Sci-Fi	2013	Iron Man 3	200000000.0	4089
12	Action,Adventure,Fantasy	2018	Aquaman	160000000.0	3350
13	Action,Crime,Thriller	2017	The Fate of the Furious	250000000.0	2257
14	Adventure,Animation,Comedy	2017	Despicable Me 3	75000000.0	2646
15	Action,Adventure,Sci-Fi	2019	Captain Marvel	175000000.0	4265

	genres	start_year	movie	production_budgetpure	domestic_gr
16	Action,Adventure,Sci-Fi	2011	Transformers: Dark of the Moon	195000000.0	352
17	Action,Adventure,Thriller	2012	Skyfall	200000000.0	304
18	Adventure,Animation,Comedy	2013	Despicable Me 2	76000000.0	368
19	Action,Adventure,Sci-Fi	2014	Transformers: Age of Extinction	210000000.0	245

I found there are repeated movies, e.g., Frozen in 2010 and 2013, in the first code, therefore, double check the original data

In [91]:

```
# tnMovie_Budgets.head()
tnMovie_BudgetsFrozen = tnMovie_Budgets[tnMovie_Budgets['movie'] == 'Frozen']
tnMovie_BudgetsFrozen.info()
tnMovie_BudgetsFrozen.head()
# imdbTitleBasicsWithGenres.head()
# imdbTitleBasicsWithGenresFrozen = imdbTitleBasicsWithGenres[imdbTitleBasicsWithGenres['pr
# imdbTitleBasicsWithGenresFrozen.info()
# imdbTitleBasicsWithGenresFrozen.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1 entries, 155 to 155
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     1 non-null      int64  
 1   release_date           1 non-null      object 
 2   movie                  1 non-null      object 
 3   production_budget      1 non-null      object 
 4   domestic_gross         1 non-null      object 
 5   worldwide_gross        1 non-null      object 
 6   production_budgetpure  1 non-null      float64
 7   domestic_grosspure     1 non-null      float64
 8   worldwide_grosspure    1 non-null      float64
 9   release_year           1 non-null      int64  
 10  domestic_profit        1 non-null      float64
 11  worldwide_profit       1 non-null      float64
 12  domestic_profit_perc   1 non-null      float64
 13  worldwide_profit_perc  1 non-null      float64
dtypes: float64(7), int64(2), object(5)
memory usage: 120.0+ bytes
```

Out[91]:

	id	release_date	movie	production_budget	domestic_gross	worldwide_gross	productior
155	56	Nov 22, 2013	Frozen	$150,000,000	$400,738,009	$1,272,469,910	

I changed the query code and now the results are correct, and I have 1571 movies between 2010 to 2019 to check which categories of movies have more profit in each year, and how many of them released in each year

In [92]:

```
imdbTitleGenrestnMovieJoin.info()
imdbTitleGenrestnMovieJoin.head(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1571 entries, 0 to 1570
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   genres                 1571 non-null   object 
 1   start_year             1571 non-null   int64  
 2   movie                  1571 non-null   object 
 3   production_budgetpure  1571 non-null   float64
 4   domestic_grosspure     1571 non-null   float64
 5   worldwide_grosspure    1571 non-null   float64
 6   domestic_profit        1571 non-null   float64
 7   worldwide_profit       1571 non-null   float64
 8   domestic_profit_perc   1571 non-null   float64
 9   worldwide_profit_perc  1571 non-null   float64
dtypes: float64(7), int64(1), object(2)
memory usage: 122.9+ KB
```

Out[92]:

	genres	start_year	movie	production_budgetpure	domestic_gros
0	Action,Adventure,Sci-Fi	2018	Avengers: Infinity War	300000000.0	678815
1	Action,Adventure,Sci-Fi	2015	Jurassic World	215000000.0	652270
2	Action,Crime,Thriller	2015	Furious 7	190000000.0	353007
3	Action,Adventure,Sci-Fi	2012	The Avengers	225000000.0	623279
4	Action,Adventure,Sci-Fi	2018	Black Panther	200000000.0	700059
5	Action,Adventure,Sci-Fi	2018	Jurassic World: Fallen Kingdom	170000000.0	417719
6	Adventure,Animation,Comedy	2013	Frozen	150000000.0	400738
7	Family,Fantasy,Musical	2017	Beauty and the Beast	160000000.0	504014
8	Adventure,Animation,Comedy	2015	Minions	74000000.0	336045
9	Action,Adventure,Sci-Fi	2015	Avengers: Age of Ultron	330600000.0	459005
10	Action,Adventure,Animation	2018	Incredibles 2	200000000.0	608581
11	Action,Adventure,Sci-Fi	2013	Iron Man 3	200000000.0	408992
12	Action,Adventure,Fantasy	2018	Aquaman	160000000.0	335061
13	Action,Crime,Thriller	2017	The Fate of the Furious	250000000.0	225764
14	Adventure,Animation,Comedy	2017	Despicable Me 3	75000000.0	264624
15	Action,Adventure,Sci-Fi	2019	Captain Marvel	175000000.0	426525

	genres	start_year	movie	production_budgetpure	domestic_gros
16	Action,Adventure,Sci-Fi	2011	Transformers: Dark of the Moon	195000000.0	352390
17	Action,Adventure,Thriller	2012	Skyfall	200000000.0	304360
18	Adventure,Animation,Comedy	2013	Despicable Me 2	76000000.0	368065
19	Action,Adventure,Sci-Fi	2014	Transformers: Age of Extinction	210000000.0	245439

In [93]:

```
# since many movies have more than one types of genres, I need to split it
df = imdbTitleGenrestnMovieJoin
df.head()
df['genres'] = df['genres'].str.split(',')
df = df.explode('genres').reset_index(drop=True)
cols = list(df.columns)
cols.append(cols.pop(cols.index('movie')))
df = df[cols]
imdbTitleGenrestnMovieJoinInd = df
del df
imdbTitleGenrestnMovieJoinInd.info()
imdbTitleGenrestnMovieJoinInd.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3951 entries, 0 to 3950
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   genres                 3951 non-null   object 
 1   start_year             3951 non-null   int64  
 2   production_budgetpure  3951 non-null   float64
 3   domestic_grosspure     3951 non-null   float64
 4   worldwide_grosspure    3951 non-null   float64
 5   domestic_profit        3951 non-null   float64
 6   worldwide_profit       3951 non-null   float64
 7   domestic_profit_perc   3951 non-null   float64
 8   worldwide_profit_perc  3951 non-null   float64
 9   movie                  3951 non-null   object 
dtypes: float64(7), int64(1), object(2)
memory usage: 308.8+ KB
```

Out[93]:

	genres	start_year	production_budgetpure	domestic_grosspure	worldwide_grosspure	dor
0	Action	2018	300000000.0	678815482.0	2.048134e+09	
1	Adventure	2018	300000000.0	678815482.0	2.048134e+09	
2	Sci-Fi	2018	300000000.0	678815482.0	2.048134e+09	
3	Action	2015	215000000.0	652270625.0	1.648855e+09	
4	Adventure	2015	215000000.0	652270625.0	1.648855e+09	

In [94]:

```
imdbTitleGenrestnMovieJoinInd.info()
imdbTitleGenrestnMovieJoinInd.tail(20)
# I noticed some movies have negative worldwide_profit, therefore,
# I may need to remove the ones that has zero value in worldwide_grosspure in the future
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3951 entries, 0 to 3950
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   genres                 3951 non-null   object 
 1   start_year             3951 non-null   int64  
 2   production_budgetpure  3951 non-null   float64
 3   domestic_grosspure     3951 non-null   float64
 4   worldwide_grosspure    3951 non-null   float64
 5   domestic_profit        3951 non-null   float64
 6   worldwide_profit       3951 non-null   float64
 7   domestic_profit_perc   3951 non-null   float64
 8   worldwide_profit_perc  3951 non-null   float64
 9   movie                  3951 non-null   object 
dtypes: float64(7), int64(1), object(2)
memory usage: 308.8+ KB
```

Out[94]:

	genres	start_year	production_budgetpure	domestic_grosspure	worldwide_grosspure
3931	Family	2010	90000000.0	195459.0	20466016.0
3932	Fantasy	2010	90000000.0	195459.0	20466016.0
3933	Drama	2017	90000000.0	8224288.0	10551417.0
3934	Comedy	2017	90000000.0	8224288.0	10551417.0
3935	Drama	2017	90000000.0	8224288.0	10551417.0
3936	Horror	2017	90000000.0	8224288.0	10551417.0
3937	Thriller	2017	90000000.0	8224288.0	10551417.0
3938	Drama	2017	90000000.0	8224288.0	10551417.0
3939	Action	2017	90000000.0	0.0	0.0
3940	Crime	2017	90000000.0	0.0	0.0
3941	Fantasy	2017	90000000.0	0.0	0.0
3942	Action	2019	110000000.0	3100000.0	3100000.0
3943	Adventure	2019	110000000.0	3100000.0	3100000.0

	genres	start_year	production_budgetpure	domestic_grosspure	worldwide_grosspure
3944	Comedy	2019	110000000.0	3100000.0	3100000.0
3945	Adventure	2011	150000000.0	21392758.0	39549758.0
3946	Animation	2011	150000000.0	21392758.0	39549758.0
3947	Family	2011	150000000.0	21392758.0	39549758.0
3948	Action	2019	350000000.0	42762350.0	149762350.0
3949	Adventure	2019	350000000.0	42762350.0	149762350.0
3950	Sci-Fi	2019	350000000.0	42762350.0	149762350.0

Do some summary and plots:

1) sum or mean of each category for each year

2) total number of movies for each category and each year

In [95]:

```
genress = imdbTitleGenrestnMovieJoinInd['genres'].unique()
years = imdbTitleGenrestnMovieJoinInd['start_year'].unique()
# years = sort(years)
years.sort()
print(years)
# imdbgenresyear = imdbTitleGenrestnMovieJoinInd[(imdbTitleGenrestnMovieJoinInd['genres'] =
# imdbgenresyear.info()
# len(imdbgenresyear)
```

```
[2010 2011 2012 2013 2014 2015 2016 2017 2018 2019]
```

In [96]:

```
from statistics import mean
listgenres = []
listyear = []
listnomovies = []
listwwprofitsum = []
listwwprofitmean = []

for genres in genress:
    for year in years:
#         print('genres = ' + genres + ', year = ' + str(year))
        listgenres.append(genres)
        listyear.append(year)
        imdbgenresyear = imdbTitleGenrestnMovieJoinInd[(imdbTitleGenrestnMovieJoinInd['genr
#         print(imdbgenresyear.head())
        imdbgenreswwprofit = imdbgenresyear['worldwide_profit']
#         print(imdbgenreswwprofit)
        listnomovies.append(len(imdbgenresyear))
        listwwprofitsum.append(sum(imdbgenreswwprofit))
        if len(imdbgenresyear) > 1:
            listwwprofitmean.append(mean(imdbgenreswwprofit))
        else:
            listwwprofitmean.append(0)
        del imdbgenresyear,imdbgenreswwprofit
```

In [97]:

```
# listyear
data = {'genres': listgenres,
        'year': listyear,
        'nomovies': listnomovies,
        'worldwideprofitsum': listwwprofitsum,
        'worldwideprofitmean':listwwprofitmean}
genresyearsum = pd.DataFrame(data)
del data
genresyearsum.info()
genresyearsum.head(20)
# len(genresyearsum.genres.unique())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220 entries, 0 to 219
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   genres               220 non-null    object 
 1   year                 220 non-null    int64  
 2   nomovies             220 non-null    int64  
 3   worldwideprofitsum   220 non-null    float64
 4   worldwideprofitmean  220 non-null    float64
dtypes: float64(2), int64(2), object(1)
memory usage: 8.7+ KB
```

Out[97]:

	genres	year	nomovies	worldwideprofitsum	worldwideprofitmean
0	Action	2010	48	4.418963e+09	9.206174e+07
1	Action	2011	50	6.044774e+09	1.208955e+08
2	Action	2012	45	7.392425e+09	1.642761e+08
3	Action	2013	47	6.269770e+09	1.333994e+08
4	Action	2014	56	1.103460e+10	1.970464e+08
5	Action	2015	58	7.258649e+09	1.251491e+08
6	Action	2016	57	8.061890e+09	1.414367e+08
7	Action	2017	37	8.890198e+09	2.402756e+08
8	Action	2018	45	1.092058e+10	2.426795e+08
9	Action	2019	15	1.478470e+09	9.856469e+07
10	Adventure	2010	36	7.663667e+09	2.128796e+08
11	Adventure	2011	37	6.137608e+09	1.658813e+08
12	Adventure	2012	33	9.254049e+09	2.804257e+08
13	Adventure	2013	42	9.522152e+09	2.267179e+08
14	Adventure	2014	40	1.071673e+10	2.679183e+08
15	Adventure	2015	40	8.885882e+09	2.221471e+08
16	Adventure	2016	45	1.191502e+10	2.647781e+08
17	Adventure	2017	32	8.856685e+09	2.767714e+08
18	Adventure	2018	38	1.060465e+10	2.790698e+08

	genres	year	nomovies	worldwideprofitsum	worldwideprofitmean
19	Adventure	2019	15	2.221993e+09	1.481329e+08

Now I have get the data for worldwideprofit for different types of movies between 2010 to 2019. In the following, I will plot them using bar plot for nomovies, worldwideprofitsum,worldwideprofitmean

In [98]:

```
# there are 22 categories in genres, 10 years(2010-2019), three metrics: nomovies, worldwid
# for each metric, I will plot 22 categories into 2 plots, with x axis as year
# convert pandas from long to wide for the sake of subsequent ploting
genresyearsumtmp = genresyearsum.pivot_table(index=['year'],columns='genres',aggfunc='mean'
genresyearsumtmp.columns = genresyearsumtmp.columns.map(lambda x: '{}_{}'.format(x[0], x[1]
genresyearsumtmp = genresyearsumtmp.reset_index()
genresyearsumtmp.head(10)
```

Out[98]:

	year	nomovies_Action	nomovies_Adventure	nomovies_Animation	nomovies_Biography	no
0	2010	48	36	11	9	
1	2011	50	37	14	10	
2	2012	45	33	11	6	
3	2013	47	42	12	15	
4	2014	56	40	11	19	
5	2015	58	40	9	20	
6	2016	57	45	14	23	
7	2017	37	32	12	15	
8	2018	45	38	8	17	
9	2019	15	15	4	5	

10 rows × 67 columns

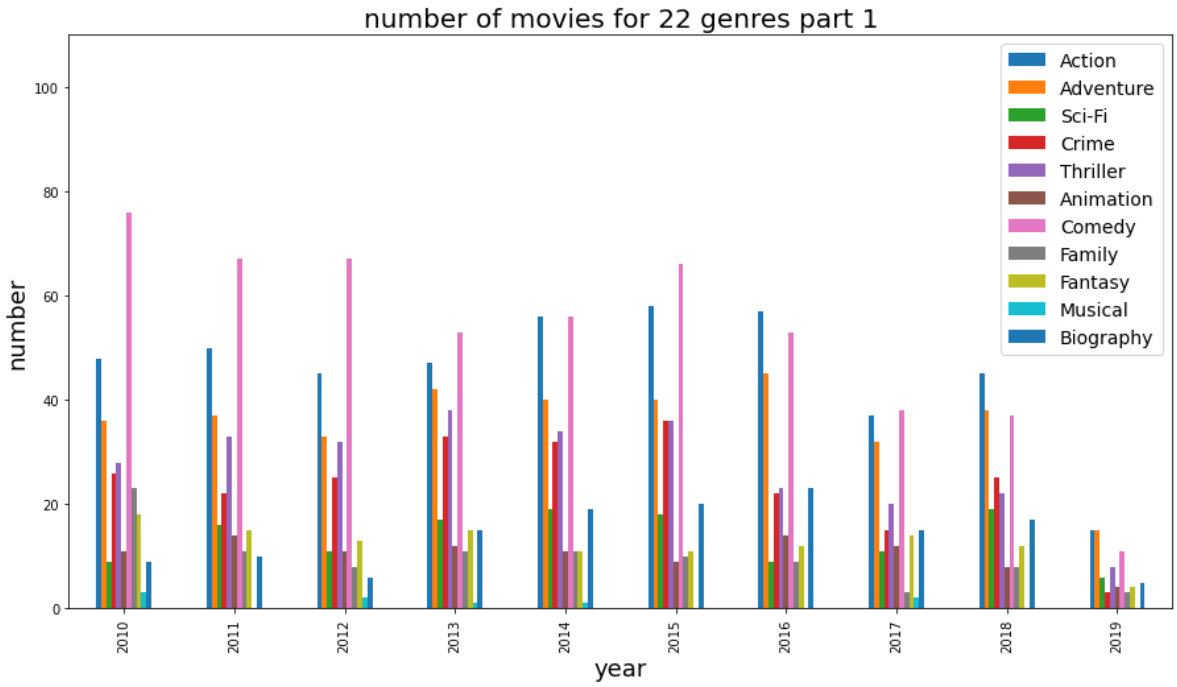
In [99]:

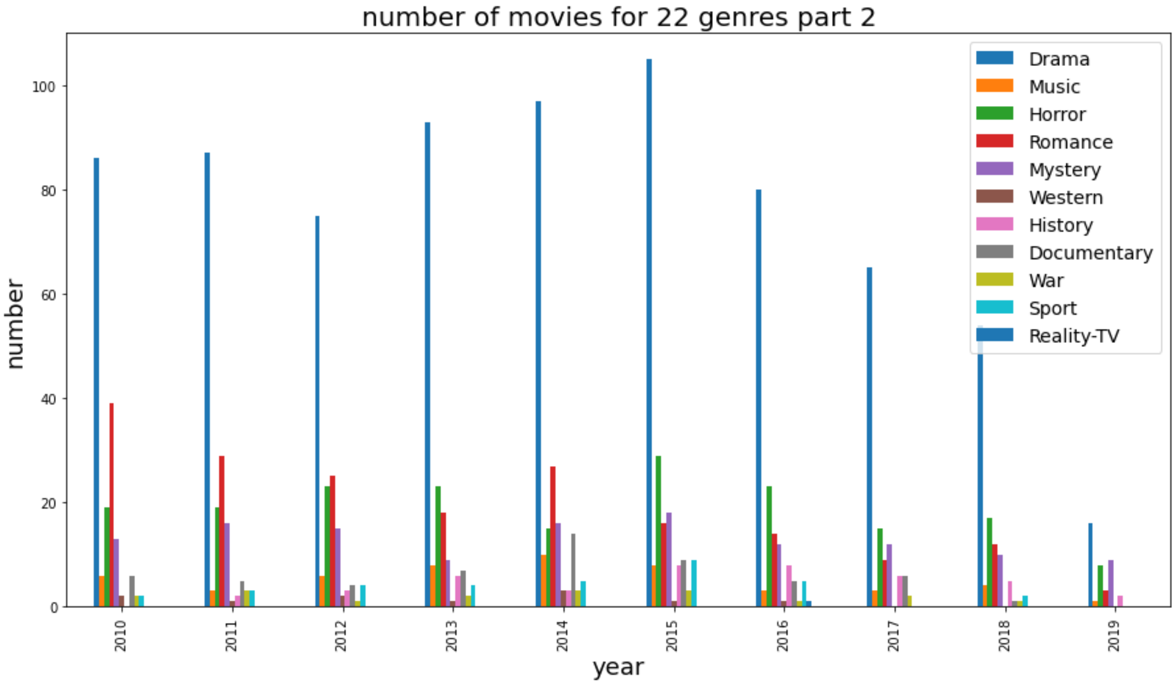
```
genrestmp  = genresyearsum['genres'].unique()
print(genrestmp)
# 1) nomoives
nomoviescols = []
for ii in range(11):
    nomoviescols.append("nomovies_"+genrestmp[ii])
ax = genresyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestmp[0:11],fontsize=14)
ax.set_title('number of movies for 22 genres part 1',fontsize=20)
nomoviescols = []
for ii in range(11):
    nomoviescols.append("nomovies_"+genrestmp[ii+11])
ax = genresyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestmp[11:22],fontsize=14)
ax.set_title('number of movies for 22 genres part 2',fontsize=20)
```

```
['Action' 'Adventure' 'Sci-Fi' 'Crime' 'Thriller' 'Animation' 'Comedy'
 'Family' 'Fantasy' 'Musical' 'Biography' 'Drama' 'Music' 'Horror'
 'Romance' 'Mystery' 'Western' 'History' 'Documentary' 'War' 'Sport'
 'Reality-TV']
```

Out[99]:

```
Text(0.5, 1.0, 'number of movies for 22 genres part 2')
```



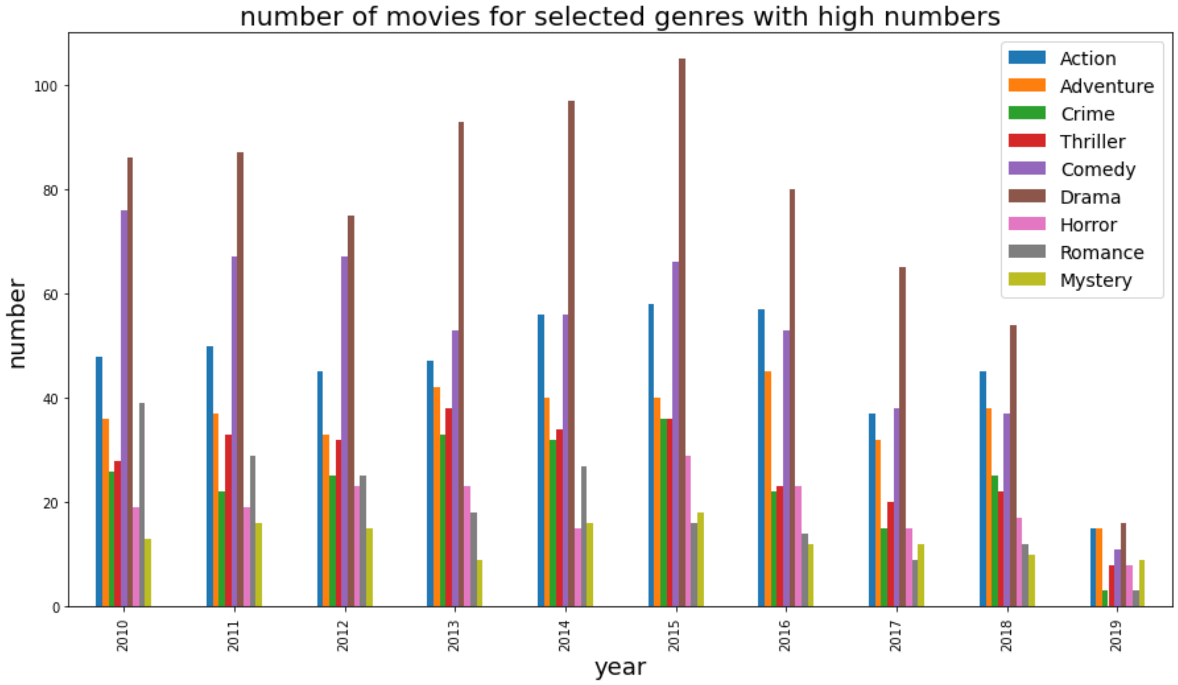

number of movies for 22 genres part 2
number
year
0
20
40
60
80
100
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
Drama
Music
Horror
Romance
Mystery
Western
History
Documentary
War
Sport
Reality-TV

In [100]:

```
# From above figures for number of movies released each year for different categories,
# 'Action','Adventure','Crime','Thriller','Comedy', 'Drama', 'Horror', 'Romance',and 'Myste
# have more released, therefore, plot these categories in a single figure
genrestsel = [ 'Action','Adventure','Crime','Thriller','Comedy', 'Drama', 'Horror', 'Romanc
nomoviescols = []
for ii in range(len(genrestsel)):
    nomoviescols.append("nomovies_"+genrestsel[ii])
ax = genresyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestsel,fontsize=14)
ax.set_title('number of movies for selected genres with high numbers',fontsize=20)
```

Out[100]:

```
Text(0.5, 1.0, 'number of movies for selected genres with high numbers')
```



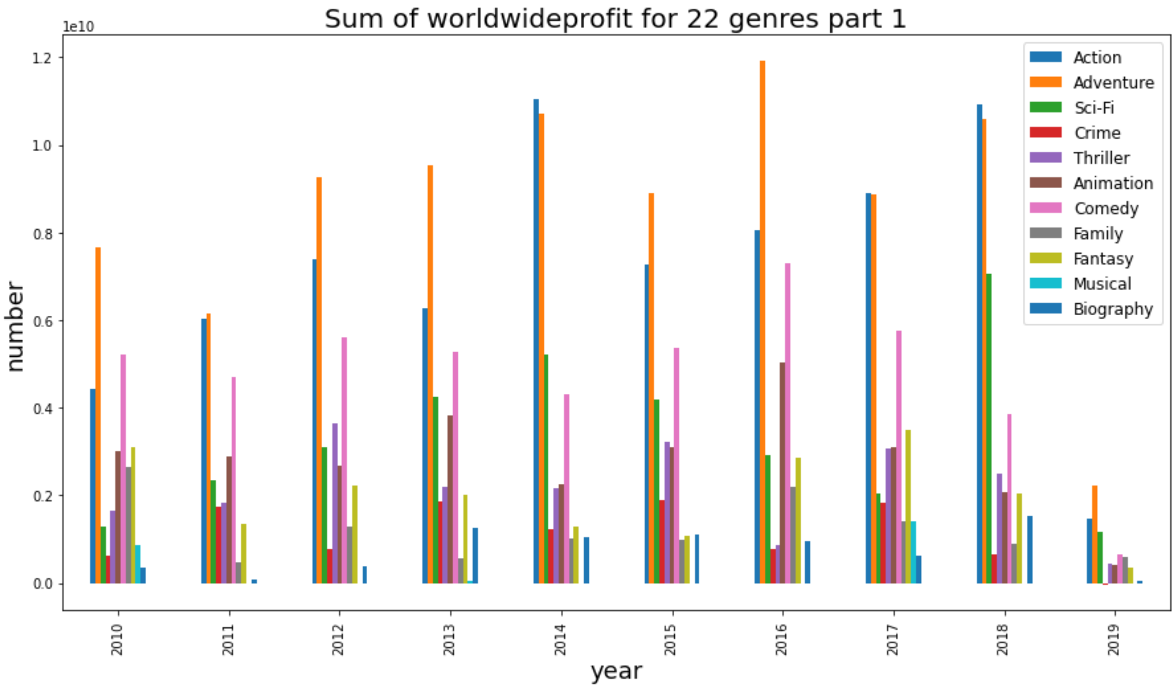
From the above figure, Drama, Comedy, Action,Adventure are the top four categories in terms of number of movies released each year between 2010 and 2019

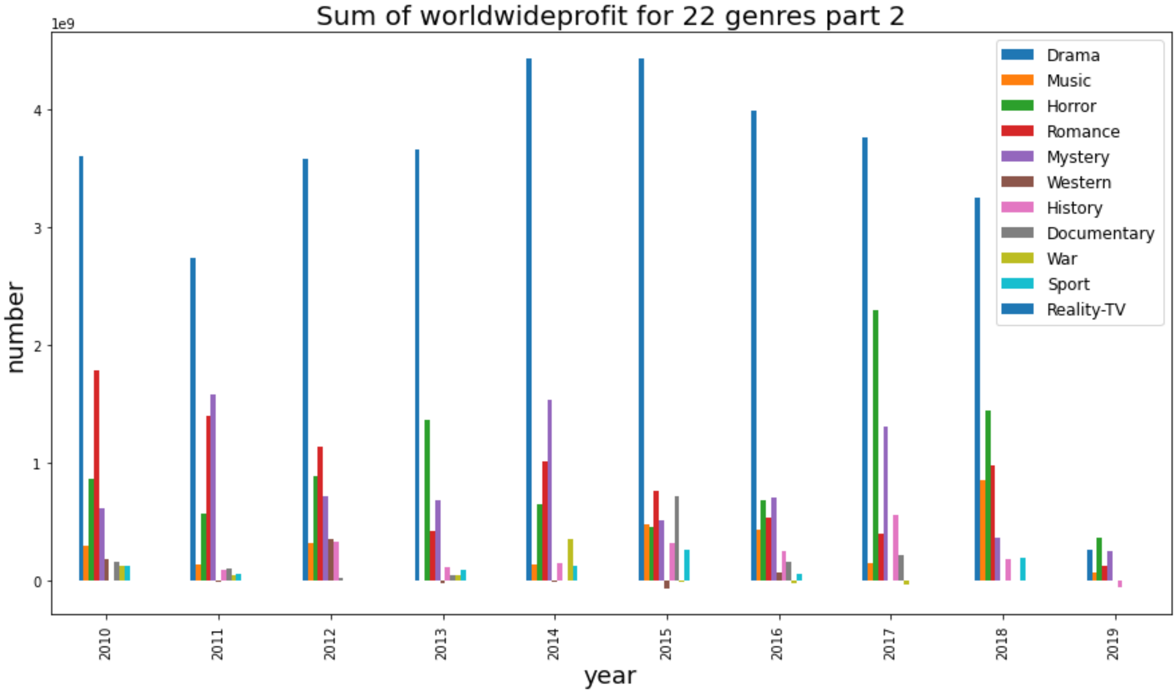
In [101]:

```
# 2) worldwideprofitsum
worldwideprofitsumcols = []
for ii in range(11):
    worldwideprofitsumcols.append("worldwideprofitsum_"+genrestmp[ii])
ax = genresyearsumtmp.plot( x = "year", y = worldwideprofitsumcols, kind="bar",figsize=(15,
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
# ax.set_ylim([0,110])
ax.legend(genrestmp[0:11],fontsize=12)
ax.set_title('Sum of worldwideprofit for 22 genres part 1',fontsize=20)

worldwideprofitsumcols = []
for ii in range(11):
    worldwideprofitsumcols.append("worldwideprofitsum_"+genrestmp[ii+11])
ax = genresyearsumtmp.plot( x = "year", y =worldwideprofitsumcols, kind="bar",figsize=(15,8
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
# ax.set_ylim([0,110])
ax.legend(genrestmp[11:22],fontsize=12)
ax.set_title('Sum of worldwideprofit for 22 genres part 2',fontsize=20)
```

Out[101]:

```
Text(0.5, 1.0, 'Sum of worldwideprofit for 22 genres part 2')
```



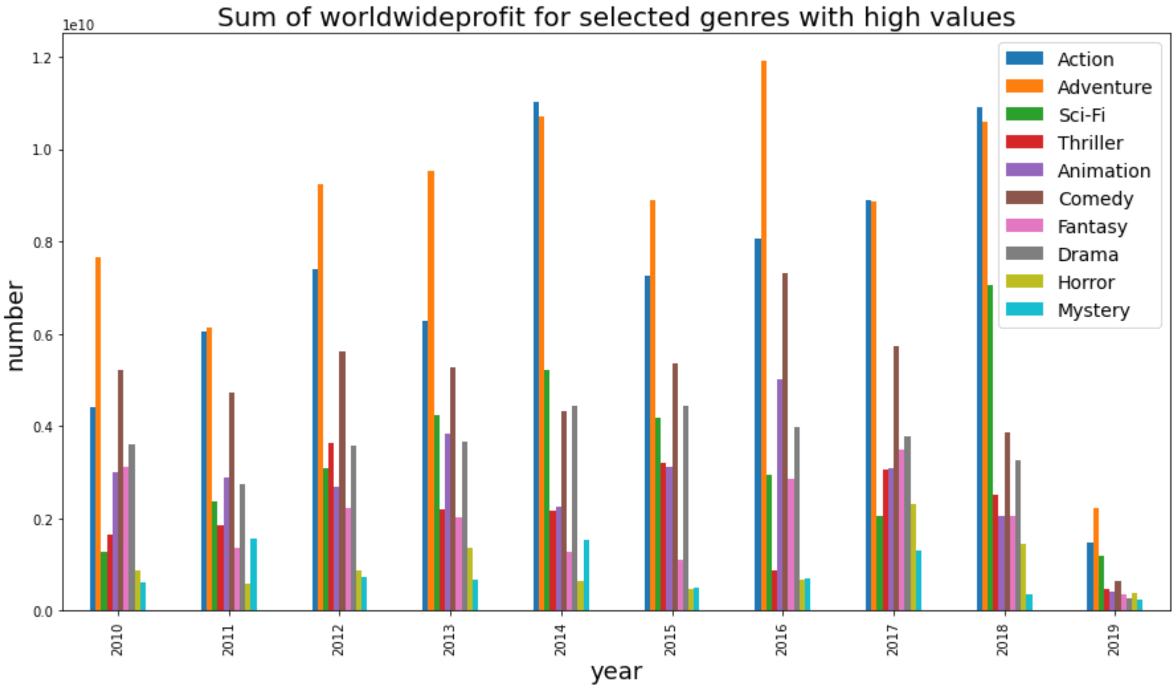

Sum of worldwideprofit for 22 genres part 2
1e9
number
4
3
2
1
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
year
Drama
Music
Horror
Romance
Mystery
Western
History
Documentary
War
Sport
Reality-TV

In [102]:

```
# From above figures,
#  'Action','Adventure','Sci-Fi','Thriller','Animation','Comedy', 'Fantasy','Drama', 'Horro
# have high profit, therefore, plot these categories in a single figure
genrestsel = [ 'Action','Adventure','Sci-Fi','Thriller','Animation','Comedy', 'Fantasy','Dr
worldwideprofitsumcols = []
for ii in range(len(genrestsel)):
    worldwideprofitsumcols.append("worldwideprofitsum_"+genrestsel[ii])
ax = genresyearsumtmp.plot( x = "year", y = worldwideprofitsumcols, kind="bar",figsize=(15,
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
# ax.set_ylim([0,110])
ax.legend(genrestsel,fontsize=14)
ax.set_title('Sum of worldwideprofit for selected genres with high values',fontsize=20)
```

Out[102]:

```
Text(0.5, 1.0, 'Sum of worldwideprofit for selected genres with high value
s')
```



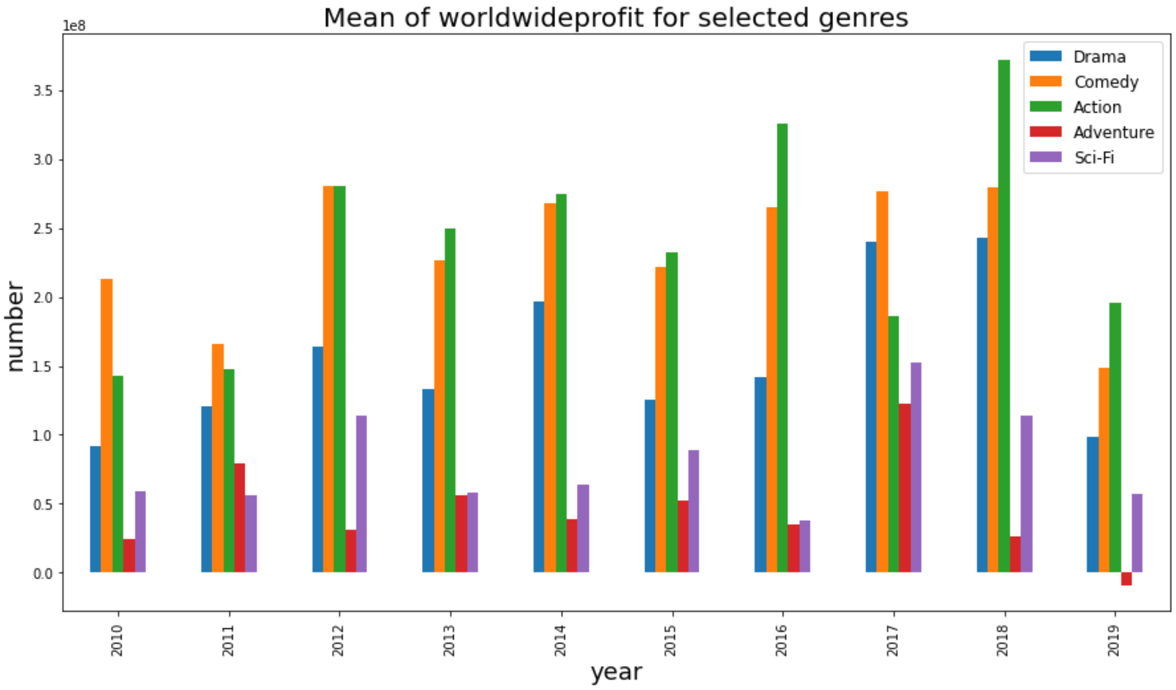
From above figure, Adventure, Action, Comedy, and Sci-Fi, are the top 4 categories in terms of sum of worldwideprofit

In [103]:

```
# 3) worldwideprofitmean: only examine values for Drama, Comedy, Action,Adventure,and Sci-F
# based on the conclusions from nomovies and worldwideprofitsum
genrestsel = ['Drama', 'Comedy', 'Action', 'Adventure', 'Sci-Fi']
worldwideprofitmeancols = []
for ii in range(len(genrestsel)):
    worldwideprofitmeancols.append("worldwideprofitmean_"+genrestmp[ii])
ax = genresyearsumtmp.plot( x = "year", y = worldwideprofitmeancols, kind="bar",figsize=(15
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
# ax.set_ylim([0,110])
ax.legend(genrestsel,fontsize=12)
ax.set_title('Mean of worldwideprofit for selected genres',fontsize=20)
```

Out[103]:

```
Text(0.5, 1.0, 'Mean of worldwideprofit for selected genres')
```



From mean of worldwide profit, action, comedy and Drama have higher mean values of worldwide profit

End of Question 2: What types of films have large number of box office?

Answer: Action, comedy and Drama are the categories of movies to be invested in terms of profit

Quesion 3: What types of films have good ratings?

Originally, I plan to use imdbTitleBasics and imdbTitleRatings to investigate this question. However, since I already prepared imdbTitleGenrestnMovieJoinInd above, I will combine this one with imdbTitleRatings to answer this question

In [104]:

```
# refresh for imdb related files
imdbTitleBasics.info()
imdbTitleBasics.head()
# imdbTitleRatings.info()
# imdbTitleRatings.head()
# imdbTitleGenrestnMovieJoinInd.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 146144 entries, 0 to 146143
Data columns (total 6 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   tconst           146144 non-null  object 
 1   primary_title    146144 non-null  object 
 2   original_title   146123 non-null  object 
 3   start_year       146144 non-null  int64  
 4   runtime_minutes  114405 non-null  float64
 5   genres           140736 non-null  object 
dtypes: float64(1), int64(1), object(4)
memory usage: 6.7+ MB
```

Out[104]:

	tconst	primary_title	original_title	start_year	runtime_minutes	genres
0	tt0063540	Sunghursh	Sunghursh	2013	175.0	Action,Crime,Drama
1	tt0066787	One Day Before the Rainy Season	Ashad Ka Ek Din	2019	114.0	Biography,Drama
2	tt0069049	The Other Side of the Wind	The Other Side of the Wind	2018	122.0	Drama
3	tt0069204	Sabse Bada Sukh	Sabse Bada Sukh	2018	NaN	Comedy,Drama
4	tt0100275	The Wandering Soap Opera	La Telenovela Errante	2017	80.0	Comedy,Drama,Fantasy

In [105]:

```
# it seems I need to get tconst to combine three dataframes
q2 = """SELECT
        imdbtn.genres, imdbtn.start_year, imdbtn.worldwide_profit, imdbtn.movie,
        imdbt.tconst,imdbtR.averagerating
     FROM
        imdbTitleGenrestnMovieJoinInd imdbtn
     JOIN
        imdbTitleBasics imdbt
           ON imdbt.primary_title = imdbtn.movie
           and imdbt.start_year = imdbtn.start_year
     INNER JOIN
        imdbTitleRatings imdbtR
           USING(tconst)
     ORDER bY averagerating desc;"""
imdbTitleGenrestnMovieJoinIndRating = pysqldf(q2)
```

In [106]:

```
imdbTitleGenrestnMovieJoinIndRating.info()
imdbTitleGenrestnMovieJoinIndRating.head(20)
# imdbTitleGenrestnMovieJoinIndtconst.tail(20)
# imdbTitleGenrestnMovieJoinInd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4014 entries, 0 to 4013
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   genres            4014 non-null   object 
 1   start_year        4014 non-null   int64  
 2   worldwide_profit  4014 non-null   float64
 3   movie             4014 non-null   object 
 4   tconst            4014 non-null   object 
 5   averagerating     4014 non-null   float64
dtypes: float64(2), int64(1), object(3)
memory usage: 188.3+ KB
```

Out[106]:

	genres	start_year	worldwide_profit	movie	tconst	averagerating
0	Action	2010	6.755246e+08	Inception	tt1375666	8.8
1	Adventure	2010	6.755246e+08	Inception	tt1375666	8.8
2	Sci-Fi	2010	6.755246e+08	Inception	tt1375666	8.8
3	Drama	2017	-7.944858e+07	The Promise	tt4192918	8.8
4	Comedy	2017	-7.944858e+07	The Promise	tt4192918	8.8
5	Drama	2017	-7.944858e+07	The Promise	tt4192918	8.8
6	Horror	2017	-7.944858e+07	The Promise	tt4192918	8.8
7	Thriller	2017	-7.944858e+07	The Promise	tt4192918	8.8
8	Drama	2017	-7.944858e+07	The Promise	tt4192918	8.8
9	Adventure	2014	5.013794e+08	Interstellar	tt0816692	8.6
10	Drama	2014	5.013794e+08	Interstellar	tt0816692	8.6
11	Sci-Fi	2014	5.013794e+08	Interstellar	tt0816692	8.6
12	Action	2018	1.748134e+09	Avengers: Infinity War	tt4154756	8.5
13	Adventure	2018	1.748134e+09	Avengers: Infinity War	tt4154756	8.5
14	Sci-Fi	2018	1.748134e+09	Avengers: Infinity War	tt4154756	8.5
15	Action	2016	2.851546e+08	Dangal	tt5074352	8.5
16	Biography	2016	2.851546e+08	Dangal	tt5074352	8.5
17	Drama	2016	2.851546e+08	Dangal	tt5074352	8.5
18	Drama	2014	3.566904e+07	Whiplash	tt2582802	8.5
19	Music	2014	3.566904e+07	Whiplash	tt2582802	8.5

In [107]:

```
# I noticed the rows increased, therefore, I checked and found some movies have more than o
# e.g., The Promise. Here, I will not further clean the data since I don't know which tcons
# but I will keep it in mind
imdbTitleBasics[imdbTitleBasics['primary_title'] == 'The Promise'].head(10)
# imdbTitleGenrestnMovieJoinInd.info()
# imdbTitleGenrestnMovieJoinInd.tail(20)
```

Out[107]:

	tconst	primary_title	original_title	start_year	runtime_minutes	ger
645	tt10013288	The Promise	The Promise	2017	NaN	Dra
73582	tt4192918	The Promise	The Promise	2017	NaN	Com
79008	tt4532628	The Promise	Ghoul	2014	75.0	N
82799	tt4776998	The Promise	The Promise	2016	133.0	Drama,His
96138	tt5609150	The Promise	The Promise	2016	74.0	Documen
103635	tt6072400	The Promise	The Promise	2016	90.0	Thr
114193	tt6745888	The Promise	The Promise	2018	NaN	Ac
115160	tt6825024	The Promise	The Promise	2019	62.0	Comedy,Drama,Roma
120201	tt7232438	The Promise	Puen Tee Raluek	2017	114.0	Drama,Horror,Thr
145937	tt9889072	The Promise	The Promise	2017	NaN	Dra

In [108]:

```
# Now I am going to do some similar operations and plots as for the worldwide profit
# 1) calculate mean rating for each category per year
listgenres = []
listyear = []
listnomovies = []
listratingmean = []
genress = imdbTitleGenrestnMovieJoinIndRating['genres'].unique()
years = imdbTitleGenrestnMovieJoinIndRating['start_year'].unique()
# years = sort(years)
years.sort()

for genres in genress:
    for year in years:
#         print('genres = ' + genres + ', year = ' + str(year))
        listgenres.append(genres)
        listyear.append(year)
        imdbgenresyear = imdbTitleGenrestnMovieJoinIndRating[(imdbTitleGenrestnMovieJoinInd
                                                                & (imdbTitleGenrestnMovieJoinInd
#         print(imdbgenresyear.head())
        imdbgenresrating = imdbgenresyear['averagerating']
#         print(imdbgenreswwprofit)
        listnomovies.append(len(imdbgenresyear))
        if len(imdbgenresyear) > 1:
            listratingmean.append(mean(imdbgenresrating))
        else:
            listratingmean.append(0)
        del imdbgenresyear,imdbgenresrating
data = {'genres': listgenres,
        'year': listyear,
        'nomovies': listnomovies,
        'ratingmean':listratingmean}
genresrateyearsum = pd.DataFrame(data)
del data
genresrateyearsum.info()
genresrateyearsum.head(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220 entries, 0 to 219
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   genres      220 non-null    object 
 1   year        220 non-null    int64  
 2   nomovies    220 non-null    int64  
 3   ratingmean  220 non-null    float64
dtypes: float64(1), int64(2), object(1)
memory usage: 7.0+ KB
```

Out[108]:

	genres	year	nomovies	ratingmean
0	Action	2010	48	6.141667
1	Action	2011	52	6.223077
2	Action	2012	45	6.286667

	genres	year	nomovies	ratingmean
3	Action	2013	49	6.357143
4	Action	2014	56	6.328571
5	Action	2015	61	5.852459
6	Action	2016	57	6.291228
7	Action	2017	37	6.570270
8	Action	2018	44	6.386364
9	Action	2019	10	6.530000
10	Adventure	2010	35	6.425714
11	Adventure	2011	37	6.302703
12	Adventure	2012	33	6.754545
13	Adventure	2013	41	6.424390
14	Adventure	2014	42	6.645238
15	Adventure	2015	40	6.265000
16	Adventure	2016	44	6.570455
17	Adventure	2017	33	6.342424
18	Adventure	2018	39	6.310256
19	Adventure	2019	13	6.492308

In [109]:

```
# # 2) convert long into wide dataframe
genresrateyearsumtmp = genresrateyearsum.pivot_table(index=['year'],columns='genres',aggfun
genresrateyearsumtmp.columns = genresrateyearsumtmp.columns.map(lambda x: '{}_{}'.format(x[
genresrateyearsumtmp = genresrateyearsumtmp.reset_index()
genresrateyearsumtmp.head(20)
```

Out[109]:

	year	nomovies_Action	nomovies_Adventure	nomovies_Animation	nomovies_Biography	no
0	2010	48	35	8	9	
1	2011	52	37	14	10	
2	2012	45	33	11	6	
3	2013	49	41	12	14	
4	2014	56	42	11	18	
5	2015	61	40	9	20	
6	2016	57	44	13	24	
7	2017	37	33	13	15	
8	2018	44	39	8	17	
9	2019	10	13	4	5	

10 rows × 45 columns

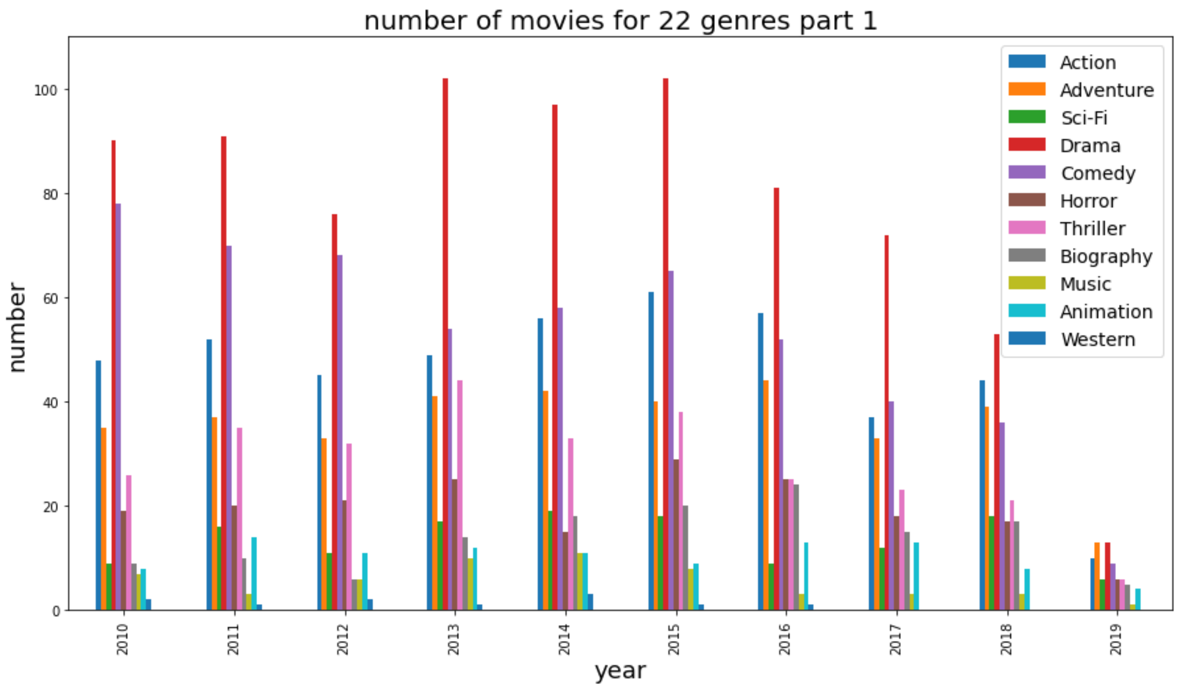
In [110]:

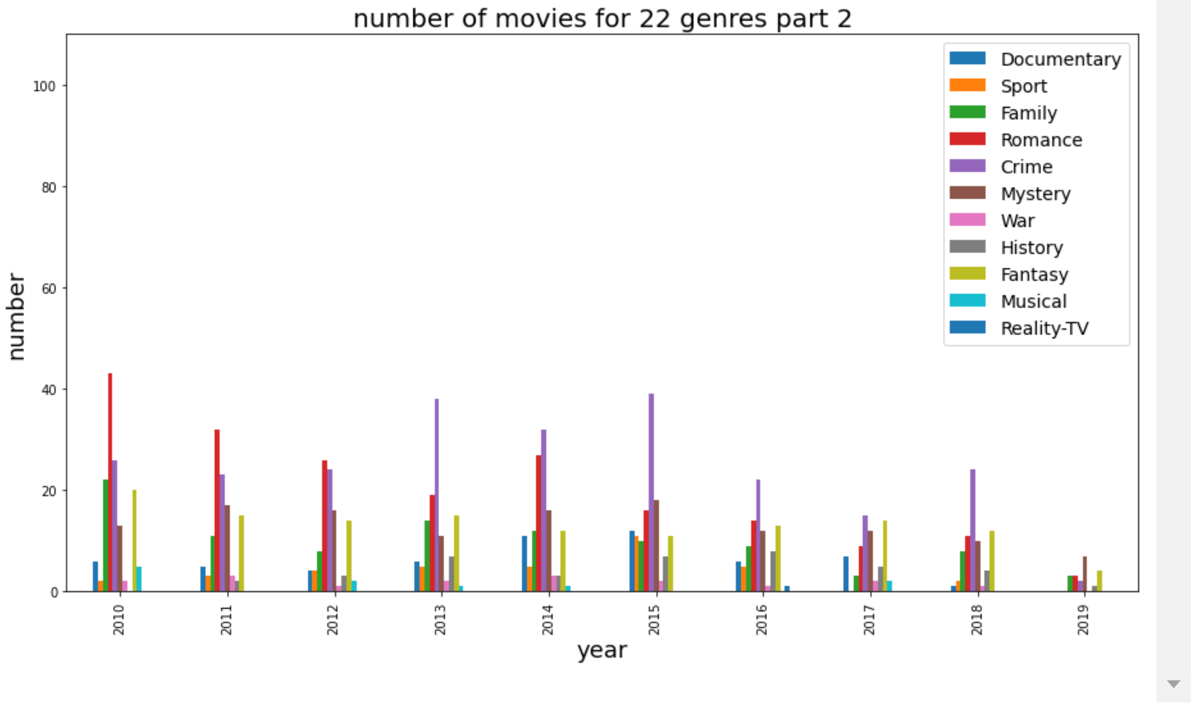
```
genrestmp  = genresrateyearsum['genres'].unique()
print(genrestmp)
# 1) nomoives
nomoviescols = []
for ii in range(11):
    nomoviescols.append("nomovies_"+genrestmp[ii])
ax = genresrateyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestmp[0:11],fontsize=14)
ax.set_title('number of movies for 22 genres part 1',fontsize=20)
nomoviescols = []
for ii in range(11):
    nomoviescols.append("nomovies_"+genrestmp[ii+11])
ax = genresrateyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestmp[11:22],fontsize=14)
ax.set_title('number of movies for 22 genres part 2',fontsize=20)
```

```
['Action' 'Adventure' 'Sci-Fi' 'Drama' 'Comedy' 'Horror' 'Thriller'
 'Biography' 'Music' 'Animation' 'Western' 'Documentary' 'Sport' 'Family'
 'Romance' 'Crime' 'Mystery' 'War' 'History' 'Fantasy' 'Musical'
 'Reality-TV']
```

Out[110]:

```
Text(0.5, 1.0, 'number of movies for 22 genres part 2')
```



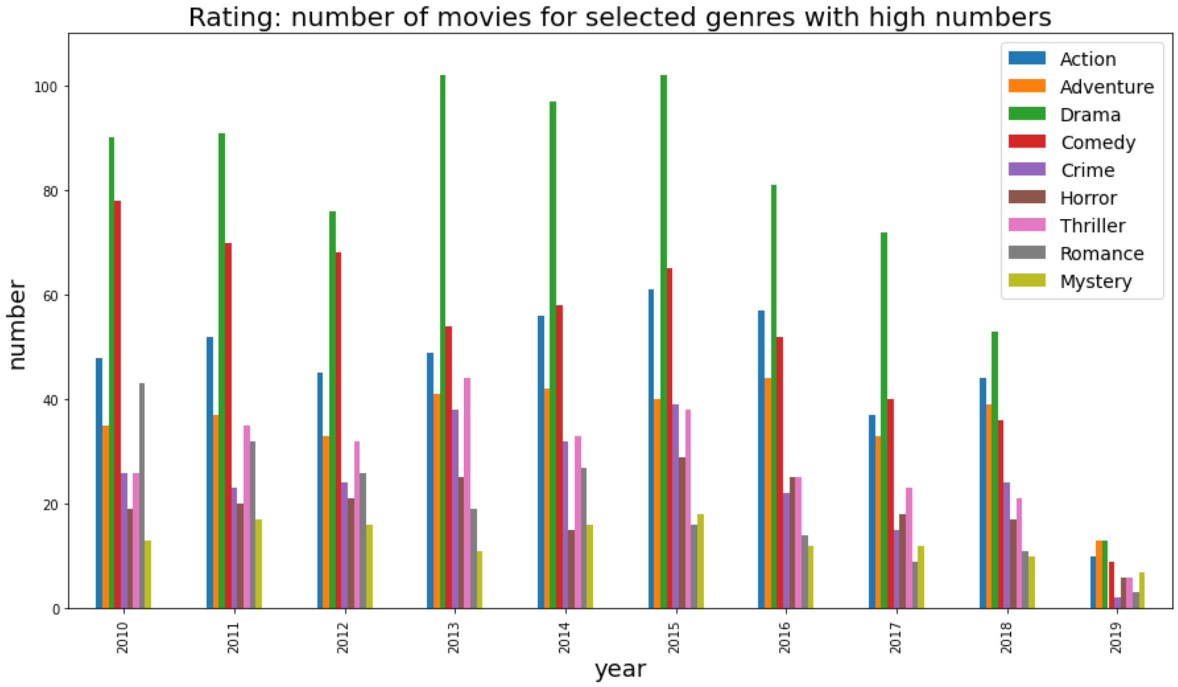

number of movies for 22 genres part 2
number
year
0
20
40
60
80
100
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
Documentary
Sport
Family
Romance
Crime
Mystery
War
History
Fantasy
Musical
Reality-TV

In [111]:

```
# From above figures for number of movies released each year for different categories,
# 'Action','Adventure','Drama','Comedy','Crime','Horror','Thriller', 'Romance',and 'Mystery
# have more released, therefore, plot these categories in a single figure
genrestsel = [ 'Action','Adventure','Drama','Comedy','Crime','Horror','Thriller', 'Romance'
nomoviescols = []
for ii in range(len(genrestsel)):
    nomoviescols.append("nomovies_"+genrestsel[ii])
ax = genresrateyearsumtmp.plot( x = "year", y =nomoviescols, kind="bar",figsize=(15,8))
ax.set_ylabel('number',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([0,110])
ax.legend(genrestsel,fontsize=14)
ax.set_title('Rating: number of movies for selected genres with high numbers',fontsize=20)
```

Out[111]:

```
Text(0.5, 1.0, 'Rating: number of movies for selected genres with high numbe
rs')
```



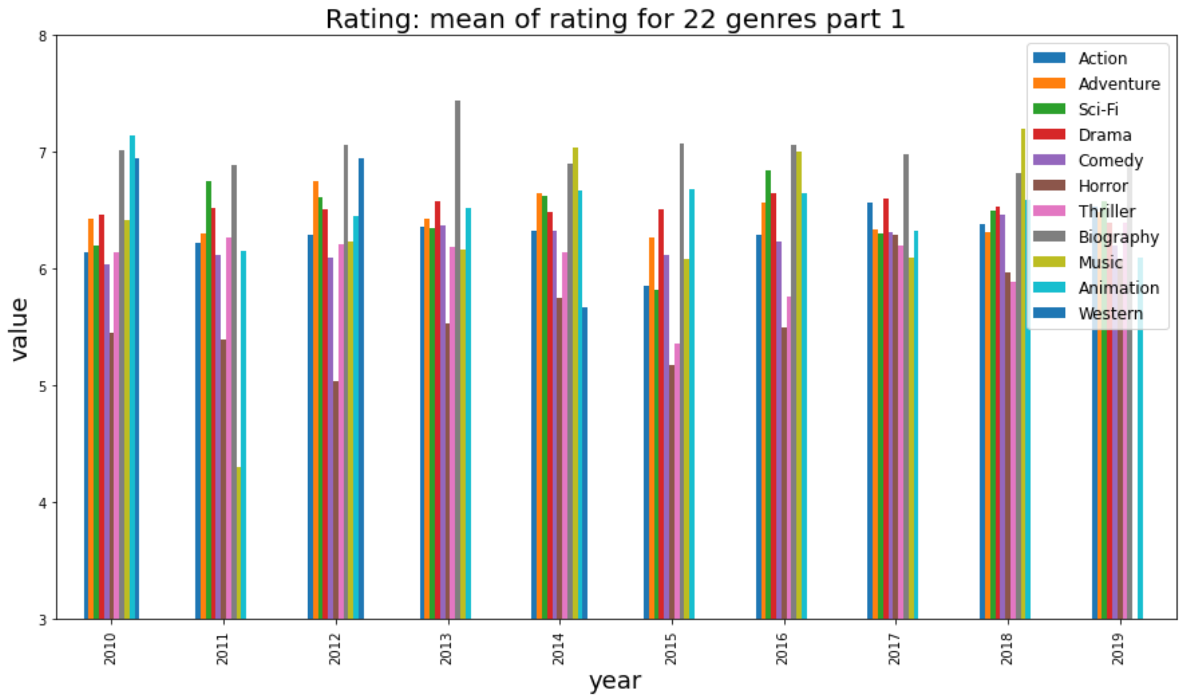
From the above figure, Drama, Comedy, Action,Adventure are the top four categories in terms of number of movies released each year between 2010 and 2019

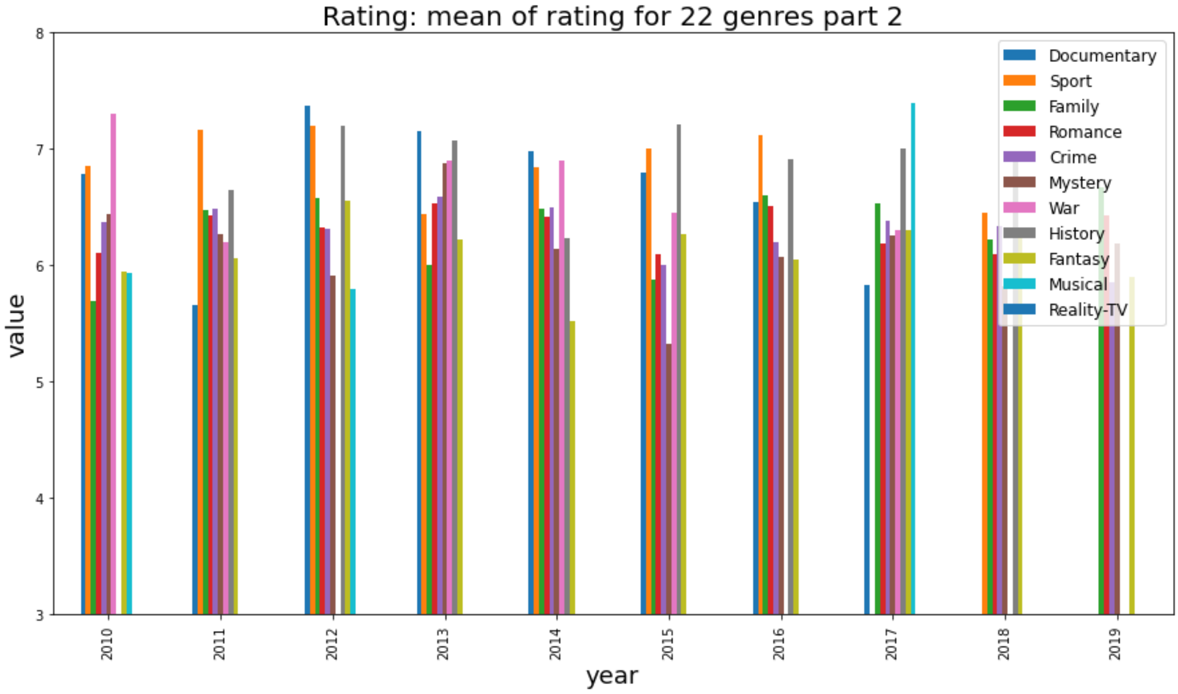
In [112]:

```
# 2) ratingmean
ratingmeancols = []
for ii in range(11):
    ratingmeancols.append("ratingmean_"+genrestmp[ii])
ax = genresrateyearsumtmp.plot( x = "year", y = ratingmeancols, kind="bar",figsize=(15,8))
ax.set_ylabel('value',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([3,8])
ax.legend(genrestmp[0:11],fontsize=12)
ax.set_title('Rating: mean of rating for 22 genres part 1',fontsize=20)

ratingmeancols = []
for ii in range(11):
    ratingmeancols.append("ratingmean_"+genrestmp[ii+11])
ax = genresrateyearsumtmp.plot( x = "year", y = ratingmeancols, kind="bar",figsize=(15,8))
ax.set_ylabel('value',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([3,8])
ax.legend(genrestmp[11:22],fontsize=12)
ax.set_title('Rating: mean of rating for 22 genres part 2',fontsize=20)
```

Out[112]:

```
Text(0.5, 1.0, 'Rating: mean of rating for 22 genres part 2')
```



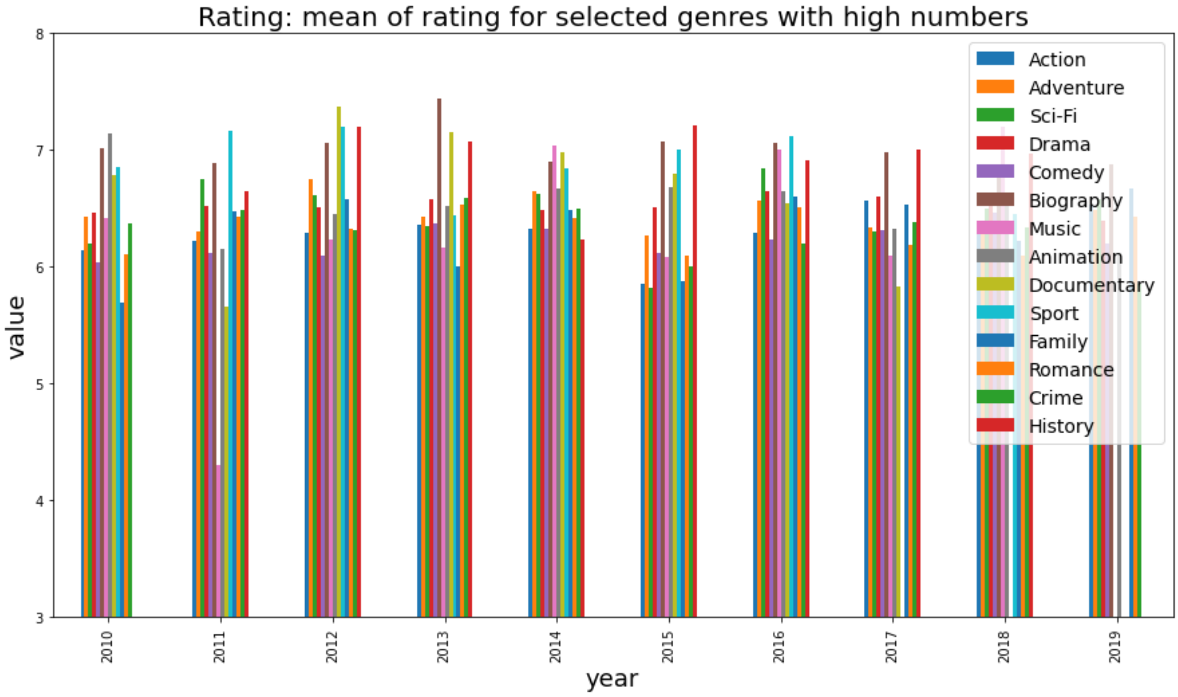

Rating: mean of rating for 22 genres part 2
value
year
3
4
5
6
7
8
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
Documentary
Sport
Family
Romance
Crime
Mystery
War
History
Fantasy
Musical
Reality-TV

In [113]:

```
# From above figures for mean rating for each year for different categories,
# 'Action' 'Adventure' 'Sci-Fi' 'Drama' 'Comedy'
# 'Biography' 'Music' 'Animation''Documentary' 'Sport' 'Family','Romance' 'Crime''History'
# have high rating values, therefore, plot these categories in a single figure
genrestsel = ['Action','Adventure','Sci-Fi','Drama','Comedy','Biography','Music','Animation
ratingmeancols = []
for ii in range(len(genrestsel)):
    ratingmeancols.append("ratingmean_"+genrestsel[ii])
ax = genresrateyearsumtmp.plot( x = "year", y =ratingmeancols, kind="bar",figsize=(15,8))
ax.set_ylabel('value',fontsize=18)
ax.set_xlabel('year',fontsize=18)
ax.set_ylim([3,8])
ax.legend(genrestsel,fontsize=14)
ax.set_title('Rating: mean of rating for selected genres with high numbers',fontsize=20)
```

Out[113]:

```
Text(0.5, 1.0, 'Rating: mean of rating for selected genres with high number
s')
```



In [114]:

```
# I also could calculate the mean rating for each category across 2010-2019
genresrategenressum = genresrateyearsum.groupby('genres').mean()
genresrategenressum = genresrategenressum.reset_index()
genresrategenressum = genresrategenressum.sort_values(by='ratingmean', ascending=False)
genresrategenressum = genresrategenressum.reset_index()
genresrategenressum.head(22)
```

Out[114]:

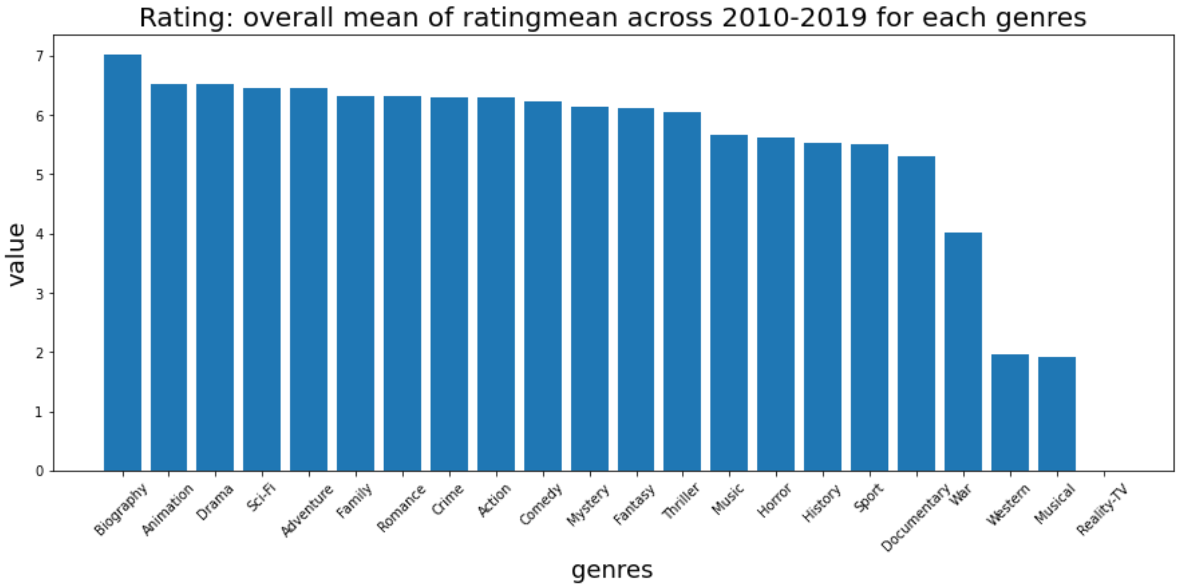
	index	genres	year	nomovies	ratingmean
0	3	Biography	2014.5	13.8	7.014217
1	2	Animation	2014.5	10.3	6.527364
2	7	Drama	2014.5	77.7	6.525539
3	17	Sci-Fi	2014.5	13.5	6.458045
4	1	Adventure	2014.5	35.7	6.453303
5	8	Family	2014.5	10.0	6.313985
6	16	Romance	2014.5	20.0	6.313853
7	5	Crime	2014.5	24.5	6.302245
8	0	Action	2014.5	45.9	6.296745
9	4	Comedy	2014.5	53.0	6.228296
10	14	Mystery	2014.5	13.2	6.130485
11	9	Fantasy	2014.5	13.0	6.113650
12	19	Thriller	2014.5	28.3	6.056561
13	12	Music	2014.5	5.5	5.653148
14	11	Horror	2014.5	19.5	5.620300
15	10	History	2014.5	4.0	5.525655
16	18	Sport	2014.5	3.7	5.507576
17	6	Documentary	2014.5	5.8	5.312872
18	20	War	2014.5	1.7	4.005000
19	21	Western	2014.5	1.1	1.956667
20	13	Musical	2014.5	1.1	1.914000
21	15	Reality-TV	2014.5	0.1	0.000000

In [115]:

```
fig, ax = plt.subplots(1,1,figsize=(15,6))
ax.bar(genresrategenressum.index,genresrategenressum.ratingmean)
ax.set_ylabel('value',fontsize=18)
ax.set_xlabel('genres',fontsize=18)
# ax.set_ylim([3,8])
ax.set_xticks(range(22))
# ax.set_xticklabels(['zero','two','four','six'])
ax.set_xticklabels(genresrategenressum['genres'])
plt.xticks(rotation = 45)
ax.set_title('Rating: overall mean of ratingmean across 2010-2019 for each genres',fontsize
```

Out[115]:

```
Text(0.5, 1.0, 'Rating: overall mean of ratingmean across 2010-2019 for each
genres')
```



End of Question 3: What types of films have good ratings?

Answer: Biography, Animation,Drama, Sci-Fi, and Adventure are top five for the rating

If combined both rating and box office, adventure and action should be invested in the newly found Microsoft

studio

In [ ]:

```
```