Project: Investigating the TMDb movie data dataset - [tmdb-movies.csv]

Table of Contents

Introduction

Dataset Description

> In this project we will be analysing the tmdb-movies dataset (tmdb-movies.csv), this data set contains information about 10,000 movies collected from The Movie Database (TMDb) with 21 columns filled with different informations about the movies. This dataset was assembled to answer most of the questions posed by the film industry (is there a consistent formula to predict if the movie will be successful ?

> The column are :
>
> - **id**
> - **imdb_id:** the id of the movie in imdb website
> - **popularity:** cumulative decided by number of star ratings, is a very important metric here on TMDB. It helps us boost search results
> - **budget:** budget of the movie before the inflation
> - **revenue:** revenue of the movie before the inflation
> - **original_title**
> - **cast:** the actors
> - **homepage:** url for the homepage of the movie
> - **director:** director name
> - **tagline:** famous tagline of the movie
> - **keywords:** keywords that describe the movie
> - **overview:** an brief overview of the movie (plot)
> - **runtime:** the length of the film plus the length of the ending credits
> - **genres:** categories that define films based on narrative or stylistic elements
> - **production_companies**

> - **release_date**
> - **vote_count:** number of votes/ratings of the movie on imdb
> - **vote_average:** a vote out of 10 (10 being the highest - 0 is the lowest)
> - **release_year**
> - **budget_adj:** the budget of the associated movie in terms of 2010 dollars, accounting for inflation over time
> - **revenue_adj:** the revenue of the associated movie in terms of 2010 dollars, accounting for inflation over time

Question(s) for Analysis

> Questions asked:
>
> - Which genres are most popular from year to year?
> - What kinds of properties are associated with movies that have high profit?
> - Are movie with higher budget profitable?
> - Do movie with higher budget recieve better ratting?

```
# 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
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Wrangling

> In this section of the report, we will load in the data, check for cleanliness, and then trim and clean your dataset for analysis.

```
# Loading the data

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

	id	imdb_id	popularity	budget	revenue	original_title	ca
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bry Dal Howard|Irrf Khan|V
1	76341	tt1392190	28.419936	150000000	378436354	Mad Max: Fury Road	To Hardy|Charli Theron|Hu Kea Byrne|Ni
2	262500	tt2908446	13.112507	110000000	295238201	Insurgent	Shaile Woodley|Th James|Ka Winslet|Anse
3	140607	tt2488496	11.173104	200000000	2068178225	Star Wars: The Force Awakens	Harris Ford|Ma Hamill|Car Fisher|Adam D
4	168259	tt2820852	9.335014	190000000	1506249360	Furious 7	Vin Diesel|Pa Walker|Jas Statham|Miche

```
#Checking for the numbers of rows:10866 , columns:21
df.shape
```

```
(10866, 21)
```

```
#Cheaking datatypes and missing values
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
```

```
# Identifying the columns with missing values, and the number of rows with missing values
print(df.isnull().sum())
print("\n \n")
print("The name of the columns with missing values: ", df.isnull().sum().index[df.isnull(
print("\n")
missing = df.shape[0] - df.dropna().shape[0]
print("The number of rows with missing values: ", missing)
```

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



The name of the columns with missing values:  Index(['imdb_id', 'cast', 'homepage',
       'overview', 'genres', 'production_companies'],
      dtype='object')


The number of rows with missing values:  8874
```

```
# Further information about our data
df.describe()
```

	id	popularity	budget	revenue	runtime	vote_c
count	10866.000000	10866.000000	1.086600e+04	1.086600e+04	10866.000000	10866.00
mean	66064.177434	0.646441	1.462570e+07	3.982332e+07	102.070863	217.38
std	92130.136561	1.000185	3.091321e+07	1.170035e+08	31.381405	575.61
min	5.000000	0.000065	0.000000e+00	0.000000e+00	0.000000	10.00
25%	10596.250000	0.207583	0.000000e+00	0.000000e+00	90.000000	17.00
50%	20669.000000	0.383856	0.000000e+00	0.000000e+00	99.000000	38.00
75%	75610.000000	0.713817	1.500000e+07	2.400000e+07	111.000000	145.75
max	417859.000000	32.985763	4.250000e+08	2.781506e+09	900.000000	9767.00

> From the describe() function we can see that some columns have missing data in the form of 0.0 as we can se in the rows of min, 25%, 50% of the columns 'runtime', 'budget_adj', 'revenue_adj'

> We can also conclude from the describe() function that there is a presence of outliers in the columns ('popularity', 'budget', 'revenue' 'runtime', 'budget_adj', 'revenue_adj'). We can confirm the presence of the outliers because we can see that there is a big gap between the max and min values in each of the columns

First Conclusion

> after exploring this dataset we noticed that:
>
> - Columns to drop:
>   ["id", "imdb_id", "budget", "revenue", "cast", "production_companies", "release_date", "vote_count","original_title", "homepage", "tagline", "overview"]
> - Missing values that need to be delt with in the columns:
>   ['imdb_id', 'cast', 'homepage', 'director', 'tagline', 'keywords', 'overview', 'genres', 'production_companies']
> - Datatypes are correct for the columns
> - From the describe() function we can see that in the columns:
>   ['budget', 'revenue', 'runtime', 'budget_adj', 'revenue_adj'] have missing values in the form of 0.0 as we can see in the min == 0 and 25% == 0 and 50% == 0
> - From the describe() function we can see the presence of outliers

> Now we start the cleaning of this Dataset

Data Cleaning

Dealling with columns

```
#we start by dropping the unnecessary columns
df.drop(["id", "imdb_id", "budget", "revenue", "cast", "production_companies", "release_da
```

```
#we visualize the new data
df.head()
```

	popularity	director	keywords	runtime	g
0	32.985763	Colin Trevorrow	monster|dna|tyrannosaurus rex|velociraptor|island	124	Action|Adventure|Sc Fiction|T
1	28.419936	George Miller	future|chase|post-apocalyptic|dystopia|australia	120	Action|Adventure|Sc Fiction|T
2	13.112507	Robert Schwentke	based on novel|revolution|dystopia|sequel|dyst...	119	Adventure|Sc Fiction|T
3	11.173104	J.J. Abrams	android|spaceship|jedi|space opera|3d	136	Action|Adventure|Sc Fiction|Fa
4	9.335014	James Wan	car race|speed|revenge|suspense|car	137	Action|Crime|T

Missing values

```
# Identifying the columns with missing values, and the number of rows with missing values
print(df.isnull().sum())
print("\n \n")
print("The name of the columns with missing values: ", df.isnull().sum().index[df.isnull()
print("\n")
missing = df.shape[0] - df.dropna().shape[0]
print("The number of rows with missing values: ", missing)
```

```
popularity         0
director          44
keywords        1493
runtime            0
genres            23
vote_average       0
release_year       0
budget_adj         0
revenue_adj        0
dtype: int64



The name of the columns with missing values:  Index(['director', 'keywords', 'genres
```

```
The number of rows with missing values:  1514
```

```
df.shape
```

```
(10866, 9)
```

```
# For the missing values, we will just drop them because we dont have the info to fill thc
df.dropna(axis=0, inplace=True)
print(df.isnull().sum())
```

```
popularity      0
director        0
keywords        0
runtime         0
genres          0
vote_average    0
release_year    0
budget_adj      0
revenue_adj     0
dtype: int64
```

```
# Now we will deal with the missing values in the form of 0.0
df.describe()
```

	popularity	runtime	vote_average	release_year	budget_adj	revenue_ac
count	9352.000000	9352.000000	9352.000000	9352.000000	9.352000e+03	9.352000e+(
mean	0.705390	103.205731	6.004940	2000.443435	1.982606e+07	5.900329e+(
std	1.061420	28.625692	0.912052	13.068431	3.612686e+07	1.542159e+(
min	0.000188	0.000000	1.500000	1960.000000	0.000000e+00	0.000000e+(
25%	0.229523	91.000000	5.400000	1993.000000	0.000000e+00	0.000000e+(
50%	0.421030	100.000000	6.100000	2005.000000	4.720913e+05	0.000000e+(
75%	0.787676	113.000000	6.600000	2011.000000	2.551349e+07	4.613363e+(
max	32.985763	900.000000	9.200000	2015.000000	4.250000e+08	2.827124e+(

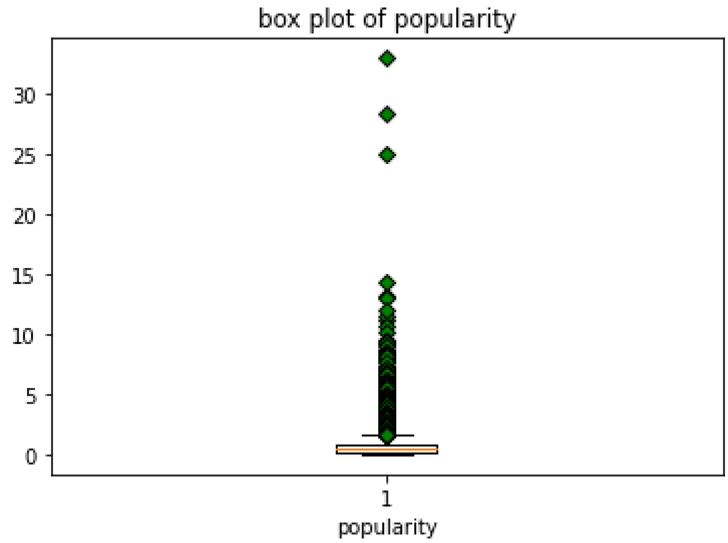
```
# We will change the values of 0.0 with the mean to not lose data in the columns ['runtime
numeric_columns=['runtime', 'budget_adj', 'revenue_adj']
for col in numeric_columns:
  df[col] = df[col].replace(0.0, df[col].mean())
```

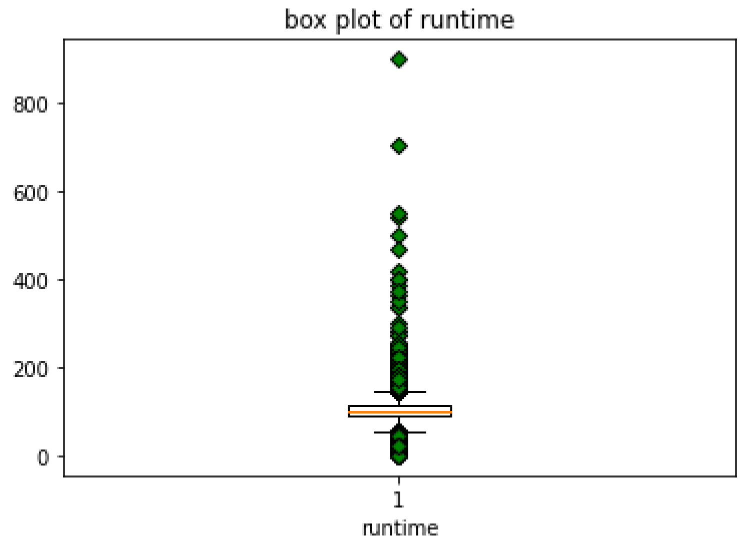
Dealing with outliers

```
# Now as we saw before there is still a presence of outliers in the columns ["popularity",
df.describe()
```

	popularity	runtime	vote_average	release_year	budget_adj	revenue_a(
count	9352.000000	9352.000000	9352.000000	9352.000000	9.352000e+03	9.352000e+(
mean	0.705390	103.349195	6.004940	2000.443435	2.939990e+07	8.888979e+(
std	1.061420	28.365503	0.912052	13.068431	3.199453e+07	1.453467e+(
min	0.000188	2.000000	1.500000	1960.000000	9.210911e-01	2.861934e+(
25%	0.229523	91.000000	5.400000	1993.000000	1.982606e+07	4.906792e+(
50%	0.421030	100.000000	6.100000	2005.000000	1.982606e+07	5.900329e+(
75%	0.787676	113.000000	6.600000	2011.000000	2.551349e+07	5.900329e+(
max	32.985763	900.000000	9.200000	2015.000000	4.250000e+08	2.827124e+(

```
# Need to visualize the presence of the outliers using the box plot
numeric_columns=['popularity', 'runtime', 'budget_adj', 'revenue_adj']
green_diamond = dict(markerfacecolor='g', marker='D')
for col in numeric_columns:
  fig, ax = plt.subplots()
  ax.boxplot(df[col], flierprops=green_diamond)
  ax.set_title("box plot of " + col)
  ax.set_xlabel(col)
```





> for the outliers we can see that the revenue and budget have a wide intervale and we dont have to much data so we can't afford dropping those outliers.

> So we will deal with the popularity and runtime

```
#we define a function to find the outliers
def find_outliers_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   return outliers
```

```
# Now we try to find the outliers for one columns to verify they match the statistics we s
# which helps confirm we calculated the outliers correctly
outliers = find_outliers_IQR(df.runtime)

print("number of outliers: "+ str(len(outliers)))

print("max outlier value: "+ str(outliers.max()))

print("min outlier value: "+ str(outliers.min()))
```

```
number of outliers: 519
max outlier value: 900.0
min outlier value: 2.0
```

```
# We will just drop the outliers
df = df.drop(index= find_outliers_IQR(df["popularity"]).index)
df = df.drop(index= find_outliers_IQR(df["runtime"]).index)
```

```
df.describe()
```

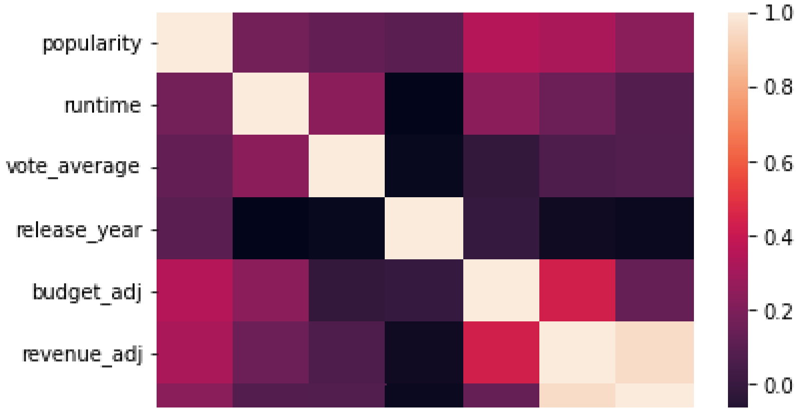
	popularity	runtime	vote_average	release_year	budget_adj	revenue_a(
count	8051.000000	8051.000000	8051.000000	8051.000000	8.051000e+03	8.051000e+(
mean	0.483121	100.837097	5.902844	2000.159608	2.463574e+07	6.485850e+(
std	0.355184	14.295109	0.891060	12.931246	2.281901e+07	7.354172e+(
min	0.000188	59.000000	1.500000	1960.000000	9.210911e-01	2.861934e+(
25%	0.218070	90.000000	5.400000	1993.000000	1.982606e+07	4.015516e+(
50%	0.386180	99.000000	6.000000	2004.000000	1.982606e+07	5.900329e+(
75%	0.658450	110.000000	6.500000	2010.000000	1.982606e+07	5.900329e+(
max	1.624483	142.000000	8.900000	2015.000000	4.250000e+08	1.583050e+(

```
# Next, we create a new column to easaly manipulate the genres
df["new genres"] = df.genres.str.split('|')
```

```
# We create a profit columns
df["profit"] = df["revenue_adj"] - df["budget_adj"]
```

```
# And we just check the correlation of the column
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb98b38b9a0>
```



> we conclude that there is coorelation between popularityand the budget, and the budget and revenue

Exploratory Data Analysis

Which genres are most popular from year to year?

Exploring the data

```
df.head()
```

	popularity	director	keywords	runtime	ger
104	1.532997	Doug Ellin	friendship|hollywood|movie star|entourage	104.0	Con
105	1.510096	Jeremy Garelick	male friendship|impersonator|wedding|lying|bes...	101.0	Con
106	1.499614	Christopher B. Landon	female nudity|shotgun|nudity|strip club|party	93.0	Comedy|Hc
107	1.495112	M. Night Shyamalan	rap|pennsylvania|brother sister relationship|f...	94.0	Horror|Th
108	1.483246	Scott Cooper	boston|based on true story|organized crime	122.0	Crime|Dr

```
# First we checked the genres
df.genres.value_counts()
```

```
Drama                                            546
Comedy                                           536
Comedy|Drama                                     240
Drama|Romance                                    236
Horror|Thriller                                  220
                                                ...
Adventure|Science Fiction|Thriller|Mystery         1
Thriller|Comedy|Action                             1
Comedy|Fantasy|Thriller                            1
Comedy|Romance|Science Fiction|Drama               1
Mystery|Comedy                                     1
Name: genres, Length: 1629, dtype: int64
```

Function definition

```
# Third we define two functions
# getGenres_without_duplicates to get all the unique genres thar are in the dataset
def getGenres_without_duplicates(data):
    a = set()
    for i in data:
        for j in i:
            a.add(j)

    b = np.array(list(a))
    return b.reshape(b.shape[0], 1)

# The seconde function collecting_genres is to get all the genres (with duplicates) of an
def collecting_genres(data):
    a = []
    for i in data:
        for j in i:
            a.append(j)
    return a
```

Creating additional dataframes

```
# Now we create a dataframe "s" that contains all the unique genres and a column called cc
# this dataframe will be used to callculate the count of each genres in the selected year
# for example to count all the unique genres that appear in the films releases in 2015

#first we get all the unique genres in the dataset
data = getGenres_without_duplicates(df["new genres"])

#seconde we create a np array for the seconde column "count"
longeur = data.shape[0]
k = np.zeros((longeur, 1))

#and last, we create the dataframe
real_data = np.concatenate((data, k), axis=1)
s = pd.DataFrame(real_data, columns=["type", "count"])
```

```
#and we just change the type of count to float so that we can operate on it
s["count"] = s["count"].astype('float')
s
```

	type	count
0	Science Fiction	0.0
1	Western	0.0
2	Action	0.0
3	Fantasy	0.0
4	Drama	0.0
5	History	0.0
6	TV Movie	0.0
7	War	0.0
8	Mystery	0.0
9	Adventure	0.0
10	Foreign	0.0
11	Family	0.0
12	Music	0.0
13	Horror	0.0
14	Romance	0.0
15	Crime	0.0
16	Thriller	0.0
17	Documentary	0.0
18	Animation	0.0
19	Comedy	0.0

```
# Here we get all the unique years found in the dataset
years = df.release_year.value_counts().index
years
```

```
Int64Index([2014, 2013, 2009, 2012, 2008, 2011, 2015, 2007, 2006, 2010, 2005,
            2004, 2002, 2003, 2000, 2001, 1999, 1998, 1996, 1997, 1994, 1993,
            1995, 1988, 1990, 1992, 1991, 1986, 1989, 1987, 1984, 1985, 1980,
            1982, 1983, 1981, 1978, 1973, 1971, 1977, 1979, 1976, 1966, 1974,
            1975, 1972, 1964, 1967, 1970, 1968, 1963, 1965, 1960, 1961, 1962,
            1969],
           dtype='int64')
```

```
# just as the "s" table here we create a "best_genres_perYear" dataframe to store the best
```

```
#first we sort the years from the "years" table
years_sorted = np.array(years.sort_values())
years_sorted = years_sorted.reshape(years_sorted.shape[0], 1)

#seconde we create a np array for the seconde column "genres"
k = np.zeros((len(years), 1))

#and last, we create the dataframe
real_data = np.concatenate((years_sorted, k), axis=1)
best_genres_perYear = pd.DataFrame(real_data, columns=["year", "genres"])
best_genres_perYear
```

	year	genres
0	1960.0	0.0
1	1961.0	0.0
2	1962.0	0.0
3	1963.0	0.0
4	1964.0	0.0
5	1965.0	0.0
6	1966.0	0.0
7	1967.0	0.0
8	1968.0	0.0
9	1969.0	0.0
10	1970.0	0.0
11	1971.0	0.0
12	1972.0	0.0
13	1973.0	0.0
14	1974.0	0.0
15	1975.0	0.0
16	1976.0	0.0
17	1977.0	0.0
18	1978.0	0.0
19	1979.0	0.0
20	1980.0	0.0
21	1981.0	0.0
22	1982.0	0.0
23	1983.0	0.0
24	1984.0	0.0
25	1985.0	0.0
26	1986.0	0.0
27	1987.0	0.0
28	1988.0	0.0
29	1989.0	0.0

Finding the best genres of each year

```
# Here is where all the magic happens

# We start by initializing a variable "final_genres" which will hold the best geners of th
final_genres=""

#  And then we iterate for each year in the "years" dataframe
for i in years:

  # Firstly, we select the data of the selected year "i" from our dataset
  selected_data = df[df.release_year == i]
  # Next we get the list with all of the genres of that year (with duplicates) using the c
  genre_list = collecting_genres(selected_data["new genres"])

  # Now, we update the count in the "s" dataframe for each genre
  for j in genre_list:
    s.loc[s["type"] == j , "count"] += 1

  # After that, we find the max count in the "s" dataframe to identify the best genres
  best_genre_count = s["count"].max()
  # And we fetch those best genres from the "s" dataframe
  bests = s[s["count"] == best_genre_count]["type"]

  # Here we concatinate the best genres in one variable "final_genres"
  for l in bests:
   final_genres = final_genres + " " + l

  # Lastly, we update the dataframe "best_genres_perYear"
  best_genres_perYear.loc[best_genres_perYear.year == i, "genres"] = final_genres

  # At the end we empty the "final_genres" string
  final_genres=""
  # And we empty the count column of the "s" dataframe for the next iteration
  s["count"] = np.zeros(s.shape[0])

best_genres_perYear
```

	year	genres
0	1960.0	Drama Comedy
1	1961.0	Drama
2	1962.0	Drama
3	1963.0	Comedy
4	1964.0	Drama
5	1965.0	Drama
6	1966.0	Comedy
7	1967.0	Drama
8	1968.0	Drama
9	1969.0	Drama
10	1970.0	Drama
11	1971.0	Drama
12	1972.0	Drama
13	1973.0	Drama
14	1974.0	Thriller
15	1975.0	Drama
16	1976.0	Drama
17	1977.0	Drama
18	1978.0	Drama
19	1979.0	Drama
20	1980.0	Drama
21	1981.0	Drama
22	1982.0	Comedy
23	1983.0	Drama
24	1984.0	Drama Comedy
25	1985.0	Comedy
26	1986.0	Drama
27	1987.0	Comedy
28	1988.0	Comedy
29	1989.0	Comedy
30	1990.0	Drama
31	1991.0	Comedy

32 1992.0 Drama

> From the dataframe above we can see each year with it respective best genres of the year

35 1995.0 Drama

What kinds of properties are associated with movies that have high profit?

```
# Now we slice the data into to categories "profitable" and "non_profitable"
profitable = df[df["profit"]  > 0]
non_profitable = df[df["profit"]  <= 0]

print(profitable.shape)
print(non_profitable.shape)
```

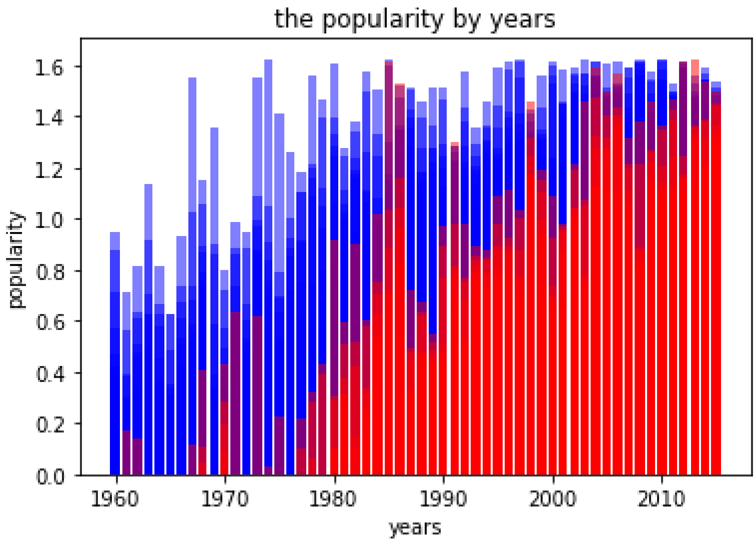
```
(6532, 11)
(1519, 11)
```

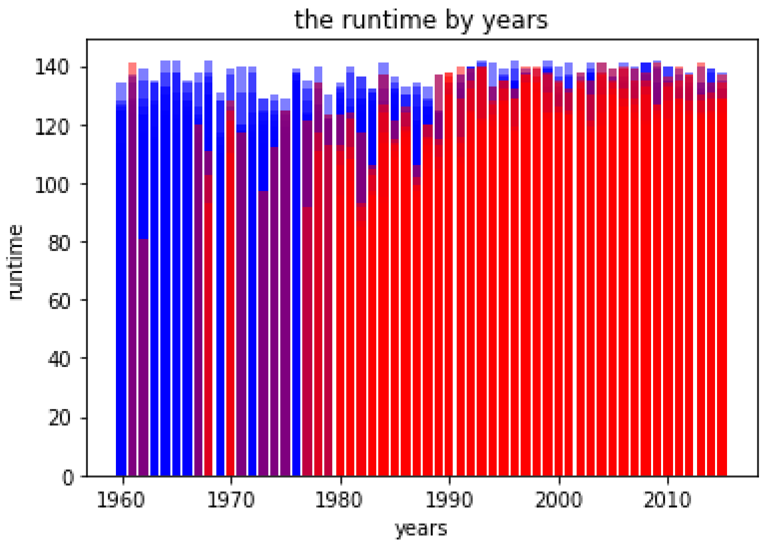
```
# we see what numerical columns to use
numerical = df.select_dtypes(include='number').columns
numerical
```

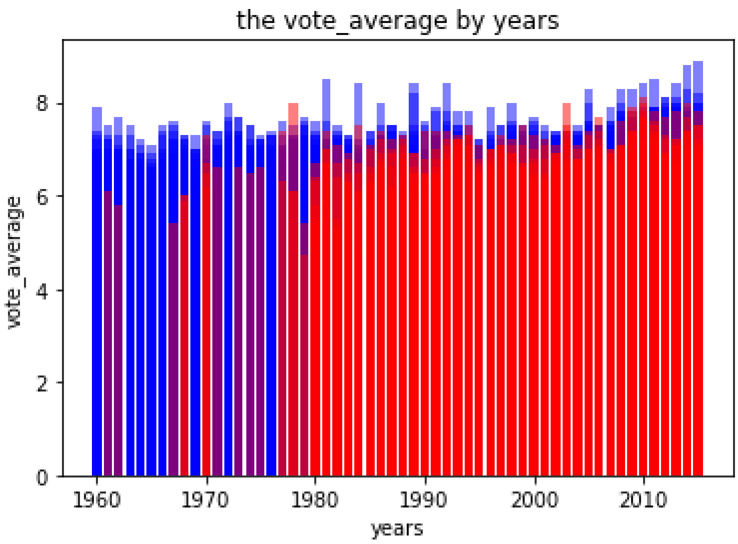
```
Index(['popularity', 'runtime', 'vote_average', 'release_year', 'budget_adj',
       'revenue_adj', 'profit'],
      dtype='object')
```

50 2010.0 Drama

```
# We draw box plot of selected columns by years
investigate = ['popularity', 'runtime', 'vote_average', 'budget_adj']
for col in investigate:
  fig, ax = plt.subplots()
  ax.bar(profitable.release_year, profitable[col], color="blue", alpha=0.5)
  ax.bar(non_profitable.release_year, non_profitable[col], color="red", alpha=0.5)
  ax.set_xlabel("years")
  ax.set_ylabel(col)
  ax.set_title("the " + col + " by years")
```







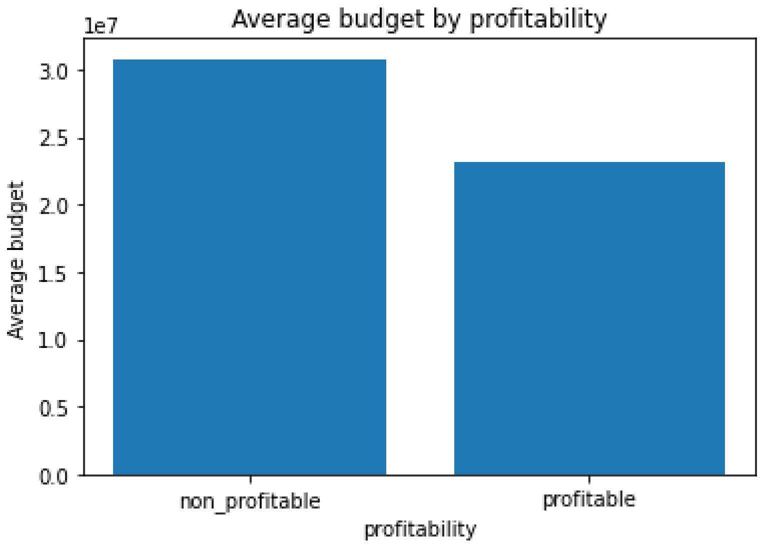
> We arrive to the conclusion that :
>
> - the most profitable films are popular
> - the non profitable films have lower budget than the profitable movies
> - the runtime is high in the nonprofitable film (after the year 1970 in the plot because of a short dataset)

> - the votes are higher for the profitable movies
>
>   we can tell that popularity, budget, vote_average play a role in the profitability of the film

Are movie with higher budget profitable?

```
# here we get the average budget for the profitable and non_profitable movies
mean_profitable_budget = profitable.budget_adj.mean()
mean_non_profitable_budget = non_profitable.budget_adj.mean()


# Create a bar chart to see the results
locs = [1, 2]
heights = [mean_non_profitable_budget, mean_profitable_budget]
labels = ['non_profitable', 'profitable']
plt.bar(locs, heights, tick_label=labels)
plt.title('Average budget by profitability')
plt.xlabel('profitability')
plt.ylabel('Average budget');
```



> we conclude that the non-profitable movies have higher average budgets

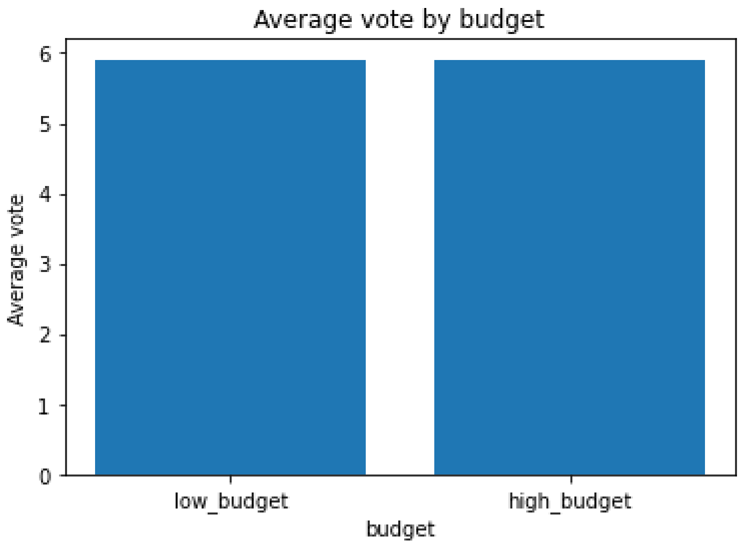
Do movie with higher budget recieve better ratting?

```
# here we need to find the mediane (middle budget)
middle = df.budget_adj.median()

# and then we split to two dataframes "low_budget" and "high_budget"
low_budget = df[df.budget_adj < middle]
high_budget = df[df.budget_adj >= middle]
```

```
# and then we calculate the average vote for the two dataframes
mean_low_vote = low_budget.vote_average.mean()
mean_high_vote = high_budget.vote_average.mean()

# Create a bar chart to see the results
locs = [1, 2]
heights = [mean_low_vote, mean_high_vote]
labels = ['low_budget', 'high_budget']
plt.bar(locs, heights, tick_label=labels)
plt.title('Average vote by budget')
plt.xlabel('budget')
plt.ylabel('Average vote');
```



we see that movies with higher budget recieves an average rating almost equal to movies with low budget

Conclusions

> During this analysis of this dataset we have arrived to a better understanding of what makes a movie have a high chance of it being profitable. here we present our findings:

- As we saw in our first question (Which genres are most popular from year to year?) that the drama genres is present as the most popular genre from year to year
- for the seconde question (What kinds of properties are associated with movies that have high revenues?) we found that usally movies with higher runtime are non_profitable, votes and popularity of the movies indicates its profitability and profitable movie usually have a higher budget
- in the third question (Are movie with higher budget profitable?) we found that no, a higher budget doesn't mean the movie will be profitable.
- Do movie with higher budget recieve better rating? we saw that the movies with higher budget recieve the same rating as the low budget ones on average

> to summarize our findings, if you want a profitable movie we need to create a drama movie with a high budget and a small runtime (this is the secret formula that we arrived to)

> In this analysis we faced some limitation which are:

- The missing data (in the form of Nan values or 0.0
- the dataset isn't diverse enough in the genres(almost every movie is in the drama genre)
- the size of the dataset (10866) isn't enough to find the ultimate secret formula to produce a garantied profitable movie

✓ 0 s terminée à 20:51