Investigate_a_Dataset

August 3, 2022

> **Tip**: Welcome to the Investigate a Dataset project! You will find tips in quoted sections like this to help organize your approach to your investigation. Once you complete this project, remove these **Tip** sections from your report before submission. First things first, you might want to double-click this Markdown cell and change the title so that it reflects your dataset and investigation.

1 Project: Investigate a Dataset - TMDb movie data

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

Certain columns, like 'cast' and 'genres', contain multiple values separated by pipe (|) characters. There are some odd characters in the 'cast' column. Don't worry about cleaning them. You can leave them as is. The final two columns ending with "_adj" show the budget and revenue of the associated movie in terms of 2010 dollars, accounting for inflation over time. ## Table of Contents

Introduction

Data Wrangling

Exploratory Data Analysis

Conclusions

Introduction

1.0.1 Dataset Description

> **Tip**: In this section of the report, provide a brief introduction to the dataset you've selected/downloaded for analysis. Read through the description available on the homepage-links present here. List all column names in each table, and their significance. In case of multiple tables, describe the relationship between tables.

The Movie Database (TMDB) is a community built movie and TV database. Every piece of data has been added by our amazing community dating back to 2008. TMDb's strong international focus and breadth of data is largely unmatched and something we're incredibly proud of. Put simply, we live and breathe community and that's precisely what makes us different.

The TMDB Advantage

- 1 Every year since 2008, the number of contributions to our database has increased. With over 400,000 developers and companies using our platform, TMDB has become a premiere source for metadata.

- 2 Along with extensive metadata for movies, TV shows and people, we also offer one of the best selections of high resolution posters and fanart. On average, over 1,000 images are added every single day.
- 3 We're international. While we officially support 39 languages we also have extensive regional data. Every single day TMDB is used in over 180 countries.
- 4 Our community is second to none. Between our staff and community moderators, we're always here to help. We're passionate about making sure your experience on TMDB is nothing short of amazing.
- 5 Trusted platform. Every single day our service is used by millions of people while we process over 3 billion requests. We've proven for years that this is a service that can be trusted and relied on.

source

1.0.2 Question(s) for Analysis

Tip: Clearly state one or more questions that you plan on exploring over the course of the report. You will address these questions in the **data analysis** and **conclusion** sections. Try to build your report around the analysis of at least one dependent variable and three independent variables. If you're not sure what questions to ask, then make sure you familiarize yourself with the dataset, its variables and the dataset context for ideas of what to explore.

Tip: Once you start coding, use NumPy arrays, Pandas Series, and DataFrames where appropriate rather than Python lists and dictionaries. Also, **use good coding practices**, such as, define and use functions to avoid repetitive code. Use appropriate comments within the code cells, explanation in the mark-down cells, and meaningful variable names.

1) Which genres are most popular from year to year?
2) What kinds of properties are associated with movies that have high revenues?
3)

[ ]:
```
```

[1]:
```
# Use this cell to set up import statements for all of the packages that you
#   plan to use.
# Remember to include a 'magic word' so that your visualizations are plotted
#   inline with the notebook. See this page for more:
#   http://ipython.readthedocs.io/en/stable/interactive/magics.html
# loading packages
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
# Upgrade pandas to use dataframe.explode() function.
# !pip install --upgrade pandas==0.25.0
```

Data Wrangling

> **Tip**: In this section of the report, you will load in the data, check for cleanliness, and then trim and clean your dataset for analysis. Make sure that you **document your data cleaning steps in mark-down cells precisely and justify your cleaning decisions.**

1.0.3 General Properties

> **Tip**: You should *not* perform too many operations in each cell. Create cells freely to explore your data. One option that you can take with this project is to do a lot of explorations in an initial notebook. These don't have to be organized, but make sure you use enough comments to understand the purpose of each code cell. Then, after you're done with your analysis, create a duplicate notebook where you will trim the excess and organize your steps so that you have a flowing, cohesive report.

[3]:
```
# Load your data and print out a few lines. Perform operations to inspect data
#   types and look for instances of missing or possibly errant data.
df = pd.read_csv('./Database_TMDb_movie_data/tmdb-movies.csv')
```

1.0.4 Printing DataFrame's data type and dimensions

[4]:
```
df.shape
```

[4]:
```
(10866, 21)
```

The TMdb dataset contains 10866 rows and 21 columns.

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10866 entries, 0 to 10865
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    10866 non-null  int64
 1   imdb_id               10856 non-null  object
 2   popularity            10866 non-null  float64
 3   budget                10866 non-null  int64
 4   revenue               10866 non-null  int64
 5   original_title        10866 non-null  object
 6   cast                  10790 non-null  object
 7   homepage              2936 non-null   object
 8   director              10822 non-null  object
 9   tagline               8042 non-null   object
 10  keywords              9373 non-null   object
```

```
 11  overview              10862 non-null  object
 12  runtime               10866 non-null  int64
 13  genres                10843 non-null  object
 14  production_companies  9836 non-null   object
 15  release_date          10866 non-null  object
 16  vote_count            10866 non-null  int64
 17  vote_average          10866 non-null  float64
 18  release_year          10866 non-null  int64
 19  budget_adj            10866 non-null  float64
 20  revenue_adj           10866 non-null  float64
dtypes: float64(4), int64(6), object(11)
memory usage: 1.7+ MB
```

As we can see there are missing values in some columns of our dataset. We'll handle them shortly.

1.0.5 Printing DataFrame's head

[6]:
```
df.head(5)
```

[6]:
```
      id    imdb_id  popularity     budget     revenue  \
0  135397  tt0369610   32.985763  150000000  1513528810
1   76341  tt1392190   28.419936  150000000   378436354
2  262500  tt2908446   13.112507  110000000   295238201
3  140607  tt2488496   11.173104  200000000  2068178225
4  168259  tt2820852    9.335014  190000000  1506249360

                original_title  \
0                Jurassic World
1            Mad Max: Fury Road
2                     Insurgent
3  Star Wars: The Force Awakens
4                     Furious 7

                                                cast  \
0  Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi…
1  Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic…
2  Shailene Woodley|Theo James|Kate Winslet|Ansel…
3  Harrison Ford|Mark Hamill|Carrie Fisher|Adam D…
4  Vin Diesel|Paul Walker|Jason Statham|Michelle …

                                        homepage          director  \
0                    http://www.jurassicworld.com/   Colin Trevorrow
1                      http://www.madmaxmovie.com/     George Miller
2    http://www.thedivergentseries.movie/#insurgent  Robert Schwentke
3  http://www.starwars.com/films/star-wars-episod…       J.J. Abrams
4                          http://www.furious7.com/         James Wan

                       tagline  …  \
```

```
0              The park is open.  …
1             What a Lovely Day.  …
2     One Choice Can Destroy You  …
3  Every generation has a story.  …
4            Vengeance Hits Home  …

                                          overview runtime  \
0  Twenty-two years after the events of Jurassic …     124
1  An apocalyptic story set in the furthest reach…     120
2  Beatrice Prior must confront her inner demons …     119
3  Thirty years after defeating the Galactic Empi…     136
4  Deckard Shaw seeks revenge against Dominic Tor…     137

                                     genres  \
0  Action|Adventure|Science Fiction|Thriller
1  Action|Adventure|Science Fiction|Thriller
2         Adventure|Science Fiction|Thriller
3   Action|Adventure|Science Fiction|Fantasy
4                       Action|Crime|Thriller

                                production_companies release_date vote_count  \
0  Universal Studios|Amblin Entertainment|Legenda…       6/9/15       5562
1  Village Roadshow Pictures|Kennedy Miller Produ…      5/13/15       6185
2  Summit Entertainment|Mandeville Films|Red Wago…      3/18/15       2480
3          Lucasfilm|Truenorth Productions|Bad Robot     12/15/15       5292
4  Universal Pictures|Original Film|Media Rights …       4/1/15       2947

   vote_average  release_year    budget_adj   revenue_adj
0           6.5          2015  1.379999e+08  1.392446e+09
1           7.1          2015  1.379999e+08  3.481613e+08
2           6.3          2015  1.012000e+08  2.716190e+08
3           7.5          2015  1.839999e+08  1.902723e+09
4           7.3          2015  1.747999e+08  1.385749e+09

[5 rows x 21 columns]
```

Here is presented the first 5 rows of the TMdb dataset, lets dive in.

The are `Id` and `imdb_id` columns corresponding to each film.

The film runtime `runtime`

The is `popularity` of the film

The film budget `budget_adj` and the film revenue `revenue_adj` adjusted to the inflation over time

The `genres` of the movie, the `production_compagnies` the `release_date`, the audience votes `vote_count` and the average vote `vote_average`.

1.0.6 Data Cleaning

> **Tip**: Make sure that you keep your reader informed on the steps that you are taking in your investigation. Follow every code cell, or every set of related code cells, with a markdown cell to describe to the reader what was found in the preceding cell(s). Try to make it so that the reader can then understand what they will be seeing in the following cell(s).

In this part there are columns that are not valuable for our analysis such as `id`, `imdb_id`, `homepage`, `tagline`,`keywords`, `overview`, `budget`and `revenue`.

We'll also identify and count rows with crucial missing values such as `revenue_adj`, `budget_adj`, `cast` and `genres`. Even though we could fill numeric columns with the `mean()`, how can we deal with missing `cast` or missing `genres` ? We can't predict or fill them.

For the numeric values we will count and if there are minor we'll drop all of them. Otherwise We'll fill them with the `mean()`.

[7]:
```
# After discussing the structure of the data and any problems that need to be
#   cleaned, perform those cleaning steps in the second part of this section.
```

[8]:
```
df.drop(['id', 'imdb_id', 'homepage', 'tagline','keywords', 'overview',␣
↪'budget', 'revenue'],
        axis=1,
        inplace=True)
```

Let's print the head of the new DataFrame.

[9]:
```
df.head()
```

[9]:
```
   popularity                original_title  \
0   32.985763                Jurassic World
1   28.419936           Mad Max: Fury Road
2   13.112507                     Insurgent
3   11.173104  Star Wars: The Force Awakens
4    9.335014                     Furious 7

                                                cast           director  \
0  Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi…    Colin Trevorrow
1  Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic…      George Miller
2  Shailene Woodley|Theo James|Kate Winslet|Ansel…   Robert Schwentke
3  Harrison Ford|Mark Hamill|Carrie Fisher|Adam D…        J.J. Abrams
4  Vin Diesel|Paul Walker|Jason Statham|Michelle …          James Wan

   runtime                                     genres  \
0      124  Action|Adventure|Science Fiction|Thriller
1      120  Action|Adventure|Science Fiction|Thriller
2      119         Adventure|Science Fiction|Thriller
3      136   Action|Adventure|Science Fiction|Fantasy
4      137                      Action|Crime|Thriller
```

```
                                production_companies release_date  vote_count  \
0  Universal Studios|Amblin Entertainment|Legenda…       6/9/15        5562
1  Village Roadshow Pictures|Kennedy Miller Produ…      5/13/15        6185
2  Summit Entertainment|Mandeville Films|Red Wago…      3/18/15        2480
3         Lucasfilm|Truenorth Productions|Bad Robot     12/15/15        5292
4  Universal Pictures|Original Film|Media Rights …       4/1/15        2947

   vote_average  release_year    budget_adj   revenue_adj
0           6.5          2015  1.379999e+08  1.392446e+09
1           7.1          2015  1.379999e+08  3.481613e+08
2           6.3          2015  1.012000e+08  2.716190e+08
3           7.5          2015  1.839999e+08  1.902723e+09
4           7.3          2015  1.747999e+08  1.385749e+09
```

1.0.7 Descriptive Statistics

Let's describe basic statistics for each numeric column of our dataset.

[10]:
```
df.describe()
```

[10]:
```
         popularity       runtime    vote_count  vote_average  release_year  \
count  10866.000000  10866.000000  10866.000000  10866.000000  10866.000000
mean       0.646441    102.070863    217.389748      5.974922   2001.322658
std        1.000185     31.381405    575.619058      0.935142     12.812941
min        0.000065      0.000000     10.000000      1.500000   1960.000000
25%        0.207583     90.000000     17.000000      5.400000   1995.000000
50%        0.383856     99.000000     38.000000      6.000000   2006.000000
75%        0.713817    111.000000    145.750000      6.600000   2011.000000
max       32.985763    900.000000   9767.000000      9.200000   2015.000000

         budget_adj   revenue_adj
count  1.086600e+04  1.086600e+04
mean   1.755104e+07  5.136436e+07
std    3.430616e+07  1.446325e+08
min    0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00
50%    0.000000e+00  0.000000e+00
75%    2.085325e+07  3.369710e+07
max    4.250000e+08  2.827124e+09
```

As we can see, there are problems with some rows like `runtime`, `budget_adj`, `revenue_adj` where most values are missing or are `Zeros`. We'll count them all.

Let's look after all columns with at least one `NaN` value

[11]:
```
col_with_nan_value = [col for col in df.columns if (df[col].isnull()).any() ]
col_with_nan_value
```

[11]:
```
['cast', 'director', 'genres', 'production_companies']
```

Let's look after all columns with at least one `Zeros` value

[12]:
```
col_with_zeros_value = [col for col in df.columns if (df[col]==0).any() ]
col_with_zeros_value
```

[12]:
```
['runtime', 'budget_adj', 'revenue_adj']
```

Let's count the `Zeros` values in `runtime`, `budget_adj`, `revenue_adj` columns.

[13]:
```
(df['revenue_adj']==0).sum()
```

[13]:
```
6016
```

There are 6016 `Zeros` values in the `revenue_adj` column. How Huge they are.

[14]:
```
(df['budget_adj']==0).sum()
```

[14]:
```
5696
```

There are 5696 `Zeros` values in the `budget_adj` column. How Huge they are.

[15]:
```
(df['runtime']==0).sum()
```

[15]:
```
31
```

There are 31 `Zeros` values in the `runtime` column.

We can't drop them, the number is so important.

We'll fill the zeros values per colum with the mean. Let's proceed.

Let's calculate and fill the zeros by the mean.

[16]:
```
mean = {col: df[col].mean() for col in col_with_zeros_value}
#print(mean)
fill = [df[col].replace(to_replace=0, value=mean[col], inplace=True) for col in␣
 ↪mean.keys()]
```

[17]:
```
# Deleting unnecessary variables
del col_with_nan_value, col_with_zeros_value, mean, fill
```

Let's look after the number of missing(`NaN`) values in our dataset

[18]:
```
df.isnull().any(axis=1).sum()
```

[18]:
```
1093
```

[19]:
```
100 * df.isnull().any(axis=1).sum() / df.shape[0]
```

[19]:
```
10.058899318976625
```

There are 1095 rows with at least one missing value, representing 10.07% of our dataset.

[20]:
```
100 * (df.shape[0] - df.isnull().any(axis=1).sum()) / df.shape[0]
```

[20]:
```
89.94110068102337
```

The correct values of our dataset represent 89.92% of it. Though we have 10866 rows, we can drop all of the incorrect values.

[21]:
```
df.dropna(inplace=True)
```

Let's look about duplicates in our dataset.

[22]:
```
df.duplicated().sum()
```

[22]:
```
1
```

There is one duplicate in our dataset. We'll drop it.

[23]:
```
df.drop_duplicates(inplace=True)
```

Now that we have cleaned our dataset, let's describe it again.

[24]:
```
df.describe()
```

[24]:

	popularity	runtime	vote_count	vote_average	release_year	budget_adj	revenue_adj
count	9772.000000	9772.000000	9772.000000	9772.000000	9772.000000	9.772000e+03	9.772000e+03
mean	0.694721	103.062415	239.312014	5.963528	2000.878428	2.794904e+07	8.345013e+07
std	1.036931	27.623159	603.011504	0.913174	13.036794	3.190074e+07	1.434707e+08
min	0.000188	3.000000	10.000000	1.500000	1960.000000	9.210911e-01	2.370705e+00
25%	0.232710	91.000000	18.000000	5.400000	1994.000000	1.755104e+07	4.908911e+07
50%	0.419762	100.000000	46.000000	6.000000	2005.000000	1.755104e+07	5.136436e+07
75%	0.776408	112.000000	173.000000	6.600000	2011.000000	2.464268e+07	5.136436e+07
max	32.985763	877.000000	9767.000000	8.700000	2015.000000	4.250000e+08	2.827124e+09

[25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9772 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   popularity            9772 non-null   float64
 1   original_title        9772 non-null   object 
 2   cast                  9772 non-null   object 
 3   director              9772 non-null   object 
 4   runtime               9772 non-null   float64
 5   genres                9772 non-null   object 
 6   production_companies  9772 non-null   object 
 7   release_date          9772 non-null   object 
 8   vote_count            9772 non-null   int64  
 9   vote_average          9772 non-null   float64
 10  release_year          9772 non-null   int64  
 11  budget_adj            9772 non-null   float64
 12  revenue_adj           9772 non-null   float64
dtypes: float64(5), int64(2), object(6)
memory usage: 1.0+ MB
```

Let's convert the data type of `release_date` to `datetime`

[26]:
```
df['release_date'] = pd.to_datetime(df['release_date'])
```

[27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9772 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   popularity            9772 non-null   float64       
 1   original_title        9772 non-null   object        
 2   cast                  9772 non-null   object        
 3   director              9772 non-null   object        
 4   runtime               9772 non-null   float64       
 5   genres                9772 non-null   object        
 6   production_companies  9772 non-null   object        
 7   release_date          9772 non-null   datetime64[ns]
 8   vote_count            9772 non-null   int64         
 9   vote_average          9772 non-null   float64       
 10  release_year          9772 non-null   int64         
 11  budget_adj            9772 non-null   float64       
 12  revenue_adj           9772 non-null   float64       
dtypes: datetime64[ns](1), float64(5), int64(2), object(5)
memory usage: 1.0+ MB
```

[28]:
```
df.head()
```

[28]:

```
   popularity                original_title  \
0   32.985763                Jurassic World
1   28.419936           Mad Max: Fury Road
2   13.112507                     Insurgent
3   11.173104  Star Wars: The Force Awakens
4    9.335014                     Furious 7

                                                cast          director  \
0  Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi…   Colin Trevorrow
1  Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic…     George Miller
2  Shailene Woodley|Theo James|Kate Winslet|Ansel…  Robert Schwentke
3  Harrison Ford|Mark Hamill|Carrie Fisher|Adam D…       J.J. Abrams
4  Vin Diesel|Paul Walker|Jason Statham|Michelle …         James Wan

   runtime                                     genres  \
0    124.0  Action|Adventure|Science Fiction|Thriller
1    120.0  Action|Adventure|Science Fiction|Thriller
2    119.0         Adventure|Science Fiction|Thriller
3    136.0   Action|Adventure|Science Fiction|Fantasy
4    137.0                     Action|Crime|Thriller

                              production_companies release_date  vote_count  \
0  Universal Studios|Amblin Entertainment|Legenda…   2015-06-09        5562
1  Village Roadshow Pictures|Kennedy Miller Produ…   2015-05-13        6185
2  Summit Entertainment|Mandeville Films|Red Wago…   2015-03-18        2480
3          Lucasfilm|Truenorth Productions|Bad Robot   2015-12-15        5292
4  Universal Pictures|Original Film|Media Rights …   2015-04-01        2947

   vote_average  release_year    budget_adj   revenue_adj
0           6.5          2015  1.379999e+08  1.392446e+09
1           7.1          2015  1.379999e+08  3.481613e+08
2           6.3          2015  1.012000e+08  2.716190e+08
3           7.5          2015  1.839999e+08  1.902723e+09
4           7.3          2015  1.747999e+08  1.385749e+09
```

For further analysis, we will add new columns in our dataframe. - `gain_adj` the gain value (`revenue_adj - budget_adj`) for each film - `release_month` the month value of the release date - `release_month_name` the month name of the release date - `genre_01` for the first genre of the film - `genre_02` for the second genre of the film - `genre_03` for the third genre of the film - `director_01` for the first director of the film - `director_02` for the co-director of the film - `prod_comp_01` for the first production company of the film. - `prod_comp_02` for the second production company of the film. - `cast_num` the number of cast members of the film - `actor_01` the first actor of the film - `actor_02` the second actor of the film - `actor_03` the third actor of the film - `actor_04` the fourth actor of the film - `year_by_5` group years by 5 of length. Ex: 1965[included]-1970[excluded], 1970[included]-1975[excluded] and so on - `year_by_10` group years by 10 of length.

Ex: 1965[included]-1975[excluded], 1975[included]-1985[excluded] and so on

The list of all actors who appeared in a film and the number of their appearance overall time.

[29]:
```
df['gain_adj'] = df['revenue_adj'] - df['budget_adj']
```

[30]:
```
df['release_month'] = pd.DatetimeIndex(df['release_date']).month
```

[31]:
```
df['release_month_name'] = pd.DatetimeIndex(df['release_date']).month_name()
```

[32]:
```
5 * (1967//5), 5 * (1967//5 + 1)
```

[32]:
```
(1965, 1970)
```

[33]:
```
5 * (1970//5), 5 * (1970//5 + 1)
```

[33]:
```
(1970, 1975)
```

[ ]:
```

```

[34]:
```
df['year_by_5'] = df['release_year'].apply(lambda val: f'{5 * (val//5)}-{5 *␣
↪(val//5 + 1)}')
```

[35]:
```
df['year_by_10'] = df['release_year'].apply(lambda val: f'{10 * (val//10)}-{10␣
↪* (val//10 + 1)}')
```

[36]:
```
df['genre_01'] = df['genres'].apply(lambda val: f"{val.split('|')[0] if len(val.
↪split('|')) > 0 else val}")
```

[37]:
```
df['genre_02'] = df['genres'].apply(lambda val: f"{val.split('|')[1] if len(val.
↪split('|')) > 1 else val}")
```

[38]:
```
df['genre_03'] = df['genres'].apply(lambda val: f"{val.split('|')[2] if len(val.
↪split('|')) > 2 else val}")
```

[39]:
```
df['director_01'] = df['director'].apply(lambda val: f"{val.split('|')[0] if␣
↪len(val.split('|')) > 0 else val}")
```

[40]:
```
df['director_02'] = df['director'].apply(lambda val: f"{val.split('|')[1] if␣
↪len(val.split('|')) > 1 else val}")
```

[41]:
```
df['prod_comp_01'] = df['production_companies'].apply(lambda val: f"{val.
↪split('|')[0] if len(val.split('|')) > 0 else val}")
```

[42]:
```
df['prod_comp_02'] = df['production_companies'].apply(lambda val: f"{val.
↪split('|')[1] if len(val.split('|')) > 1 else val}")
```

[43]:
```
df['cast_num'] = df['cast'].apply(lambda val: int(f"{len(val.split('|')) if␣
↪len(val.split('|')) > 0 else np.nan}"))
```

[44]:
```
df['actor_01'] = df['cast'].apply(lambda val: f"{val.split('|')[0] if len(val.
↪split('|')) > 0 else np.nan}")
```

[45]:
```
df['actor_02'] = df['cast'].apply(lambda val: f"{val.split('|')[1] if len(val.
↪split('|')) > 1 else np.nan}")
```

[46]:
```
df['actor_03'] = df['cast'].apply(lambda val: f"{val.split('|')[2] if len(val.
↪split('|')) > 2 else np.nan}")
```

[47]:
```
df['actor_04'] = df['cast'].apply(lambda val: f"{val.split('|')[3] if len(val.
↪split('|')) > 3 else np.nan}")
```

[48]:
```
df.head()
```

[48]:

	popularity	original_title	cast	director	runtime	genres	production_companies	release_date	vote_count	vote_average	…	genre_03	director_01	director_02
0	32.985763	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi…	Colin Trevorrow	124.0	Action|Adventure|Science Fiction|Thriller	Universal Studios|Amblin Entertainment|Legenda…	2015-06-09	5562	6.5	…	Science Fiction	Colin Trevorrow	Colin Trevorrow
1	28.419936	Mad Max: Fury Road	Tom Hardy|Charlize Theron|Hugh Keays-Byrne|Nic…	George Miller	120.0	Action|Adventure|Science Fiction|Thriller	Village Roadshow Pictures|Kennedy Miller Produ…	2015-05-13	6185					
2	13.112507	Insurgent	Shailene Woodley|Theo James|Kate Winslet|Ansel…	Robert Schwentke	119.0	Adventure|Science Fiction|Thriller	Summit Entertainment|Mandeville Films|Red Wago…	2015-03-18	2480					
3	11.173104	Star Wars: The Force Awakens	Harrison Ford|Mark Hamill|Carrie Fisher|Adam D…	J.J. Abrams	136.0	Action|Adventure|Science Fiction|Fantasy	Lucasfilm|Truenorth Productions|Bad Robot	2015-12-15	5292					
4	9.335014	Furious 7	Vin Diesel|Paul Walker|Jason Statham|Michelle …	James Wan	137.0	Action|Crime|Thriller	Universal Pictures|Original Film|Media Rights …	2015-04-01	2947					

```
1           7.1  …  Science Fiction     George Miller     George Miller
2           6.3  …         Thriller  Robert Schwentke  Robert Schwentke
3           7.5  …  Science Fiction       J.J. Abrams       J.J. Abrams
4           7.3  …         Thriller         James Wan         James Wan

               prod_comp_01                prod_comp_02 cast_num  \
0          Universal Studios        Amblin Entertainment        5
1  Village Roadshow Pictures  Kennedy Miller Productions        5
2       Summit Entertainment            Mandeville Films        5
3                  Lucasfilm       Truenorth Productions        5
4          Universal Pictures               Original Film        5

           actor_01             actor_02          actor_03            actor_04
0       Chris Pratt  Bryce Dallas Howard       Irrfan Khan   Vincent D'Onofrio
1         Tom Hardy      Charlize Theron  Hugh Keays-Byrne      Nicholas Hoult
2  Shailene Woodley           Theo James      Kate Winslet        Ansel Elgort
3     Harrison Ford          Mark Hamill     Carrie Fisher         Adam Driver
4        Vin Diesel          Paul Walker     Jason Statham  Michelle Rodriguez

[5 rows x 30 columns]
```

[49]:
```
df.to_csv('TMdb_edited.csv', index=False)
```

[50]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9772 entries, 0 to 10865
Data columns (total 30 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   popularity            9772 non-null   float64
 1   original_title        9772 non-null   object
 2   cast                  9772 non-null   object
 3   director              9772 non-null   object
 4   runtime               9772 non-null   float64
 5   genres                9772 non-null   object
 6   production_companies  9772 non-null   object
 7   release_date          9772 non-null   datetime64[ns]
 8   vote_count            9772 non-null   int64
 9   vote_average          9772 non-null   float64
 10  release_year          9772 non-null   int64
 11  budget_adj            9772 non-null   float64
 12  revenue_adj           9772 non-null   float64
 13  gain_adj              9772 non-null   float64
 14  release_month         9772 non-null   int64
 15  release_month_name    9772 non-null   object
 16  year_by_5             9772 non-null   object
```

```
 17  year_by_10            9772 non-null   object
 18  genre_01              9772 non-null   object
 19  genre_02              9772 non-null   object
 20  genre_03              9772 non-null   object
 21  director_01           9772 non-null   object
 22  director_02           9772 non-null   object
 23  prod_comp_01          9772 non-null   object
 24  prod_comp_02          9772 non-null   object
 25  cast_num              9772 non-null   int64
 26  actor_01              9772 non-null   object
 27  actor_02              9772 non-null   object
 28  actor_03              9772 non-null   object
 29  actor_04              9772 non-null   object
dtypes: datetime64[ns](1), float64(6), int64(4), object(19)
memory usage: 2.3+ MB
```

We define a function `extract()` to extract elements from a `pd.Series` in rows that contain `|` and return a `pd.DataFrame` of all elements with their occurences

Steps: 1) We extract elements from the `pd.Series` object with the help of the function `pd.Series.apply(lambda val: val.split("|"))` 2) We create a list of all these elements 3) Now we'll create a `dict` object to count every element occurrence with the help of this source and we will sort to firstly have the most popular element, with the help of this source 4) After that we'll convert our `dict` objet to `pd.DataFrame` object with the help of this source

[51]:
```
def extract(serie: pd.Series)-> pd.DataFrame:
    group_set = []
    for element in serie.apply(lambda val: val.split("|")) :
        #print(element)
        for subelt in element:
            #print(subelt)
            group_set.append(f'{subelt.strip()}')

    # Now we create an array of items
    group_set = np.array(group_set)

    # We create a dict object key: actor, value: number of appearances
    # Now we count every actor apparition in the `group_set`
    # variable with np.count_nonzero() and np.unique()
    dict_elts = {actor: np.count_nonzero(group_set == actor) for actor in np.
 ↪unique(group_set)}

    # We sort the dict with the most popular actor first (according
    # to the maximum number of appearances)
    dict_elts = dict(sorted(dict_actor.items(), key=lambda item: item[1],␣
 ↪reverse=True))

    # After that we'll convert our `dict` objet to `pd.DataFrame` object
```

```
    df_elts = pd.DataFrame.from_dict(data=dict_elts, orient='index',␣
↪columns=['value'])

    # We return the pd.DataFrame object
    return df_elts
```

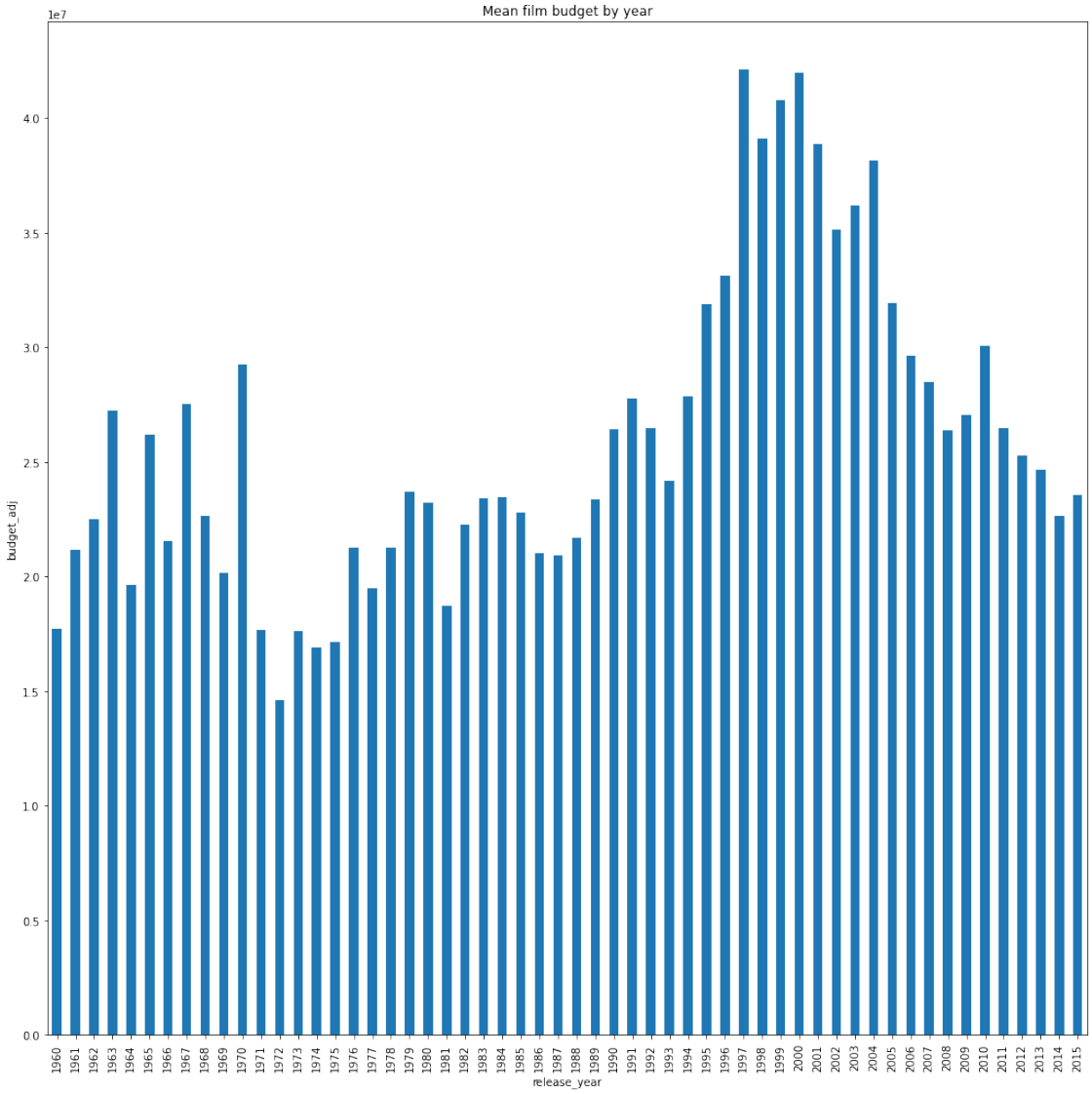
Exploratory Data Analysis

> **Tip**: Now that you've trimmed and cleaned your data, you're ready to move on to exploration. **Compute statistics** and **create visualizations** with the goal of addressing the research questions that you posed in the Introduction section. You should compute the relevant statistics throughout the analysis when an inference is made about the data. Note that at least two or more kinds of plots should be created as part of the exploration, and you must compare and show trends in the varied visualizations.
>
> **Tip**: - Investigate the stated question(s) from multiple angles. It is recommended that you be systematic with your approach. Look at one variable at a time, and then follow it up by looking at relationships between variables. You should explore at least three variables in relation to the primary question. This can be an exploratory relationship between three variables of interest, or looking at how two independent variables relate to a single dependent variable of interest. Lastly, you should perform both single-variable (1d) and multiple-variable (2d) explorations.

1.0.8 Research Question 1: What is the mean film budget over the years?

For that we will group our dataset by years and get the mean film budget for each year and plot.

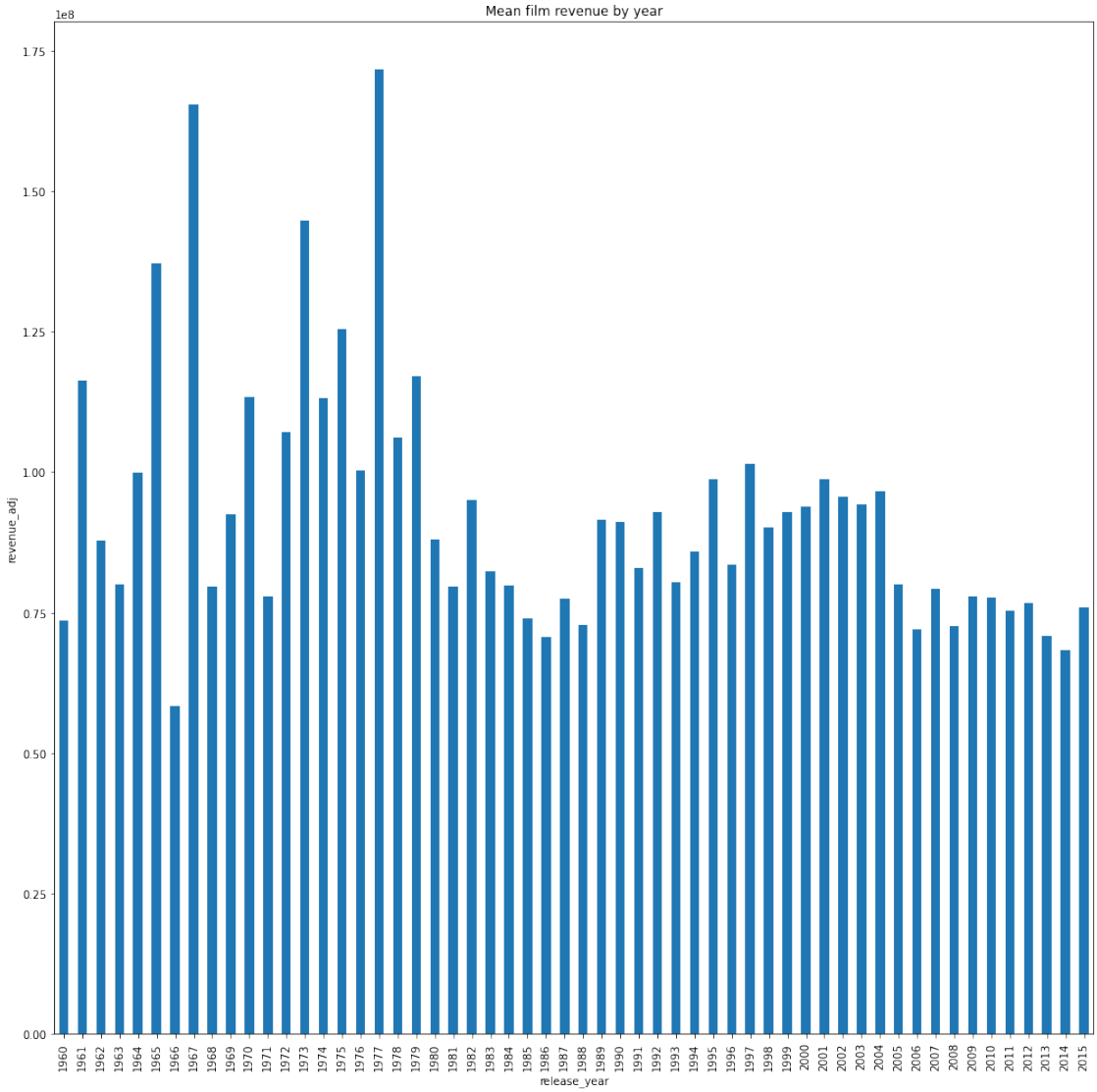
[52]:
```
df.groupby('release_year').mean()['budget_adj'].plot(kind='bar',
                                                     figsize=(17, 17),
                                                     title='Mean film budget by␣
↪year',
                                                     ylabel='budget_adj');
```



As we can see, there has been an increasing films budgets in the early 2000's, why? Perhaps beacause of the vulgarisation of the films distribution means, Internet apparition, devices, communication possibilities, and so on...

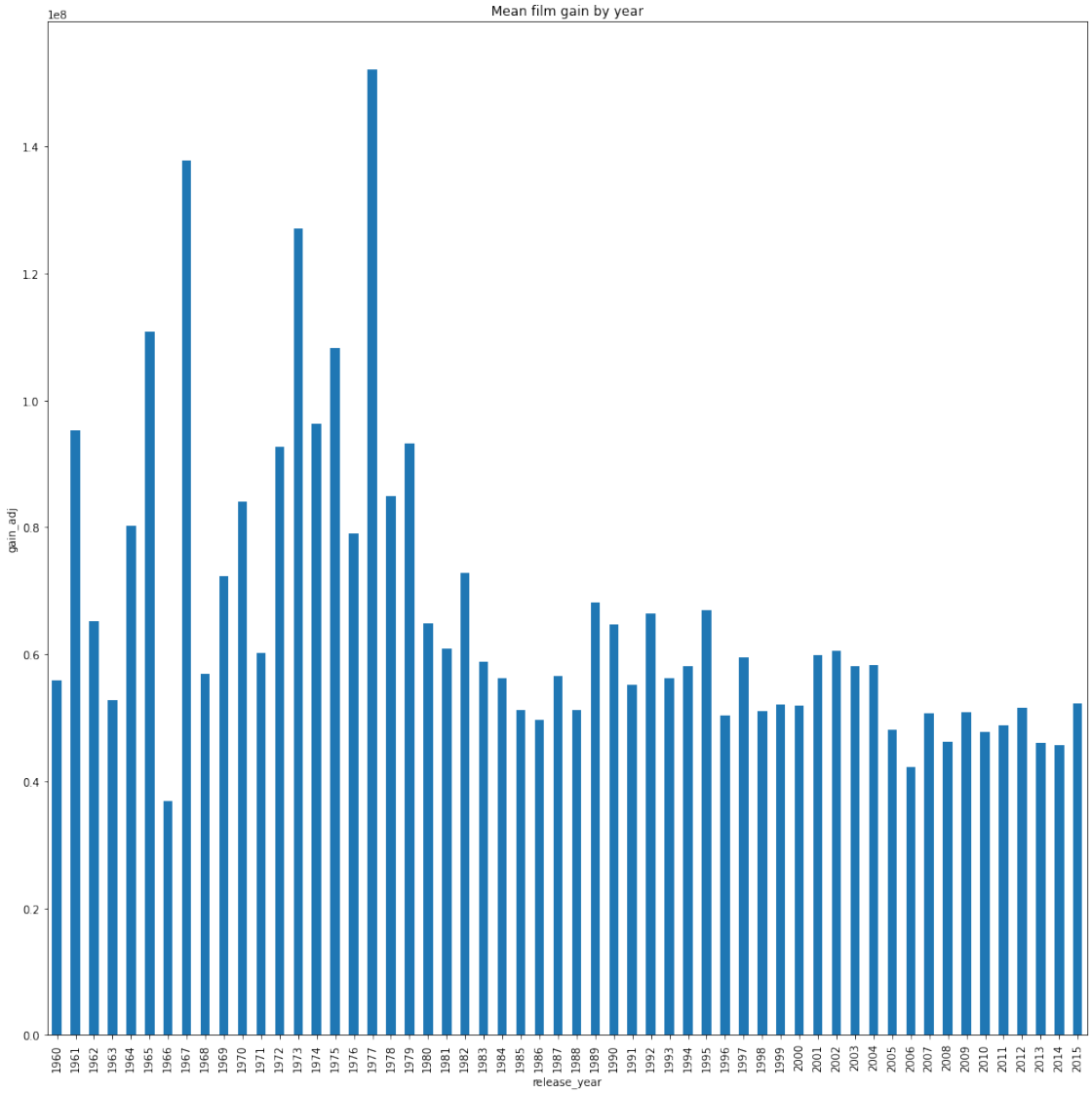
1.0.9 Research Question 2 What is the mean film revenue over the years?

[53]:
```
df.groupby('release_year').mean()['revenue_adj'].plot(kind='bar',
                                                      figsize=(17, 17),
                                                      title='Mean film revenue␣
↪by year',
                                                      ylabel='revenue_adj');
```



1.0.10 Research Question 3 What is the mean film gain over the years?

[54]:
```
df.groupby('release_year').mean()['gain_adj'].plot(kind='bar',
                                                   figsize=(17, 17),
                                                   title='Mean film gain by␣
 ↪year',
                                                   ylabel='gain_adj');
```



[55]:
```
df.groupby('release_year').mean()['gain_adj'].min()
```

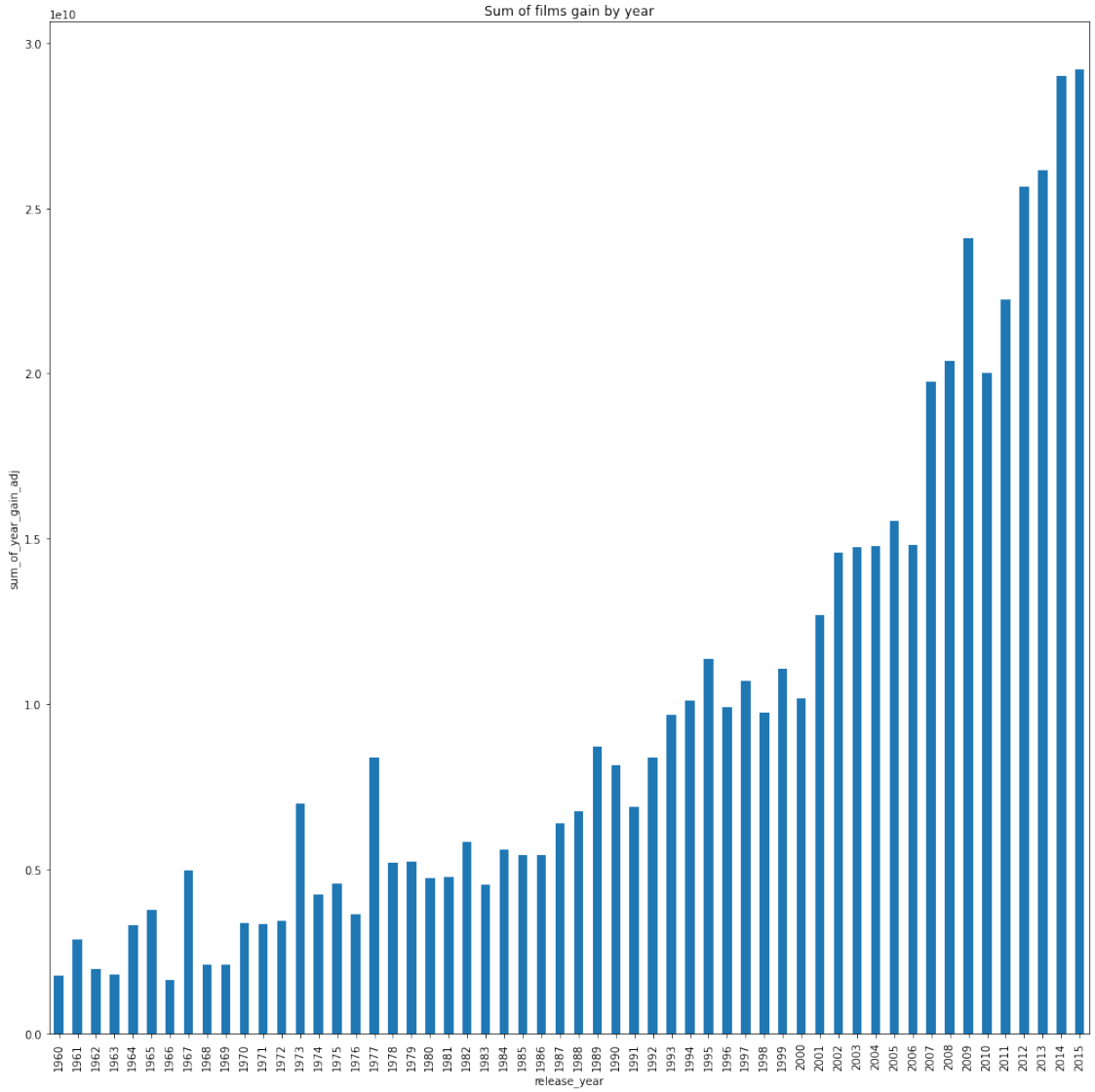
[55]:
```
36913300.504791535
```

We can surely say that the film industry is rent. the minimum mean gain overall time is around 40 000 000 $.

1.0.11 Research Question 4 What is the total year film gain over the years?

[56]:
```
df.groupby('release_year').sum()['gain_adj'].plot(kind='bar',
                                                  figsize=(17, 17),
                                                  title='Sum of films gain by␣
 ↪year',
```

```
↪ylabel='sum_of_year_gain_adj');
```

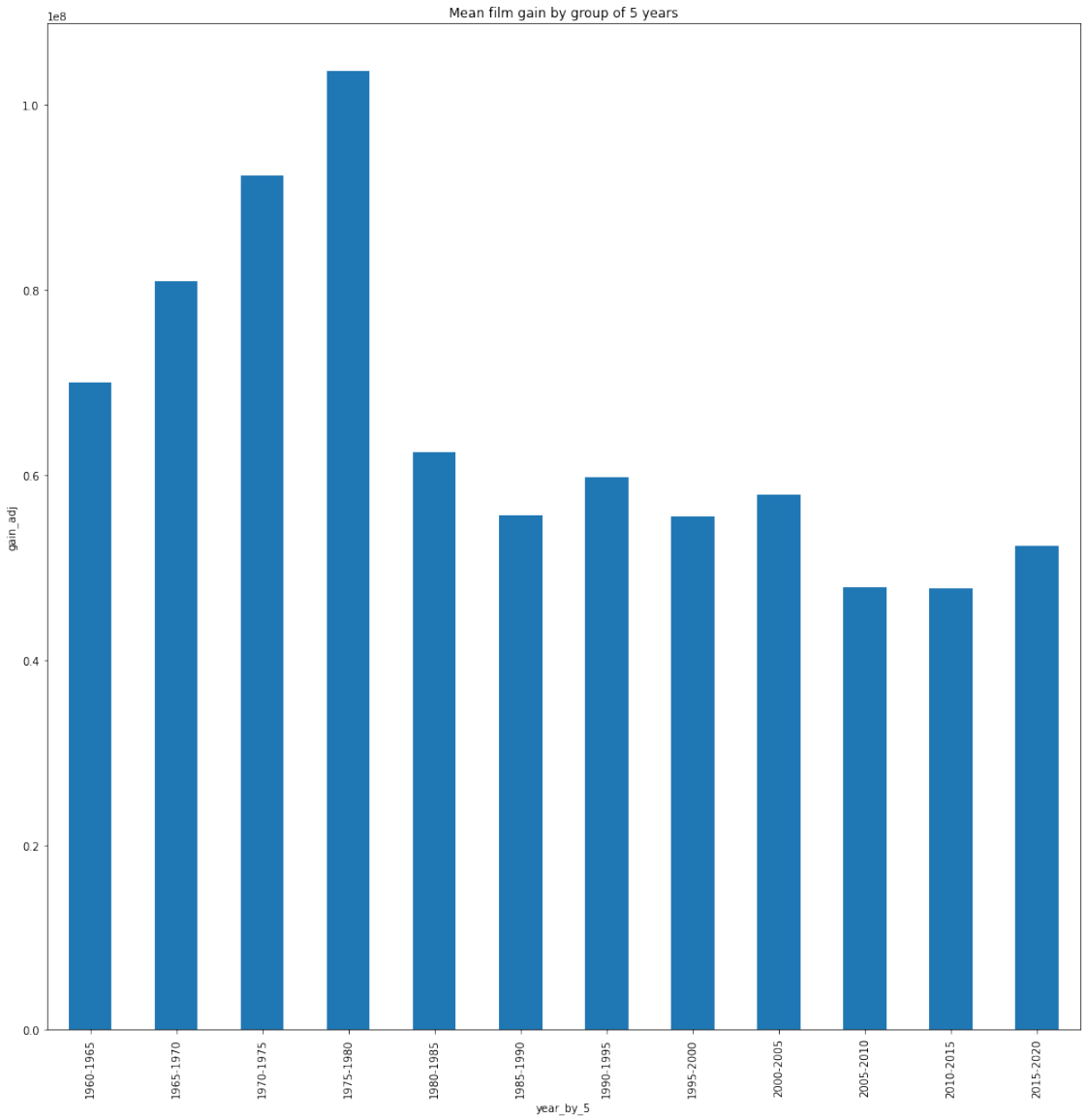


We can say that the total year film gain evolution over the years is greatly increasing. Year `2014` is the year of biggest gain overall time, and year `2015` comes closer.

What if we group `release_year` by 5 years?? Let's see what happens.

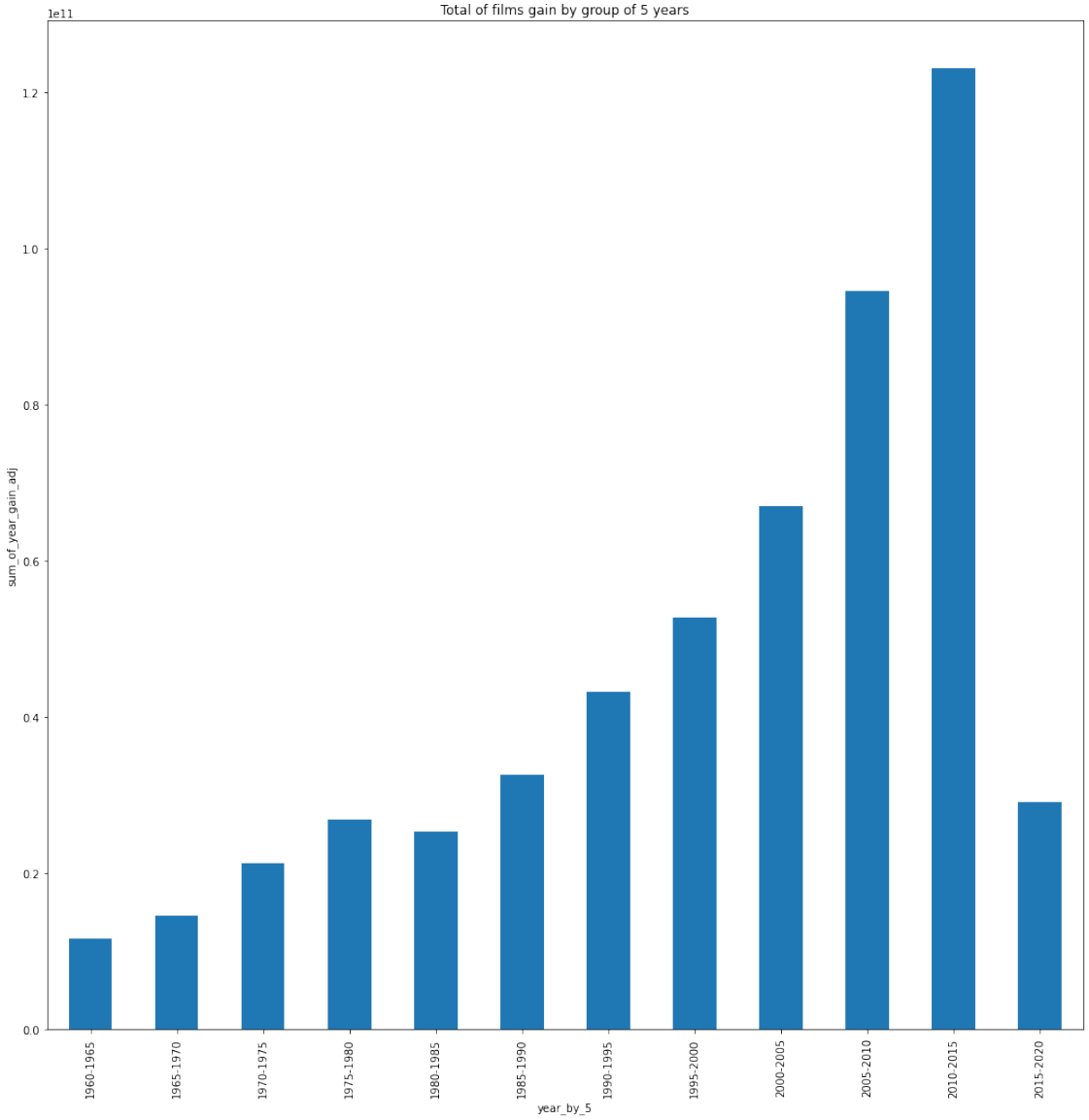
1.0.12 Research Question 5 What is the mean film gain over the years grouped by 5 years?

[57]:
```
df.groupby('year_by_5').mean()['gain_adj'].plot(kind='bar',
                                                figsize=(17, 17),
                                                title='Mean film gain by␣
↪group of 5 years',
                                                ylabel='gain_adj');
```



1.0.13 Research Question 6 What is the total year film gain over the years by group of 5 years?

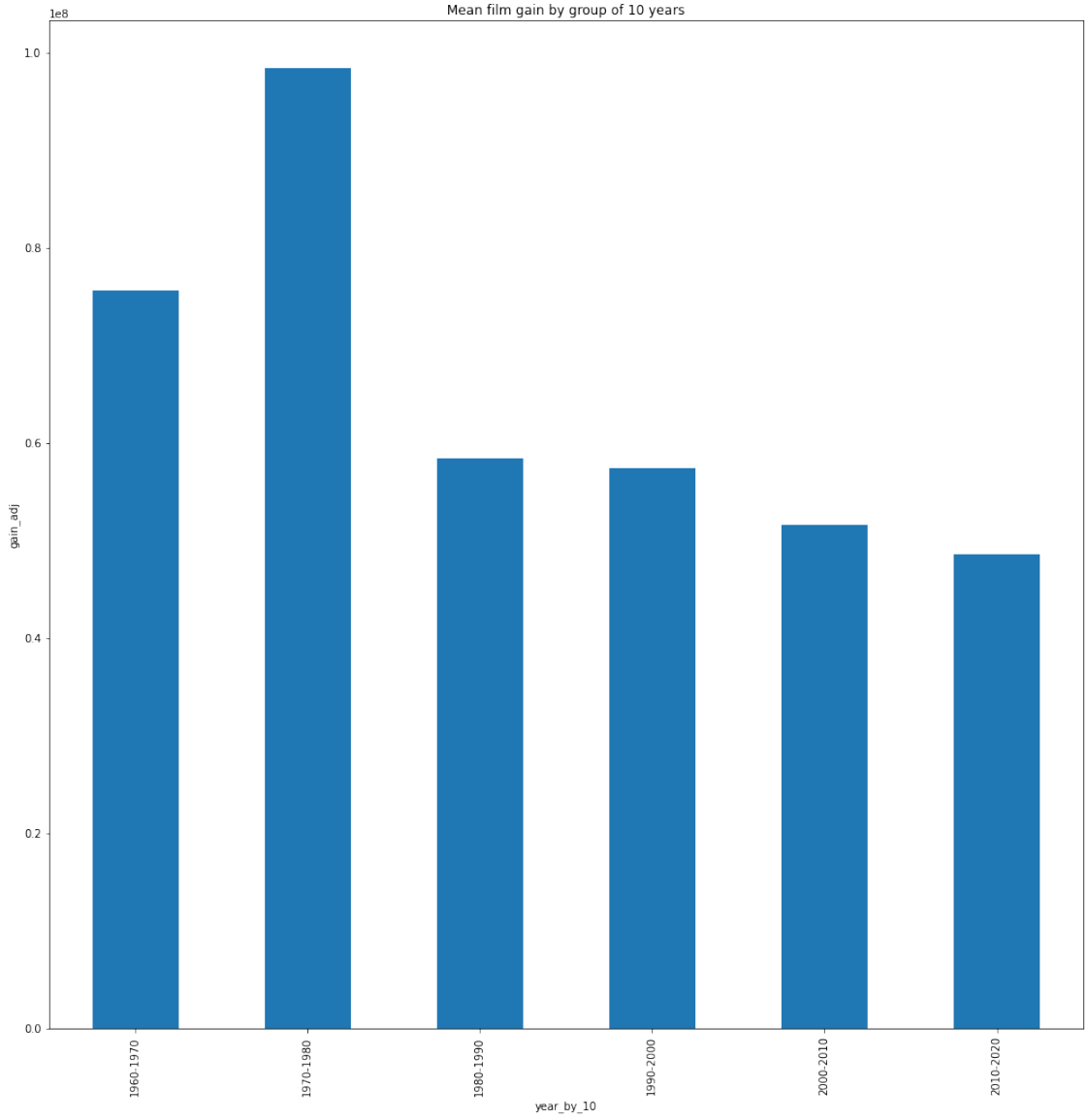
[58]:
```
df.groupby('year_by_5').sum()['gain_adj'].plot(kind='bar',
                                               figsize=(17, 17),
                                               title='Total of films gain by␣
 ↪group of 5 years',
                                               ylabel='sum_of_year_gain_adj');
```



What if we group `release_year` by 10 years?? Let's see what happens.

1.0.14 Research Question 7 What is the mean film gain over the years grouped by 10 years?

[59]:
```
df.groupby('year_by_10').mean()['gain_adj'].plot(kind='bar',
                                                 figsize=(17, 17),
                                                 title='Mean film gain by␣
↪group of 10 years',
                                                 ylabel='gain_adj');
```



1.0.15 Research Question 8 What is the best principal film genre overall time ?

Let's get the dataframe of all film genre produced overall time

[60]:
```
df_genres = extract(df['genres'])
#df_genres
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [60], in <cell line: 1>()
----> 1 df_genres = extract(df['genres'])

Input In [51], in extract(serie)
     10 group_set = np.array(group_set)
     12 # We create a dict object key: actor, value: number of appearances
     13 # Now we count every actor apparition in the `group_set`
     14 # variable with np.count_nonzero() and np.unique()
---> 15 dict_elts = {actor: np.count_nonzero(all_actors_apparition == actor) fo␣
 ↪actor in np.unique(all_actors_apparition)}
     17 # We sort the dict with the most popular actor first (according
     18 # to the maximum number of appearances)
     19 dict_elts = dict(sorted(dict_actor.items(), key=lambda item: item[1],␣
 ↪reverse=True))

NameError: name 'all_actors_apparition' is not defined
```

Now that we've converted to `pd.DataFrame`, let's plot the barchart of the 50 most popular genres according to their number of production overall time.

[ ]:
```
df_genres.iloc[:50].plot(kind='barh',
                                    figsize=(17, 17),
                                    title='Most popular genres overall time',
                                    xlabel='Genre name',
                                    ylabel='Number of produced films',
                                    alpha=0.8,
                                    color='gold');
```

The most popular actor overall time is `Robert De Niro` followed by `Samuel L. Jackson`.

[ ]:
```
df[['genre_01']].value_counts()
```

Let's plot this result.

[ ]:
```
df['genre_01'].value_counts().plot(kind='bar',
                                    figsize=(17, 17),
                                    title="Number of films by genre overall␣
 ↪time",
                                    xlabel='Film Genre',
                                    ylabel='Number of films');
```

The top 3 films genre are in order of popularity related to the total number of productions overall

time: 1) `Drama` 2) `Comedy` 3) `Action`

The best genres are `Drama` and `Comedy`. If a film production company follow this, there will certainly be successfull in term of revenue.

1.0.16 Research Question 9 What is the cast number for films overall time?

[ ]:
```
df['cast_num'].value_counts().plot(kind='bar',
                                   figsize=(17, 17),
                                   title='Number of films by number of cast',
                                   xlabel='Number fo actors',
                                   ylabel='number of films',
                                   alpha=0.5,
                                   color='green');
```

Looks like overall time, films used to have 5 principal actors in thier cast.

1.0.17 Research Question 10 What is the mean runtime evolution over the years ?

[ ]:
```
df.groupby('release_year').mean()['runtime'].describe()
```

The mean film runtime overall time is `106 min` and film runtime is between `103 min` and `109 min`.

Let's show our results.

[ ]:
```
df.groupby('release_year').mean()['runtime'].plot(kind='bar',
                                                  figsize=(17, 17),
                                                  title='Mean runtime␣
 ↪evolution',
                                                  ylabel='mean runtime',
                                                  color = 'green');
```

[ ]:
```

```

1.0.18 Research Question 11 What is the runtime of biggest films over the years?

[ ]:
```
df.groupby('release_year')['runtime'].max().plot(kind='bar',
                                                 figsize=(17, 17),
                                                 title='Sum of films gain by␣
 ↪group of 10 years',
                                                 ␣
 ↪ylabel='sum_of_year_gain_adj');
```

Yeah, it is confusing and unbelievable, but I double checked the dataset, and there is no error from me though. We can conclude that there are unrealistic values within our dataset, regarding the maximum film runtime over the years.

Let's get the dataframe of all the actors who ever casted in a movie overall time

```
[ ]: df_actors = extract(df['cast'])
     #df_actors
```

Now that we've converted to `pd.DataFrame`, let's plot the barchart of the 50 most popular actors according to their number of appearances overall time.

```
[ ]: df_actors.iloc[:50].plot(kind='barh',
                                    figsize=(17, 17),
                                    title='Most popular actors overall time',
                                    xlabel='Actor name',
                                    ylabel='Number of appearances',
                                    alpha=0.8,
                                    color='gold');
```

The most popular actor overall time is `Robert De Niro` followed by `Samuel L. Jackson`.

Let's update our `extract()` function in other to

```
[ ]: df['cast'].apply(lambda val: tuple(val.split("|")))
```

```
[ ]: #Looking for all the actors ever casted overall time
     np.unique(np.append(arr=[],  values=np.array(df['cast'].apply(lambda val: val.
      ↪split("|")))))
```

1.0.19 Research Question 12 What is the biggest production company overall time ?

```
[ ]: # Continue to explore the data to address your additional research
     #   questions. Add more headers as needed if you have more questions to
     #   investigate.
```

```
[ ]:
```

```
[ ]: # Use this, and more code cells, to explore your data. Don't forget to add
     #   Markdown cells to document your observations and findings.
```

1.0.20 Research Question 2 (Replace this header name!)

```
[ ]: # Continue to explore the data to address your additional research
     #   questions. Add more headers as needed if you have more questions to
     #   investigate.
```

1.0.21 Research Question 2 (Replace this header name!)

```
[ ]: # Continue to explore the data to address your additional research
     #   questions. Add more headers as needed if you have more questions to
     #   investigate.
```

1.0.22 Research Question 2 (Replace this header name!)

```
[ ]: # Continue to explore the data to address your additional research
     #   questions. Add more headers as needed if you have more questions to
     #   investigate.
```

1.0.23 Research Question 2 (Replace this header name!)

```
[ ]: # Continue to explore the data to address your additional research
     #   questions. Add more headers as needed if you have more questions to
     #   investigate.
```

Conclusions

Tip: Finally, summarize your findings and the results that have been performed in relation to the question(s) provided at the beginning of the analysis. Summarize the results accurately, and point out where additional research can be done or where additional information could be useful.

Tip: Make sure that you are clear with regards to the limitations of your exploration. You should have at least 1 limitation explained clearly.

Tip: If you haven't done any statistical tests, do not imply any statistical conclusions. And make sure you avoid implying causation from correlation!

Tip: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric (found on the project submission page at the end of the lesson). You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

1.1 Submitting your Project

Tip: Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Tip: Alternatively, you can download this report as .html via the **File** > **Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Tip: Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

```
[ ]: from subprocess import call
     call(['python', '-m', 'nbconvert', 'Investigate_a_Dataset.ipynb', '--to',␣
      ↪'pdf'])
```

[ ]:
```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Investigate_a_Dataset.ipynb', '--to',␣
 ↪'html'])
```

[ ]:
```

```