1. Carregamento dos Dados

Nesta etapa, foram importadas as duas principais fontes de dados que servirão para a integração:

- **IMDb (5000 filmes)**: arquivo `movie_metadata.csv`, que contém informações detalhadas como título, ano, duração, gêneros, nota do IMDb, palavras-chave e elenco principal.
- **Base do Desafio (999 filmes)**: arquivo `df_eda01.csv`, que já havia passado por um processo inicial de tratamento no notebook anterior.

In [449…

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
import string

imdb = pd.read_csv('../data/raw/movie_metadata.csv')
base_desafio =  pd.read_csv('../data/processed/df_eda01.csv')
```

2. Exploração Inicial do Dataset IMDb & Junção com a base forencida no desafio!

Antes da integração, foi feita uma análise preliminar da base de 5000 filmes para compreender a completude dos dados e a estrutura das colunas.

- **Verificação de valores nulos**.
- **Visualização da coluna de gêneros (`genres`)**.

Essa análise mostrou que alguns campos possuem valores ausentes, e que os gêneros vêm no formato de string separados por `|`.

In [450…

```
imdb.isna().sum()
```

Out[450…
```
color                       19
director_name              104
num_critic_for_reviews      50
duration                    15
director_facebook_likes    104
actor_3_facebook_likes      23
actor_2_name                13
actor_1_facebook_likes       7
gross                      884
genres                       0
actor_1_name                 7
movie_title                  0
num_voted_users              0
cast_total_facebook_likes    0
actor_3_name                23
facenumber_in_poster        13
plot_keywords              153
movie_imdb_link              0
num_user_for_reviews        21
language                    14
country                      5
content_rating             303
budget                     492
title_year                 108
actor_2_facebook_likes      13
imdb_score                   0
aspect_ratio               329
movie_facebook_likes         0
dtype: int64
```

In [451…
```
imdb['genres']
```

Out[451…
```
0       Action|Adventure|Fantasy|Sci-Fi
1              Action|Adventure|Fantasy
2             Action|Adventure|Thriller
3                       Action|Thriller
4                           Documentary
                    ...
5038                       Comedy|Drama
5039       Crime|Drama|Mystery|Thriller
5040              Drama|Horror|Thriller
5041               Comedy|Drama|Romance
5042                        Documentary
Name: genres, Length: 5043, dtype: object
```

Foi realizada uma análise exploratória para identificar todos os gêneros presentes no dataset IMDb (5000 filmes).

O objetivo foi obter a lista de categorias únicas e comparar com os gêneros existentes na base inicial do desafio (999 filmes).

In [452…
```
lista_generos = []

# Percorre cada valor da coluna 'Genre'
for generos in imdb['genres']:
    # Divide a string em gêneros individuais
    for g in generos.split('|'):
        g = g.strip()  # remove espaços extras
        if g not in lista_generos:
            lista_generos.append(g)
```

```
print('total de generos distintos: ', len(lista_generos))
print(lista_generos)
```

```
total de generos distintos:  26
['Action', 'Adventure', 'Fantasy', 'Sci-Fi', 'Thriller', 'Documentary', 'Romance', 'Animation',
'Comedy', 'Family', 'Musical', 'Mystery', 'Western', 'Drama', 'History', 'Sport', 'Crime', 'Horro
r', 'War', 'Biography', 'Music', 'Game-Show', 'Reality-TV', 'News', 'Short', 'Film-Noir']
```

Após identificar os gêneros distintos, foi criada uma nova coluna `Genres_list` no dataset IMDb.
Essa coluna armazena os gêneros de cada filme no formato de lista, em vez de manter apenas uma string separada por `|`.

Em seguida:

1. A lista de gêneros foi **"explodida"**, de forma que cada filme pudesse aparecer em várias linhas (uma para cada gênero).
2. Criou-se uma **tabela binária (dummies)** indicando, com valores `0` e `1`, a presença de cada gênero por filme.
3. Essa tabela foi integrada de volta ao dataframe original, permitindo que cada gênero passasse a ser tratado como uma coluna independente.

In [453…
```
imdb["Genres_list"] = (
    imdb["genres"].fillna("")
      .apply(lambda s: [g.strip() for g in s.split("|") if g.strip() != ""])
)
```

In [454…
```
tmp = imdb[["movie_title", "Genres_list"]].explode("Genres_list")

# Tabela binária (filme x gênero)
genre_dummies = pd.crosstab(tmp["movie_title"], tmp["Genres_list"]).astype(int)
genre_dummies.columns.name = None  # só estética

# Anexar ao df (sem risco de duplicar nomes)
imdb = imdb.merge(genre_dummies, left_on="movie_title", right_index=True, how="left")
```

Novas Colunas!!

In [455…
```
imdb.columns
```

Out[455…
```
Index(['color', 'director_name', 'num_critic_for_reviews', 'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes', 'Genres_list',
       'Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Family', 'Fantasy', 'Film-Noir', 'Game-Show',
       'History', 'Horror', 'Music', 'Musical', 'Mystery', 'News',
       'Reality-TV', 'Romance', 'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War',
       'Western'],
      dtype='object')
```

Foi feita uma checagem entre colunas que representavam os atores principais, mas com nomes diferentes em cada dataset:

- `Star1` (base desafio) vs. `actor_1_name` (IMDb) → **4387 correspondências**

- `Star2` (base desafio) vs. `actor_2_name` (IMDb) → **839 correspondências**

Essa análise confirmou que, apesar da diferença nos nomes das colunas, ambas armazenam a mesma informação.

In [456…
```
Contador1 = 0
Contador2 = 0
for i in  base_desafio['Star1']:
    for n in imdb['actor_1_name']:
        if i == n:
            Contador1+=1
for i in  base_desafio['Star2']:
    for n in imdb['actor_2_name']:
        if i == n:
            Contador2+=1
print(Contador1)
print(Contador2)
```

```
4387
839
```

Para padronizar a coluna de descrições (`Overview`), foi aplicada uma etapa de pré-processamento de texto:

- Download e uso das *stopwords* em inglês.
- Remoção de pontuações.
- Tokenização das frases em palavras.
- Exclusão de *stopwords*.
- Reconstrução do texto filtrado.

In [457…
```
# Baixar stopwords (só precisa uma vez)
nltk.download('stopwords')

# Definir stopwords em inglês
stop_words = set(stopwords.words('english'))

# Função para limpar o texto
def clean_text(text):
    if pd.isna(text):
        return text
    # remover pontuação
    text = text.translate(str.maketrans('', '', string.punctuation))
    # tokenizar
    words = text.split()
    # remover stopwords
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return " ".join(filtered_words)

# Substituir a coluna Overview diretamente
base_desafio["Overview"] = base_desafio["Overview"].apply(clean_text)
base_desafio
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\guima\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[457…

	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_sc
0	The Godfather	1972.0	A	175 min	Crime, Drama	9.2	organized crime dynastys aging patriarch trans...	1(
1	The Dark Knight	2008.0	UA	152 min	Action, Crime, Drama	9.0	menace known Joker wreaks havoc chaos people G...	
2	The Godfather: Part II	1974.0	A	202 min	Crime, Drama	9.0	early life career Vito Corleone 1920s New York...	
3	12 Angry Men	1957.0	U	96 min	Crime, Drama	9.0	jury holdout attempts prevent miscarriage just...	
4	The Lord of the Rings: The Return of the King	2003.0	U	201 min	Action, Adventure, Drama	8.9	Gandalf Aragorn lead World Men Saurons army dr...	
...	...	...	...	...	...	...	...	
994	Breakfast at Tiffany's	1961.0	A	115 min	Comedy, Drama, Romance	7.6	young New York socialite becomes interested yo...	
995	Giant	1956.0	G	201 min	Drama, Western	7.6	Sprawling epic covering life Texas cattle ranc...	
996	From Here to Eternity	1953.0	Passed	118 min	Drama, Romance, War	7.6	Hawaii 1941 private cruelly punished boxing un...	
997	Lifeboat	1944.0	NaN	97 min	Drama, War	7.6	Several survivors torpedoed merchant ship Worl...	
998	The 39 Steps	1935.0	NaN	86 min	Crime, Mystery, Thriller	7.6	man London tries help counterespionage Agent A...	

999 rows × 39 columns

Remoção de colunas irrelevantes do dataset:

In [458…

```
cols_facebook = [
    "director_facebook_likes",
    "actor_3_facebook_likes",
    "actor_1_facebook_likes",
    "cast_total_facebook_likes",
    "actor_2_facebook_likes",
    "movie_facebook_likes"
```

```
]

imdb = imdb.drop(columns=cols_facebook)
```

Verificação se as colunas representam a mesma coisa:

In [459…
```
imdb['content_rating'].value_counts()
```

Out[459…
```
content_rating
R            2118
PG-13        1461
PG            701
Not Rated     116
G             112
Unrated        62
Approved       55
TV-14          30
TV-MA          20
TV-PG          13
X              13
TV-G           10
Passed          9
NC-17           7
GP              6
M               5
TV-Y            1
TV-Y7           1
Name: count, dtype: int64
```

In [460…
```
base_desafio["Certificate"].value_counts()
```

Out[460…
```
Certificate
U           234
A           196
UA          175
R           146
PG-13        43
PG           37
Passed       34
G            12
Approved     11
TV-PG         3
GP            2
TV-14         1
Unrated       1
TV-MA         1
16            1
U/A           1
Name: count, dtype: int64
```

Mapeamento do nome das colunas

In [461…
```
mapa_colunas = {
    "movie_title": "Series_Title",
    "title_year": "Released_Year",
    "content_rating": "Certificate",
    "duration": "Runtime",
    "genres": "Genre",
    "imdb_score": "IMDB_Rating",
    "num_voted_users": "No_of_Votes",
    "gross": "Gross",
```

```
    "director_name": "Director",
    "actor_1_name": "Star1",
    "actor_2_name": "Star2",
    "actor_3_name": "Star3",
    # Não existe no dataset de 5000 mas existe no de 1000
    # "Star4": ???  # ficaria sem equivalente
    "plot_keywords": "Overview"
}
imdb = imdb.rename(columns=mapa_colunas)
```

In [462…
```
imdb.columns
```

Out[462…
```
Index(['color', 'Director', 'num_critic_for_reviews', 'Runtime', 'Star2',
       'Gross', 'Genre', 'Star1', 'Series_Title', 'No_of_Votes', 'Star3',
       'facenumber_in_poster', 'Overview', 'movie_imdb_link',
       'num_user_for_reviews', 'language', 'country', 'Certificate', 'budget',
       'Released_Year', 'IMDB_Rating', 'aspect_ratio', 'Genres_list', 'Action',
       'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Documentary',
       'Drama', 'Family', 'Fantasy', 'Film-Noir', 'Game-Show', 'History',
       'Horror', 'Music', 'Musical', 'Mystery', 'News', 'Reality-TV',
       'Romance', 'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War', 'Western'],
      dtype='object')
```

Remoção de mais colunas irrelevantes ou redundantes:

In [463…
```
cols_extra_sem_equivalente = [
    "color",
    "num_critic_for_reviews",
    "facenumber_in_poster",
    "movie_imdb_link",
    "num_user_for_reviews",
    "language",
    "country",
    "aspect_ratio"
]

# Dropar com segurança (ignora se alguma coluna não existir)
imdb = imdb.drop(columns=cols_extra_sem_equivalente, errors="ignore")
```

In [464…
```
# Colunas redundantes para remover
cols_remover = ["Genre"]

# Remover com segurança em ambos os DataFrames
base_desafio = base_desafio.drop(columns=cols_remover, errors="ignore")
imdb = imdb.drop(columns=cols_remover, errors="ignore")
```

In [465…
```
# Remover a coluna Gross antiga (texto)
base_desafio = base_desafio.drop(columns=["Gross"], errors="ignore")

# Renomear Gross float -> Gross
base_desafio = base_desafio.rename(columns={"Gross_float": "Gross"})

print("Colunas atuais:", base_desafio.columns)
```

```
Colunas atuais: Index(['Series_Title', 'Released_Year', 'Certificate', 'Runtime',
       'IMDB_Rating', 'Overview', 'Meta_score', 'Director', 'Star1', 'Star2',
       'Star3', 'Star4', 'No_of_Votes', 'Genres_list', 'Action', 'Adventure',
       'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family',
       'Fantasy', 'Film-Noir', 'History', 'Horror', 'Music', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War', 'Western',
       'Runtime_min', 'Gross'],
      dtype='object')
```

In [466…

```python
# Remover a coluna Gross antiga (texto)
base_desafio = base_desafio.drop(columns=["Runtime"], errors="ignore")

# Renomear Gross float -> Gross
base_desafio = base_desafio.rename(columns={"Runtime_min": "Runtime"})

print("Colunas atuais:", base_desafio.columns)
```

```
Colunas atuais: Index(['Series_Title', 'Released_Year', 'Certificate', 'IMDB_Rating',
       'Overview', 'Meta_score', 'Director', 'Star1', 'Star2', 'Star3',
       'Star4', 'No_of_Votes', 'Genres_list', 'Action', 'Adventure',
       'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family',
       'Fantasy', 'Film-Noir', 'History', 'Horror', 'Music', 'Musical',
       'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War', 'Western',
       'Runtime', 'Gross'],
      dtype='object')
```

O dataset IMDb importado apresentou alguns gêneros que **não estavam presentes na base do desafio**, como:

- `Short`
- `News`
- `Reality-TV`
- `Game-Show`
- `Documentary`

Foi feita uma contagem da frequência desses gêneros, e observou-se que o número de ocorrências era muito baixo.
Por esse motivo, decidiu-se:

1. **Remover os filmes** que pertenciam a esses gêneros pouco representativos.
2. **Excluir as colunas correspondentes** do dataframe final.

In [467…

```python
imdb['Short'].value_counts()
```

Out[467…

```
Short
0    5038
1       5
Name: count, dtype: int64
```

In [468…

```python
imdb['News'].value_counts()
```

Out[468…

```
News
0    5040
1       3
Name: count, dtype: int64
```

In [469…

```python
imdb['Reality-TV'].value_counts()
```

Out[469…
```
Reality-TV
0    5041
1       2
Name: count, dtype: int64
```

In [470…
```
imdb['Game-Show'].value_counts()
```

Out[470…
```
Game-Show
0    5042
1       1
Name: count, dtype: int64
```

In [471…
```
imdb['Documentary'].value_counts()
```

Out[471…
```
Documentary
0    4922
1     121
Name: count, dtype: int64
```

In [472…
```
# Gêneros pouco representativos
cols_generos_raros = ["Short", "News", "Reality-TV", "Game-Show", "Documentary"]

# 1. Remover filmes que pertencem a algum desses gêneros
imdb = imdb[~(imdb[cols_generos_raros].sum(axis=1) > 0)]

# 2. Remover as colunas desses gêneros
imdb = imdb.drop(columns=cols_generos_raros)

print("Formato final do imdb:", imdb.shape)
```

```
Formato final do imdb: (4916, 35)
```

Padronização do nome dos filmes, removendo espaços, pontuações e deixando tudo minusculo.

In [473…
```
# Padronizar Series_Title nos dois dataframes
def padronizar_titulo(df):
    df["Series_Title"] = df["Series_Title"].str.lower().str.strip()
    return df

# Aplicando
base_desafio = padronizar_titulo(base_desafio)
imdb = padronizar_titulo(imdb)
```

Como **as duas fontes são do IMDb**, primeiro conferi se havia **filmes repetidos** entre elas.
Para isso, usei **título** e **ano** como chaves de comparação:

In [474…
```
# Verificar interseção dos títulos entre os dois dataframes
titulos_base = set(base_desafio["Series_Title"])
titulos_imdb = set(imdb["Series_Title"])

# Filmes em comum
filmes_comuns = titulos_base.intersection(titulos_imdb)

print("Quantidade de filmes repetidos:", len(filmes_comuns))
```

```
Quantidade de filmes repetidos: 445
```

A base do desafio (999 filmes) **não possuía a coluna** `budget`.
Como esse dado está disponível na base de 5000 filmes do IMDb, foi realizado um processo de integração para enriquecer o dataset.

In [475…
```
# Garantir Released_Year numérico para não dar conflito
base_desafio["Released_Year"] = pd.to_numeric(base_desafio["Released_Year"], errors="coerce").as
imdb["Released_Year"] = pd.to_numeric(imdb["Released_Year"], errors="coerce").astype("Int64")

# Criar df auxiliar só com título, ano e budget
imdb_budget = imdb[["Series_Title", "Released_Year", "budget"]].drop_duplicates()

# Merge para adicionar budget ao base_desafio
base_desafio = base_desafio.merge(
    imdb_budget,
    on=["Series_Title", "Released_Year"],
    how="left"
)

# Dropar Star4 do base_desafio
base_desafio = base_desafio.drop(columns=["Star4"], errors="ignore")

# Relatório
print("Orçamentos preenchidos:", base_desafio["budget"].notna().sum())
print("Total de filmes base_desafio:", len(base_desafio))
print("Colunas finais:", base_desafio.columns.tolist())
```

```
Orçamentos preenchidos: 424
Total de filmes base_desafio: 999
Colunas finais: ['Series_Title', 'Released_Year', 'Certificate', 'IMDB_Rating', 'Overview', 'Meta
_score', 'Director', 'Star1', 'Star2', 'Star3', 'No_of_Votes', 'Genres_list', 'Action', 'Adventur
e', 'Animation', 'Biography', 'Comedy', 'Crime', 'Drama', 'Family', 'Fantasy', 'Film-Noir', 'Hist
ory', 'Horror', 'Music', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Sport', 'Thriller', 'War',
'Western', 'Runtime', 'Gross', 'budget']
```

Criação da coluna Meta_score na base de 5000 filmes, apesar de não termos os valores para eles desejo manter os valores que temos na nossa base original.

In [476…
```
if "Meta_score" not in imdb.columns:
    imdb["Meta_score"] = pd.NA
```

In [477…
```
imdb_unicos['Meta_score'] = pd.to_numeric(imdb_unicos["Meta_score"], errors="coerce")
```

Remoção dos filmes duplicados presentes tanto na base do desafio quanto na base nova que estamos importando.

In [478…
```
# Garantir tipos do ano para a chave
base_desafio["Released_Year"] = pd.to_numeric(base_desafio["Released_Year"], errors="coerce").as
imdb["Released_Year"] = pd.to_numeric(imdb["Released_Year"], errors="coerce").astype("Int64")

# Conjunto de chaves presentes no base_desafio
chaves_base = pd.MultiIndex.from_frame(base_desafio[["Series_Title", "Released_Year"]])

# Mascara: True se (titulo, ano) do imdb está no base_desafio
mask_overlap = pd.MultiIndex.from_frame(imdb[["Series_Title", "Released_Year"]]).isin(chaves_bas

# Quantos serão removidos
qtde_overlap = int(mask_overlap.sum())
print(f"Filmes a remover do imdb por serem duplicados: {qtde_overlap}")

# Remover duplicados do imdb
imdb_unicos = imdb[~mask_overlap].copy()

print("Formas:")
print(" - base_desafio:", base_desafio.shape)
```

```
print(" - imdb (original):", imdb.shape)
print(" - imdb (sem duplicados):", imdb_unicos.shape)
```

```
Filmes a remover do imdb por serem duplicados: 441
Formas:
 - base_desafio: (999, 36)
 - imdb (original): (4916, 36)
 - imdb (sem duplicados): (4475, 36)
```

Padronização das coluna overview

In [479…

```
# Substituir '|' por espaço no Overview do imdb_unicos
imdb_unicos["Overview"] = imdb_unicos["Overview"].str.replace("|", " ", regex=False).str.strip()
```

Remoção dos valores duplicados presentes dentro da base nova que estamos importando e junção dessa nova base com a base fornecida no desafio.

In [480…

```
# 1) Defina um critério para manter a MELHOR linha por (titulo, ano)
def pick_best(group):
    # Preferir quem tem budget e gross não nulos, mais votos e overview mais completo
    g = group.assign(
        budget_notna=group["budget"].notna(),
        gross_notna=group["Gross"].notna(),
        overview_len=group["Overview"].fillna("").str.len()
    )
    return (
        g.sort_values(
            ["budget_notna", "gross_notna", "No_of_Votes", "overview_len"],
            ascending=[False, False, False, False],
        )
        .iloc[[0]]
        .drop(columns=["budget_notna", "gross_notna", "overview_len"])
    )

# 2) Deduplicar o imdb_unicos por (Series_Title, Released_Year)
imdb_unicos_dedup = (
    imdb_unicos
    .groupby(["Series_Title", "Released_Year"], as_index=False, group_keys=False)
    .apply(pick_best)
    .reset_index(drop=True)
)

print("Antes:", imdb_unicos.shape, "Depois:", imdb_unicos_dedup.shape)

# 3) Agora faça o concat com o base_desafio
df_final = pd.concat([base_desafio, imdb_unicos_dedup], ignore_index=True)
print(df_final.shape)
```

```
Antes: (4475, 36) Depois: (4263, 36)
(5262, 36)
C:\Users\guima\AppData\Local\Temp\ipykernel_41256\834952163.py:22: FutureWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
  .apply(pick_best)
C:\Users\guima\AppData\Local\Temp\ipykernel_41256\834952163.py:29: FutureWarning: The behavior of DataFrame concatenation with empty or all-NA entries is deprecated. In a future version, this will no longer exclude empty or all-NA columns when determining the result dtypes. To retain the old behavior, exclude the relevant entries before the concat operation.
  df_final = pd.concat([base_desafio, imdb_unicos_dedup], ignore_index=True)
```

Validando que deu certo a junção!!!

In [481…
```
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5262 entries, 0 to 5261
Data columns (total 36 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Series_Title   5262 non-null   object
 1   Released_Year  5261 non-null   Int64
 2   Certificate    4946 non-null   object
 3   IMDB_Rating    5262 non-null   float64
 4   Overview       5139 non-null   object
 5   Meta_score     842 non-null    float64
 6   Director       5262 non-null   object
 7   Star1          5262 non-null   object
 8   Star2          5261 non-null   object
 9   Star3          5258 non-null   object
 10  No_of_Votes    5262 non-null   int64
 11  Genres_list    5262 non-null   object
 12  Action         5262 non-null   int64
 13  Adventure      5262 non-null   int64
 14  Animation      5262 non-null   int64
 15  Biography      5262 non-null   int64
 16  Comedy         5262 non-null   int64
 17  Crime          5262 non-null   int64
 18  Drama          5262 non-null   int64
 19  Family         5262 non-null   int64
 20  Fantasy        5262 non-null   int64
 21  Film-Noir      5262 non-null   int64
 22  History        5262 non-null   int64
 23  Horror         5262 non-null   int64
 24  Music          5262 non-null   int64
 25  Musical        5262 non-null   int64
 26  Mystery        5262 non-null   int64
 27  Romance        5262 non-null   int64
 28  Sci-Fi         5262 non-null   int64
 29  Sport          5262 non-null   int64
 30  Thriller       5262 non-null   int64
 31  War            5262 non-null   int64
 32  Western        5262 non-null   int64
 33  Runtime        5250 non-null   float64
 34  Gross          4413 non-null   float64
 35  budget         4330 non-null   float64
dtypes: Int64(1), float64(5), int64(22), object(8)
memory usage: 1.5+ MB
```

In [482…
```
df_final.head()
```

Out[482…

	Series_Title	Released_Year	Certificate	IMDB_Rating	Overview	Meta_score	Director	Star1	
0	the godfather	1972	A	9.2	organized crime dynastys aging patriarch trans...	100.0	Francis Ford Coppola	Marlon Brando	A
1	the dark knight	2008	UA	9.0	menace known Joker wreaks havoc chaos people G...	84.0	Christopher Nolan	Christian Bale	
2	the godfather: part ii	1974	A	9.0	early life career Vito Corleone 1920s New York...	90.0	Francis Ford Coppola	Al Pacino	Rc
3	12 angry men	1957	U	9.0	jury holdout attempts prevent miscarriage just...	96.0	Sidney Lumet	Henry Fonda	
4	the lord of the rings: the return of the king	2003	U	8.9	Gandalf Aragorn lead World Men Saurons army dr...	94.0	Peter Jackson	Elijah Wood	Mc

5 rows × 36 columns

3.Salvando os dados:

Após todas as etapas de integração e tratamento, o dataset final foi salvo em `.csv` para garantir sua reutilização.

Esse arquivo consolidado será utilizado nas próximas etapas do desafio, permitindo dar continuidade às análises exploratórias e aos testes de hipóteses com uma base mais completa e consistente.

Uma observação importante os nulos não foram tratados ainda pois pode impactar negativamente nas analises da 2 questão, porém para a modelagem da segunda iremos tratar!!

In [483…

```
df_final.to_csv("..\data\processed\df_eda01_plus_5000.csv", index=False)
```

```
<>:1: SyntaxWarning: invalid escape sequence '\d'
<>:1: SyntaxWarning: invalid escape sequence '\d'
C:\Users\guima\AppData\Local\Temp\ipykernel_41256\864039978.py:1: SyntaxWarning: invalid escape s
equence '\d'
  df_final.to_csv("..\data\processed\df_eda01_plus_5000.csv", index=False)
```

In [ ]: