TMDb Movie Data Analysis

Table of Contents

- Introduction
- Data Wrangling
- Exploratory Data Analysis
- Conclusions

Introduction

Dataset Description

In this project we will be analyzing a dataset from The Movie Database (TMDb) that contains information about more than 10,000 movies. we are interested in finding trends and patterns between different genres, popularity, budgets and the effect of these factors on revenues.

This dataset has the following 21 columns with 10866 rows:

- id
- imdb_id
- popularity
- budget
- revenue
- original_title
- cast
- homepage
- director
- tagline
- keywords
- overview
- runtime
- genres
- production_companies
- release_date
- vote_count
- vote_average
- release_year
- budget_adj
- revenue_adj

Question(s) for Analysis

Average Runtime by Genre
Average Runtime By Most liked Genres
Is there a relation between Runtime and Average Vote

In [61]:

```python
#Import panda,numpy, matplotlib
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Data Wrangling

General Properties

- **53%** of the data have possibly wrong values for the budget and revenue columns (where budget less than 5,000 usd). the data must be cleaned and stored in a seperate dataframe before answering questions related to revenue
- Column labels and data types are good, there is no need to change anything.
- Columns **(imdb_id, cast, homepage, director, tagline, keywords, overview, production_companies)** are **irrelevant** for this analysis and will be drop from the dataset.
- There are 23 missing data rows in the column **genres** and after going through them, I decided to drop them due to bad quality
- **genres** are collected into one column and delimited by **colon**, in order to to analyse different genres we need to be split them into seperate rows for the same rows
- only one duplicated data found

In [5]:
```
# This cell is to read from the dataset.
df_movies = pd.read_csv('tmdb-movies.csv')

# To show a sample of the data.
df_movies.head(1)
```

Out[5]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage	director	tagline	...	overview	ru
0	135397	tt0369610	32.985763	150000000	1513528810	Jurassic World	Chris Pratt|Bryce Dallas Howard|Irrfan Khan|Vi...	http://www.jurassicworld.com/	Colin Trevorrow	The park is open.	...	Twenty-two years after the events of Jurassic ...	

1 rows × 21 columns

The only significant column with missing data are in the **genres** column, most of the rest will be dropped.
Column names and data types are good.

In [6]:
```
# The info method to list column names, check data types and missing data
df_movies.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 
```

By looking information from the describe method
50% of the values in **budget**, **revenue** and **popularity** is **0** which indicates missing values thus further investigation required

In [7]:
```
df_movies.describe().astype(float).round()
```

Out[7]:

	id	popularity	budget	revenue	runtime	vote_count	vote_average	release_year	budget_adj	revenue_adj
count	10866.0	10866.0	10866.0	1.086600e+04	10866.0	10866.0	10866.0	10866.0	10866.0	1.086600e+04
mean	66064.0	1.0	14625701.0	3.982332e+07	102.0	217.0	6.0	2001.0	17551040.0	5.136436e+07
std	92130.0	1.0	30913214.0	1.170035e+08	31.0	576.0	1.0	13.0	34306156.0	1.446325e+08
min	5.0	0.0	0.0	0.000000e+00	0.0	10.0	2.0	1960.0	0.0	0.000000e+00
25%	10596.0	0.0	0.0	0.000000e+00	90.0	17.0	5.0	1995.0	0.0	0.000000e+00
50%	20669.0	0.0	0.0	0.000000e+00	99.0	38.0	6.0	2006.0	0.0	0.000000e+00
75%	75610.0	1.0	15000000.0	2.400000e+07	111.0	146.0	7.0	2011.0	20853251.0	3.369710e+07
max	417859.0	33.0	425000000.0	2.781506e+09	900.0	9767.0	9.0	2015.0	425000000.0	2.827124e+09

53% of the data has **budget** lower than $5,000

In [8]:
```
# returns budgets that are less than 5000 and devide it by number of rows
(df_movies.query('budget < 5000').budget.count()/df_movies.budget.count())
```

Out[8]:
```
0.5298177802319161
```

Only one duplicated row found

In [9]:
```
# Check for duplicated data
df_movies.duplicated().sum()
```

Out[9]:
```
1
```

Columns **(imdb_id, cast, homepage, director, tagline, keywords, overview)** are irrelevant for this analysis and needs to dropped from the dataset.

There are 23 missing data rows in genres and after going through I think it is better to drop them for the first question due to their bad quality

In [10]:

```
#isnull to identify missing rows in the genres column
df_movies[df_movies.genres.isnull()]
```

Out[10]:

	id	imdb_id	popularity	budget	revenue	original_title	cast	homepage	c
424	363869	tt4835298	0.244648	0	0	Belli di papÃ	Diego Abatantuono|Matilde Gioli|Andrea Pisani|...	NaN	Guido
620	361043	tt5022680	0.129696	0	0	All Hallows' Eve 2	NaN	NaN	Padova Norto Roussel
997	287663	NaN	0.330431	0	0	Star Wars Rebels: Spark of Rebellion	Freddie Prinze Jr.|Vanessa Marshall|Steve Blum...	NaN	S Lee|St
						Prayers for	Ryan Kelley|Sigourney		

Data Cleaning

First, I dropped unwanted columns:
imdb_id, cast, homepage, director, tagline, keywords, overview

In [11]:

```
# drop unwanted columns
df_movies.drop(['imdb_id', 'cast', 'homepage', 'director', 'tagline', 'keywords', 'overview','production_companies'], ax

#check columns after drop
df_movies.head(1)
```

Out[11]:

	id	popularity	budget	revenue	original_title	runtime	genres	release_date	vote_count	vote_average	release_year
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Action|Adventure|Science Fiction|Thriller	6/9/15	5562	6.5	2015

Then dropped duplicated values

In [12]:

```
# drop duplicated value
df_movies.drop_duplicates(inplace=True)

# check if it worked
df_movies.duplicated().sum()
```

Out[12]:

```
0
```

Finally, I dropped rows with missing genre values

In [13]:

```
# drop rows with missing genres value
df_movies.dropna(subset=['genres'], inplace=True)

# check number of values
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10842 entries, 0 to 10865
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              10842 non-null  int64  
 1   popularity      10842 non-null  float64
 2   budget          10842 non-null  int64  
 3   revenue         10842 non-null  int64  
 4   original_title  10842 non-null  object 
 5   runtime         10842 non-null  int64  
 6   genres          10842 non-null  object 
 7   release_date    10842 non-null  object 
 8   vote_count      10842 non-null  int64  
 9   vote_average    10842 non-null  float64
 10  release_year    10842 non-null  int64  
 11  budget_adj      10842 non-null  float64
 12  revenue_adj     10842 non-null  float64
dtypes: float64(4), int64(6), object(3)
memory usage: 1.2+ MB
```

Now we create a new dataframe to split different genres for the same movie into seperate columns by their delimitar using the split and explode methods

In [14]:

```
# split genres into a group within the same column
df_movies['genres'] = df_movies['genres'].str.split('|')
```

In [15]:

```
df_movies.head(1)
```

Out[15]:

	id	popularity	budget	revenue	original_title	runtime	genres	release_date	vote_count	vote_average	release_year	budget_adj
0	135397	32.985763	150000000	1513528810	Jurassic World	124	[Action, Adventure, Science Fiction, Thriller]	6/9/15	5562	6.5	2015	1.379999e+08

In [16]:

```
# Creates new rows for every genre for the same row
df_movies_genres = df_movies.explode('genres')
```

In [24]:

```
# check if it worked
df_movies_genres.query("original_title == 'Jurassic World'")
```

Out[24]:

	id	popularity	budget	revenue	original_title	runtime	genres	release_date	vote_count	vote_average	release_year	budget_adj
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Action	6/9/15	5562	6.5	2015	1.379999e+08
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Adventure	6/9/15	5562	6.5	2015	1.379999e+08
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Science Fiction	6/9/15	5562	6.5	2015	1.379999e+08
0	135397	32.985763	150000000	1513528810	Jurassic World	124	Thriller	6/9/15	5562	6.5	2015	1.379999e+08

We need to create a new dataframe for revenue related analysis which will exclude data with budget or revenue less than $5,000

In [52]:

```
# create new dataframe
df_movies_revenue = df_movies.query('budget >= 5000')
```

Exploratory Data Analysis

Average runtime per genre?

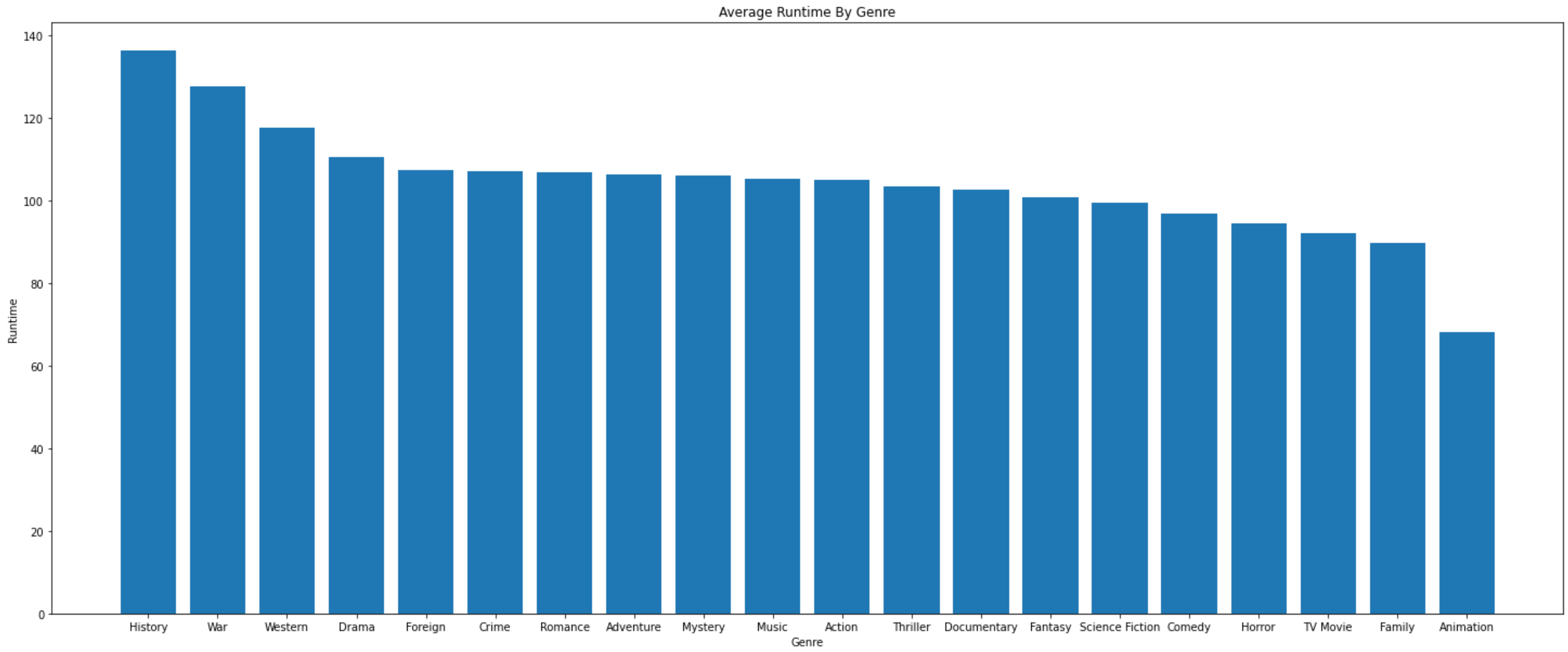
First we create genres_runtime_mean variable the contains the mean of runtime for each genre sorted in desc

In [214]:

```
runtime_mean = df_movies_genres.groupby('genres').runtime.mean().sort_values(ascending=False).to_dict()
```

In [216]:

```
#width = runtime_mean.to_dict.keys()
plt.figure(figsize=(25,10))
plt.bar(runtime_mean.keys(), runtime_mean.values())
plt.title('Average Runtime By Genre')
plt.xlabel('Genre')
plt.ylabel('Runtime');
```



Now we need to find the best five Genres

In [219]:

```
# Create a dictionary with last 5 values of vote average mean ordered descinding
genre_vote_avg = df_movies_genres.groupby('genres').vote_average.mean().sort_values().tail(5).to_dict()
```

In [366]:

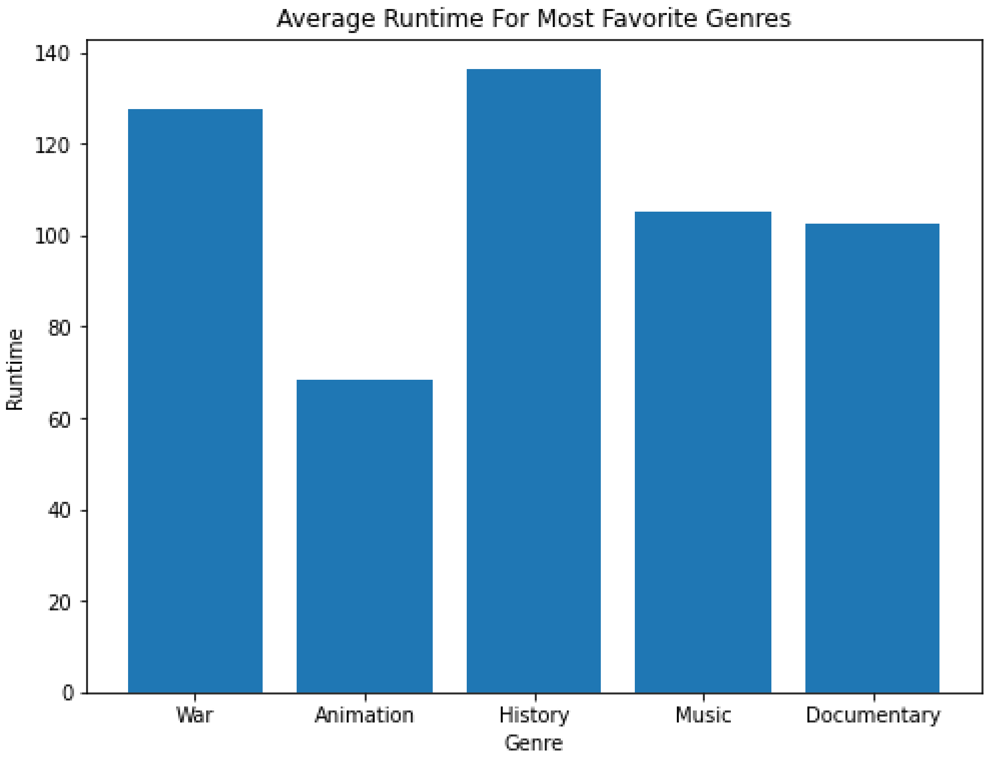
```
# match each best 5 genre with it's runtime
best_runtime = {}

for i in genre_vote_avg:
    if i in runtime_mean:
        best_runtime[i]= runtime_mean[i]
```

Average Runtime By Most liked Genres?

Now we need to plot the best five genres and their runtime

In [227]:

```
plt.figure(figsize=(8,6))
plt.bar(best_runtime.keys(), best_runtime.values())
plt.title('Average Runtime For Most Favorite Genres')
plt.xlabel('Genre')
plt.ylabel('Runtime');
```



Is there a relation between Runtime and Average Vote

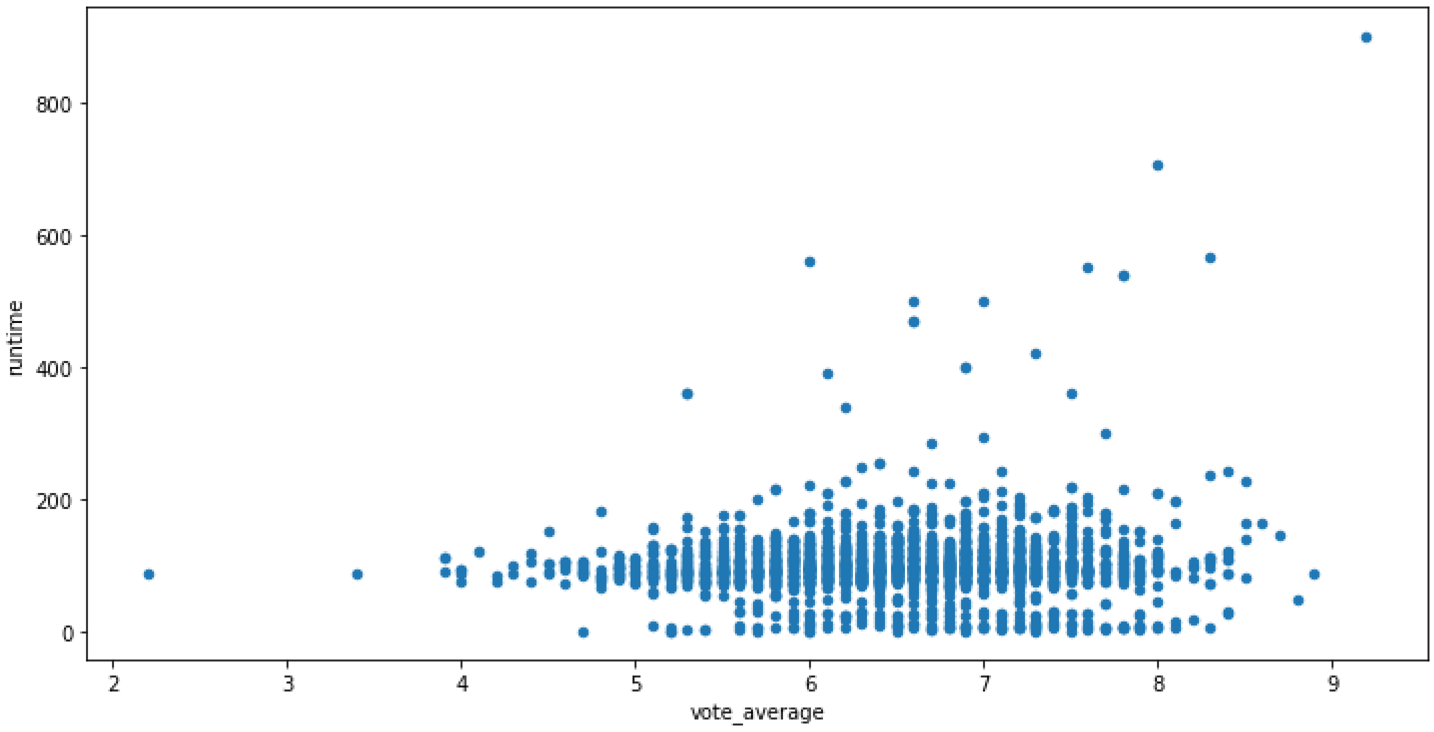
We need to use scatter plot for a new filtered dataframe that only includes the best genres

In [360]:

```
# this will filter the data for only the highest vote genres
df_best_five = df_movies_genres.query("genres in ['War', 'Animation', 'History', 'Music', 'Documentary']")
```

In [361]:

```
# This will scatter plot highest vote genres runtime and average vote
df_best_five.plot(y='runtime', x='vote_average', kind='scatter',figsize=(12,6));
```



Conclusions

- First graph shows the average runtime for every genre, where History is the highest at 136 minutes and animation the lowet at 68 minutes

- Most favorite **Genres are History 136m, War 127m, Music 105m, Documentry 102m, Animation 68m**
- There is a relation between average vote and runtime as movies that run at armound 100 minutes gets the most votes
- Most movies runtime is around that time between 100 and 110 minutes

Limitations

The data for revenue and budget dose not feel accurate and needs further investigation which is beyond the scope of this analysis.

In [368]:

```
from subprocess import call
call(['python', '-m', 'nbconvert', 'Investigate_a_Dataset.ipynb'])
```

Out[368]:

```
1
```