LH_CD_MIRIAMAGUIARSOBRALVF_pdf

July 22, 2024

0.1 DESAFIO CIENTISTA DE DADOS - PROGRAMA LIGHT HOUSE INDICIUM

POR MIRIAM O. A SOBRAL

> Você foi alocado em um time da Indicium contratado por um estúdio de Hollywood chamado PProductions, e agora deve fazer uma análise em cima de um banco de dados cinematográfico para orientar qual tipo de filme deve ser o próximo a ser desenvolvido. Lembre-se que há muito dinheiro envolvido, então a análise deve ser muito detalhada e levar em consideração o máximo de fatores possíveis (a introdução de dados externos é permitida - e encorajada).

0.2 Dicionário dos dados

- Id Colunas: Descrição
- 01 Series_Title: Nome do filme
- 02 Released_Year: Ano de lançamento
- 03 Certificate: Classificação etária
- 04 Runtime: Tempo de duração
- 05 Genre: Gênero
- 06 IMDB_Rating: Nota do IMDB
- 07 Overview: Overview do filme
- 08 Meta_score: Média ponderada de todas as críticas
- 09 Director: Diretor
- 10 Star1: Ator/atriz #1
- 11 Star2: Ator/atriz #2
- 12 Star3: Ator/atriz #3
- 13 Star4: Ator/atriz #4
- 14 No_of_Votes: Número de votos
- 15 Gross: Faturamento

##Steps » * 1 - Business Understanding » * 2 - Data Understanding
» * 3 - Data Preparation » * 4 - Modeling » * 5 - Evaluation
» * 6 - Deployment

0.3 Entregas do projeto

- Produto A: Análise exploratória de dados (EDA)
- Produto B: Achados e insights

- Produto C: Modelagem Preditiva
- Produto D: Teste do modelo
- Produto E: Arquivo salvo.pkl
- Produto F: Repositório GITHUB

##1. Business Understanding

- A indústria cinematográfica enfrenta uma grande crise. De acordo com os dados divulgados durante a CinemaCon 2024, a consultoria Gower Street Analytics prevê que o mercado cinematográfico mundial movimentará US$ 32,3 bilhões (R$ 165,7 bilhões) em 2024, uma queda de US$ 1 bilhão (R$ 5,15 bilhões) ou 3% em relação a 2023. Essa diminuição, embora pareça pequena, é um golpe significativo para uma indústria que ainda tenta retornar aos níveis de bilheteria pré-pandêmicos. Em 2019, o mercado cinematográfico global movimentou US$ 42,5 bilhões (R$ 218 bilhões), de acordo com dados da Comscore. Portanto, a projeção de 2024 representa uma queda de aproximadamente 24% em relação ao período anterior à pandemia (UOL, 2024).
- Para enfrentar esse cenário desafiador, muitas produtoras têm utilizado a ciência de dados para projetar propostas de filmes que alcancem sucesso de bilheteria nos próximos meses. O uso de big data pode ser determinante para o sucesso na escolha do gênero, classificação etária e elenco de estrelas, visando atrair o público-alvo e maximizar o retorno financeiro.
- Através da análise de dados históricos de bilheteria, preferências do público, tendências de mercado e outros insights valiosos, as produtoras podem tomar decisões mais embasadas e estratégicas na seleção de projetos cinematográficos. Essa abordagem baseada em dados pode ajudar a indústria a se adaptar mais rapidamente às mudanças no comportamento do consumidor e a oferecer conteúdo que realmente atenda às expectativas do público.

Referências:

Portal Uol: https://www.uol.com.br/splash/colunas/na-sua-tela/2024/05/19/cinema-queda-bilheteria-publico-2024.htm?cmpid=copiaecola

##2. Data Understanding

[444]:
```
# Importando as bibliotecas
import pandas as pd
import numpy as np
import re
import nltk
!pip install nltk
nltk.download('punkt')
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
```

```
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
import pickle
import plotly.express as px
from sklearn.linear_model import LinearRegression
from keras.models import Sequential
from keras.layers import Dense
import pickle
from sklearn.model_selection import train_test_split, GridSearchCV
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
  ↪texlive-plain-generic
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import os
!pip install ydata-profiling
from ydata_profiling import ProfileReport
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from nltk.tokenize import word_tokenize
from wordcloud import WordCloud
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
!pip install pandas-profiling
!pip install pydantic-settings
warnings.filterwarnings("ignore")
!pip install nbconvert
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
  ↪texlive-plain-generic
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2024.5.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.4)

[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!

Reading package lists… Done
```

```
Building dependency tree… Done
Reading state information… Done
texlive-fonts-recommended is already the newest version (2021.20220204-1).
texlive-plain-generic is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
Requirement already satisfied: ydata-profiling in
/usr/local/lib/python3.10/dist-packages (4.9.0)
Requirement already satisfied: scipy<1.14,>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.11.4)
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.0.3)
Requirement already satisfied: matplotlib<3.10,>=3.5 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.7.1)
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.10/dist-
packages (from ydata-profiling) (2.8.2)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (6.0.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (3.1.4)
Requirement already satisfied: visions[type_image_path]<0.7.7,>=0.7.5 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.7.6)
Requirement already satisfied: numpy<2,>=1.16.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.25.2)
Requirement already satisfied: htmlmin==0.1.12 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.12.4)
Requirement already satisfied: requests<3,>=2.24.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (2.31.0)
Requirement already satisfied: tqdm<5,>=4.48.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.66.4)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.13.1)
Requirement already satisfied: multimethod<2,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.12)
Requirement already satisfied: statsmodels<1,>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.14.2)
Requirement already satisfied: typeguard<5,>=3 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.3.0)
Requirement already satisfied: imagehash==4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (4.3.1)
Requirement already satisfied: wordcloud>=1.9.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (1.9.3)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.10/dist-
packages (from ydata-profiling) (1.8.1)
Requirement already satisfied: numba<1,>=0.56.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.58.1)
```

```
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-
packages (from imagehash==4.3.1->ydata-profiling) (1.6.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from imagehash==4.3.1->ydata-profiling) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-
profiling) (2.1.5)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib<3.10,>=3.5->ydata-profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling) (2.8.2)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydata-profiling)
(0.41.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling) (2024.1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-
packages (from phik<0.13,>=0.11.1->ydata-profiling) (1.4.2)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
(0.7.0)
Requirement already satisfied: pydantic-core==2.20.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
(2.20.1)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profiling)
(4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling) (3.3.2)
```

```
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.24.0->ydata-profiling) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling) (2024.7.4)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels<1,>=0.13.2->ydata-profiling) (0.5.6)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-
packages (from visions[type_image_path]<0.7.7,>=0.7.5->ydata-profiling) (23.2.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-
packages (from visions[type_image_path]<0.7.7,>=0.7.5->ydata-profiling) (3.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels<1,>=0.13.2->ydata-profiling) (1.16.0)
Requirement already satisfied: pandas-profiling in
/usr/local/lib/python3.10/dist-packages (3.6.6)
Requirement already satisfied: ydata-profiling in
/usr/local/lib/python3.10/dist-packages (from pandas-profiling) (4.9.0)
Requirement already satisfied: scipy<1.14,>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(1.11.4)
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(2.0.3)
Requirement already satisfied: matplotlib<3.10,>=3.5 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(3.7.1)
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.10/dist-
packages (from ydata-profiling->pandas-profiling) (2.8.2)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(6.0.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(3.1.4)
Requirement already satisfied: visions[type_image_path]<0.7.7,>=0.7.5 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(0.7.6)
Requirement already satisfied: numpy<2,>=1.16.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(1.25.2)
Requirement already satisfied: htmlmin==0.1.12 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(0.1.12)
Requirement already satisfied: phik<0.13,>=0.11.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
```

```
(0.12.4)
Requirement already satisfied: requests<3,>=2.24.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(2.31.0)
Requirement already satisfied: tqdm<5,>=4.48.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(4.66.4)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(0.13.1)
Requirement already satisfied: multimethod<2,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(1.12)
Requirement already satisfied: statsmodels<1,>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(0.14.2)
Requirement already satisfied: typeguard<5,>=3 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(4.3.0)
Requirement already satisfied: imagehash==4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(4.3.1)
Requirement already satisfied: wordcloud>=1.9.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(1.9.3)
Requirement already satisfied: dacite>=1.8 in /usr/local/lib/python3.10/dist-
packages (from ydata-profiling->pandas-profiling) (1.8.1)
Requirement already satisfied: numba<1,>=0.56.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas-profiling)
(0.58.1)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-
packages (from imagehash==4.3.1->ydata-profiling->pandas-profiling) (1.6.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from imagehash==4.3.1->ydata-profiling->pandas-profiling) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-
profiling->pandas-profiling) (2.1.5)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling->pandas-profiling) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib<3.10,>=3.5->ydata-profiling->pandas-profiling)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling->pandas-profiling) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
```

```
profiling->pandas-profiling) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling->pandas-profiling) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling->pandas-profiling) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.10,>=3.5->ydata-
profiling->pandas-profiling) (2.8.2)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydata-
profiling->pandas-profiling) (0.41.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling->pandas-profiling)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling->pandas-profiling)
(2024.1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-
packages (from phik<0.13,>=0.11.1->ydata-profiling->pandas-profiling) (1.4.2)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas-profiling) (0.7.0)
Requirement already satisfied: pydantic-core==2.20.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas-profiling) (2.20.1)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas-profiling) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas-profiling) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.24.0->ydata-profiling->pandas-profiling) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas-profiling) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas-profiling) (2024.7.4)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels<1,>=0.13.2->ydata-profiling->pandas-profiling)
(0.5.6)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-
packages (from visions[type_image_path]<0.7.7,>=0.7.5->ydata-profiling->pandas-
profiling) (23.2.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-
```

```
packages (from visions[type_image_path]<0.7.7,>=0.7.5->ydata-profiling->pandas-
profiling) (3.3)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels<1,>=0.13.2->ydata-profiling->pandas-profiling)
(1.16.0)
Requirement already satisfied: pydantic-settings in
/usr/local/lib/python3.10/dist-packages (2.3.4)
Requirement already satisfied: pydantic>=2.7.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic-settings) (2.8.2)
Requirement already satisfied: python-dotenv>=0.21.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic-settings) (1.0.1)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.7.0->pydantic-
settings) (0.7.0)
Requirement already satisfied: pydantic-core==2.20.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.7.0->pydantic-
settings) (2.20.1)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.7.0->pydantic-
settings) (4.12.2)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (6.5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (3.1.4)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (24.1)
```

```
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.16.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (1.3.0)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert)
(4.2.2)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from nbclient>=0.5.0->nbconvert)
(6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.19.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (24.0.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.3.3)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
texlive-fonts-recommended is already the newest version (2021.20220204-1).
texlive-plain-generic is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
```

[ ]:
```
#Conectando ao drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
df = pd.read_csv("/content/drive/MyDrive/Indicium/desafio_indicium_imdb.csv")
```

```
Mounted at /content/drive
```

[ ]:
```
df.head(3)
```

[ ]:

	Unnamed: 0	Series_Title	Released_Year	Certificate	Runtime	Genre	IMDB_Rating	Overview	Meta_score	Director	Star1	Star2	Star3	Star4	No_of_Votes	Gross
0	1	The Godfather	1972	A	175 min	Crime, Drama	9.2	An organized crime dynasty's aging patriarch t…	100.0	Francis Ford Coppola	Marlon Brando	Al Pacino	James Caan	Diane Keaton	1620367	134,966,411
1	2	The Dark Knight	2008	UA	152 min	Action, Crime, Drama	9.0	When the menace known as the Joker wreaks havo…	84.0	Christopher Nolan	Christian Bale	Heath Ledger	Aaron Eckhart	Michael Caine	2303232	534,858,444
2	3	The Godfather: Part II	1974	A	202 min	Crime, Drama	9.0	The early life and career of Vito Corleone in …	90.0	Francis Ford Coppola	Al Pacino	Robert De Niro	Robert Duvall	Diane Keaton	1129952	57,300,000

##3. Data Preparation

Os dados brutos raramente estão prontos para a análise. Nesta fase, realizamos a limpeza dos dados, tratamos valores ausentes ou inconsistentes e integramos diferentes fontes de dados. O objetivo é criar um conjunto de dados preparado para as etapas subsequentes.

[ ]:
```
#Vamos iniciar a preparação dos dados, iniciando pelo dataset que foi cedido␣
 ↪pela indicium
# Observando as primeiras infos do nosso dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     999 non-null    int64
 1   Series_Title   999 non-null    object
 2   Released_Year  999 non-null    object
 3   Certificate    898 non-null    object
 4   Runtime        999 non-null    object
 5   Genre          999 non-null    object
 6   IMDB_Rating    999 non-null    float64
 7   Overview       999 non-null    object
 8   Meta_score     842 non-null    float64
 9   Director       999 non-null    object
 10  Star1          999 non-null    object
 11  Star2          999 non-null    object
 12  Star3          999 non-null    object
 13  Star4          999 non-null    object
 14  No_of_Votes    999 non-null    int64
 15  Gross          830 non-null    object
dtypes: float64(2), int64(2), object(12)
memory usage: 125.0+ KB
```

[ ]:
```
#Conhecendo o tamanho do nosso dataset
df.shape
```

[ ]:
```
(999, 16)
```

[ ]:
```
# Analisando os tipos de variáveis
df.dtypes
```

[ ]:
```
Unnamed: 0         int64
Series_Title      object
Released_Year     object
Certificate       object
Runtime           object
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
```

```
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross             object
dtype: object
```

[ ]:
```
#Visualizando a existência de dados nulos
df.isnull().sum()
```

[ ]:
```
Unnamed: 0         0
Series_Title       0
Released_Year      0
Certificate      101
Runtime            0
Genre              0
IMDB_Rating        0
Overview           0
Meta_score       157
Director           0
Star1              0
Star2              0
Star3              0
Star4              0
No_of_Votes        0
Gross            169
dtype: int64
```

[ ]:
```
# Notamos que a coluna Unnamed não agrega ao estudo, portanto, vamos eliminá-la
df.drop(columns="Unnamed: 0", inplace=True)
```

[ ]:
```
# Contagem de linhas duplicadas
df.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
# Contagem de NaNs por coluna
df.isna().sum()
```

[ ]:
```
Series_Title       0
Released_Year      0
Certificate      101
Runtime            0
Genre              0
IMDB_Rating        0
Overview           0
```

```
Meta_score       157
Director           0
Star1              0
Star2              0
Star3              0
Star4              0
No_of_Votes        0
Gross            169
dtype: int64
```

[ ]:
```
#Usamos a função para encontrar o número de valores únicos no eixo da coluna.
df.nunique()
```

[ ]:
```
Series_Title     998
Released_Year    100
Certificate       16
Runtime          140
Genre            202
IMDB_Rating       16
Overview         999
Meta_score        63
Director         548
Star1            659
Star2            840
Star3            890
Star4            938
No_of_Votes      998
Gross            822
dtype: int64
```

[ ]:
```
#Criando função para verificar os valores únicos de algumas colunas que␣
 ↪selecionamos
def valores_unicos(df, colunas):
    for coluna in colunas:
        print(f'\n Valores únicos coluna {coluna}:')
        print(df[coluna].unique())
```

[ ]:
```
#Verificando os valores únicos de cada coluna
val_unicos=['Released_Year', 'Runtime', 'Gross', 'Certificate']
valores_unicos(df,val_unicos)
```

```
 Valores únicos coluna Released_Year:
['1972' '2008' '1974' '1957' '2003' '1994' '1993' '2010' '1999' '2001'
 '1966' '2002' '1990' '1980' '1975' '2020' '2019' '2014' '1998' '1997'
 '1995' '1991' '1977' '1962' '1954' '1946' '2011' '2006' '2000' '1988'
 '1985' '1968' '1960' '1942' '1936' '1931' '2018' '2017' '2016' '2012'
```

```
 '2009' '2007' '1984' '1981' '1979' '1971' '1963' '1964' '1950' '1940'
 '2013' '2005' '2004' '1992' '1987' '1986' '1983' '1976' '1973' '1965'
 '1959' '1958' '1952' '1948' '1944' '1941' '1927' '1921' '2015' '1996'
 '1989' '1978' '1961' '1955' '1953' '1925' '1924' '1982' '1967' '1951'
 '1949' '1939' '1937' '1934' '1928' '1926' '1920' '1970' '1969' '1956'
 '1947' '1945' '1930' '1938' '1935' '1933' '1932' '1922' '1943' 'PG']

 Valores únicos coluna Runtime:
['175 min' '152 min' '202 min' '96 min' '201 min' '154 min' '195 min'
 '148 min' '139 min' '178 min' '142 min' '161 min' '179 min' '136 min'
 '146 min' '124 min' '133 min' '160 min' '132 min' '153 min' '169 min'
 '130 min' '125 min' '189 min' '116 min' '127 min' '118 min' '121 min'
 '207 min' '122 min' '106 min' '112 min' '151 min' '150 min' '155 min'
 '119 min' '110 min' '88 min' '137 min' '89 min' '165 min' '109 min'
 '102 min' '87 min' '126 min' '147 min' '117 min' '181 min' '149 min'
 '105 min' '164 min' '170 min' '98 min' '101 min' '113 min' '134 min'
 '229 min' '115 min' '143 min' '95 min' '104 min' '123 min' '131 min'
 '108 min' '81 min' '99 min' '114 min' '129 min' '228 min' '128 min'
 '103 min' '107 min' '68 min' '138 min' '156 min' '167 min' '163 min'
 '186 min' '321 min' '135 min' '140 min' '180 min' '158 min' '210 min'
 '86 min' '162 min' '177 min' '204 min' '91 min' '172 min' '45 min'
 '145 min' '100 min' '196 min' '93 min' '120 min' '92 min' '144 min'
 '80 min' '183 min' '111 min' '141 min' '224 min' '171 min' '188 min'
 '94 min' '185 min' '85 min' '205 min' '212 min' '238 min' '72 min'
 '67 min' '76 min' '159 min' '83 min' '90 min' '84 min' '191 min'
 '197 min' '174 min' '97 min' '75 min' '157 min' '209 min' '82 min'
 '220 min' '64 min' '184 min' '168 min' '166 min' '192 min' '194 min'
 '193 min' '69 min' '70 min' '242 min' '79 min' '71 min' '78 min']

 Valores únicos coluna Gross:
['134,966,411' '534,858,444' '57,300,000' '4,360,000' '377,845,905'
 '107,928,762' '96,898,818' '292,576,195' '37,030,102' '315,544,750'
 '330,252,182' '6,100,000' '342,551,365' '171,479,930' '46,836,394'
 '290,475,067' '112,000,000' nan '53,367,844' '188,020,017' '7,563,397'
 '10,055,859' '216,540,909' '136,801,374' '57,598,247' '100,125,643'
 '130,742,922' '322,740,140' '269,061' '335,451,311' '13,092,000'
 '13,182,281' '53,089,891' '132,384,315' '32,572,577' '187,705,427'
 '6,719,864' '23,341,568' '19,501,238' '422,783,777' '204,843,350'
 '11,990,401' '210,609,762' '5,321,508' '32,000,000' '1,024,560' '163,245'
 '19,181' '1,661,096' '5,017,246' '12,391,761' '190,241,310' '858,373,000'
 '678,815,482' '209,726,015' '162,805,434' '448,139,099' '6,532,908'
 '1,223,869' '223,808,164' '11,286,112' '707,481' '25,544,867' '2,375,308'
 '248,159,971' '44,017,374' '83,471,511' '78,900,000' '275,902'
 '8,175,000' '36,764,313' '288,475' '159,227,644' '1,373,943' '687,185'
 '7,098,492' '6,857,096' '120,540,719' '34,400,301' '33,225,499'
 '30,328,156' '3,635,482' '130,096,601' '138,433,435' '933,933'
 '191,796,233' '75,600,000' '2,832,029' '46,357,676' '85,160,248'
 '51,973,029' '45,598,982' '309,125,409' '11,487,676' '28,262,574'
```

'159,600,000' '6,207,725' '56,954,992' '15,000,000' '44,824,144'
'18,600,000' '13,275,000' '3,200,000' '8,819,028' '55,240' '332,930'
'5,720,000' '1,585,634' '28,877' '1,236,166' '5,450,000' '898,575'
'4,186,168' '85,080,171' '54,513,740' '342,370' '20,186,659' '739,478'
'1,429,534' '144,501' '1,626,289' '7,461' '39,567' '6,391,436'
'13,657,115' '128,012,934' '293,004,164' '116,900,694' '1,113,541'
'40,222,514' '37,634,615' '415,004,880' '70,511,035' '2,197,331'
'733,094' '206,852,432' '1,223,240' '5,509,040' '4,711,096' '170,742,341'
'3,897,569' '64,616,940' '67,436,818' '42,438,300' '101,157,447'
'197,171,806' '280,015' '1,105,564' '83,008,852' '4,135,750' '23,383,987'
'234,723' '1,229,197' '12,100,000' '25,000,000' '57,226' '12,562'
'96,568' '10,177' '5,014,000' '977,375' '3,759,854' '1,241,223'
'2,006,788' '3,492,754' '901,610' '226,277,068' '14,677,674' '3,107,072'
'165,520' '10,616,104' '923,221' '59,100,318' '167,767,189' '67,209,615'
'356,461,711' '2,804,874' '56,671,993' '26,947,624' '117,624,028'
'45,055,776' '857,524' '1,035,953' '3,108,485' '61,002,302' '154,058,340'
'148,095,302' '381,011,219' '1,498,210' '217,581,231' '18,354,356'
'74,283,625' '2,217,561' '100,492,203' '23,530,892' '1,111,061'
'5,820,649' '14,131' '300,000' '70,099,045' '380,843,261' '164,615,351'
'5,383,834' '289,916,256' '70,147' '293,506,292' '259,127' '125,618,201'
'348,660' '17,498,804' '2,734,044' '16,501,785' '24,611,975' '171,082'
'309,811' '5,535,405' '4,043,686' '600,200' '402,453,882' '25,010,410'
'5,216,888' '2,603,061' '95,860,116' '52,287,414' '138,530,565'
'2,181,987' '495,770' '13,782,838' '22,244,207' '4,971,340' '32,868,943'
'20,045,115' '48,979,328' '117,235,247' '177,345' '29,000,000'
'30,933,743' '1,742,348' '39,481' '16,217,773' '102,021' '55,908'
'4,050,000' '1,526,000' '74,700,000' '752,045' '44,908,000' '9,600,000'
'3,969,893' '449,191' '10,900,000' '203,300' '198,676,459' '172,885'
'21,877' '539,540' '1,033,895' '8,178,001' '151,101,803' '51,739,495'
'228,433,663' '341,268,248' '6,738,000' '1,506,975' '132,422,809'
'5,566,534' '1,079,369' '1,010,414' '2,625,803' '91,125,683'
'333,176,600' '92,054,159' '25,568,251' '216,428,042' '183,637,894'
'17,738,570' '100,119' '138,797,449' '169,708,112' '363,070,709'
'106,662' '10,950' '4,018,695' '141,319,928' '106,954,678' '8,060'
'623,279,547' '4,445,756' '27,298,285' '163,566,459' '57,366,262'
'227,471,070' '238,507' '74,103,820' '5,990,075' '206,445,654'
'167,445,960' '66,208,183' '502,028' '2,380,788' '2,086,345' '61,649,911'
'1,787,378' '169,659' '305,413,918' '66,257,002' '261,441,092'
'1,530,386' '1,480,006' '22,455,976' '4,184,036' '6,203,044' '776,665'
'1,647,780' '5,595,428' '23,159,305' '3,296' '24,475,416' '13,417,292'
'57,141,459' '515,905' '75,082,668' '70,906,973' '4,496,583' '63,895,607'
'217,350,219' '70,405,498' '218,967,620' '184,208,848' '27,545,445'
'178,800,000' '553,171' '30,857,814' '3,333,969' '4,542,825' '38,400,000'
'52,767,889' '30,177,511' '39,200,000' '260,000,000' '50,000,000'
'86,300,000' '53,267,000' '232,906,145' '4,081,254' '29,133,000'
'80,500,000' '541,940' '102,308,889' '33,395,426' '104,945,305'
'163,214,286' '111,722,000' '14,500,000' '50,690' '8,284,000'
'19,516,000' '12,535,000' '11,900,000' '2,237,659' '17,570,324' '654,000'

'8,000,000' '23,650,000' '2,108,060' '55,000' '2,076,020' '3,270,000'
'50,970' '165,359,751' '3,313,513' '2,000,000' '18,095,701' '258,168'
'32,015,231' '5,202,582' '5,875,006' '3,237,118' '315,058,289'
'32,381,218' '349,555' '100,546,139' '936,662,225' '8,114,627'
'233,921,534' '138,730' '44,671,682' '100,206,256' '6,739,492'
'7,000,000' '2,084,637' '5,009,677' '2,122,065' '115,646,235'
'26,236,603' '410,800' '25,379,975' '1,185,783' '257,730,019' '7,757,130'
'13,756,082' '2,280,348' '124,987,023' '21,002,919' '3,635,164'
'318,412,101' '13,542,874' '90,135,191' '249,358,727' '53,710,019'
'9,284,265' '1,059,830' '1,000,045' '121,661,683' '1,221,261'
'35,552,383' '32,534,850' '7,220,243' '245,852,179' '26,400,640'
'2,201,126' '12,281,500' '1,324,974' '36,948,322' '56,362,352'
'14,743,391' '18,254,702' '76,270,454' '40,084,041' '9,929,135' '188,751'
'20,605,209' '57,229,890' '45,700,000' '37,823,676' '5,100,000'
'70,600,000' '61,001' '198,809' '61,700,000' '12,064,472' '89,029'
'22,276,975' '24,379,978' '13,474,588' '30,000,000' '88,300' '93,740,000'
'9,450,000' '7,630,000' '2,650,000' '10,464,000' '6,540,000' '9,460,000'
'296,000' '3,981,000' '2,537,520' '10,000,000' '2,402,067' '188,373,161'
'169,607,287' '40,442,052' '48,023,088' '1,782,795' '47,695,120'
'435,266' '532,177,324' '408,084,349' '54,117,416' '108,101,214'
'6,735,118' '1,330,596' '2,852,400' '4,231,500' '169,209' '222,527,828'
'15,322,921' '1,670,773' '5,209,580' '434,038,008' '85,433' '45,512,466'
'177,002,924' '70,259,870' '128,392' '102,515,793' '107,100,855' '6,460'
'161,197,785' '547,292' '258,366,855' '2,222,647' '10,095,170'
'93,617,009' '145,000,989' '9,030,581' '9,439,923' '303,003,568'
'741,283' '50,927,067' '35,061,555' '130,164,645' '760,507,625' '3,600'
'50,866,635' '59,891,098' '23,637,265' '127,509,326' '2,921,738' '1,305'
'38,405,088' '5,128,124' '128,985' '25,514,517' '119,519,402' '15,280'
'81,001,787' '16,756,372' '181,655' '7,002,255' '54,234,062' '2,892,011'
'115,654,751' '128,078,872' '8,264,530' '233,632,142' '25,812'
'28,965,197' '32,481,825' '267,665,011' '659,325,379' '233,986'
'12,339,633' '48,323,648' '4,040,691' '5,887,457' '9,170,214'
'56,505,065' '32,416,586' '22,954,968' '1,769,305' '183,875,760'
'17,266,971' '5,617,391' '2,015,810' '1,999,955' '57,504,069'
'61,276,872' '52,096,475' '26,830,000' '118,500,000' '34,603,943'
'59,735,548' '5,923,044' '70,136,369' '1,436,000' '61,503,218'
'98,467,863' '45,875,171' '34,700,291' '238,632,124' '21,500,000'
'2,500,000' '435,110,554' '106,260,000' '31,800,000' '4,420,000'
'193,817' '16,056,255' '42,765,000' '4,000,000' '44,785,053' '17,550,741'
'56,700,000' '72,000,000' '102,272,727' '39,100,000' '2,616,000'
'336,705' '335,609' '324,591,735' '33,800,859' '176,040,665'
'220,159,104' '3,358,518' '3,333,000' '765,127' '35,893,537'
'128,261,724' '124,872,350' '6,743,776' '42,340,598' '2,199,675'
'1,122,527' '175,058' '259,766,572' '985,912' '17,654,912' '189,422,889'
'1,339,152' '71,177' '56,816,662' '257,760,692' '274,092,705'
'228,778,661' '83,861' '96,962,694' '146,408,305' '277,322,503'
'304,360,277' '132,092,958' '136,025,503' '32,391,374' '295,983,305'
'15,090,400' '18,593,156' '881,302' '35,739,802' '1,752,214' '327,919'

```
 '25,442,958' '412,544' '17,605,861' '33,080,084' '22,494,487' '1,054,361'
 '871,577' '200,821,936' '4,398,392' '83,043,761' '53,606,916'
 '54,580,300' '17,108,591' '176,241,941' '1,082,715' '75,331,600'
 '44,585,453' '290,013,036' '77,911,774' '75,286,229' '111,110,575'
 '4,890,878' '4,064,200' '15,539,266' '40,311,852' '22,245,861'
 '108,638,745' '104,454,762' '141,072' '21,995,263' '3,029,081'
 '183,417,150' '151,086' '45,512,588' '52,037,603' '10,824,921'
 '2,807,390' '76,631,907' '17,105,219' '368,234' '22,858,926'
 '148,478,011' '63,540,020' '4,065,116' '41,909,762' '35,811,509'
 '56,116,183' '4,414,535' '10,019,307' '21,848,932' '3,151,130'
 '6,110,979' '77,324,422' '27,281,507' '48,169,908' '75,505,856'
 '10,725,228' '141,340,178' '82,418,501' '6,153,939' '5,080,409'
 '156,452,370' '1,544,889' '43,984,230' '22,238,696' '7,153,487'
 '8,551,228' '10,631,333' '10,600,000' '11,798,616' '54,000,000'
 '78,912,963' '47,212,904' '54,800,000' '83,400,000' '47,000,000'
 '1,924,733' '119,500,000' '29,800,000' '7,056,013' '15,630,710'
 '35,900,000' '44,527,234' '45,300,000' '26,331' '28,350,000' '51,081,062'
 '11,403,529' '83,957' '4,905,000' '52,709' '10,550,000' '536,118'
 '76,408,097' '26,020,957' '142,502,728' '566,356' '5,904,366'
 '24,801,212' '13,122,642' '845,464' '389,813,101' '107,825,862'
 '18,340,051' '72,313,754' '608,581,744' '248,757,044' '46,889,293'
 '109,767,581' '1,092,800' '26,862,450' '37,707,719' '208,545,589'
 '4,105,123' '41,003,371' '19,202,743' '215,288,866' '171,243,005'
 '1,008,098' '251,513,985' '35,014,192' '48,071,303' '75,605,492'
 '3,185,812' '75,590,286' '209,028,679' '255,959,475' '9,422,422'
 '69,951,824' '17,114,882' '38,634,938' '686,383' '88,513,495'
 '20,300,218' '10,301,706' '24,633,730' '39,868,642' '6,013' '301,959,197'
 '210,614,939' '23,089,926' '107,509,799' '24,149,632' '56,000,369'
 '3,081,925' '697,181' '548,707' '6,167,817' '5,739,376' '16,290,476'
 '45,289' '51,680,613' '13,060,843' '57,938,693' '45,064,915' '52,364,010'
 '13,622,333' '317,575,550' '96,522,687' '52,990,775' '51,401,758'
 '132,072,926' '50,668,906' '15,070,285' '7,267,585' '36,400,491'
 '120,620,254' '10,680,275' '14,378,331' '49,100,000' '3,796,699'
 '43,182,776' '100,012,499' '1,037,847' '71,516,617' '173,837,933'
 '1,464,625' '40,903,593' '7,993,039' '52,929,168' '453,243' '1,794,187'
 '285,761,243' '66,666,062' '92,823,600' '111,543,479' '78,756,177'
 '49,530,280' '65,207,127' '2,150,000' '119,285,432' '12,465,371'
 '22,490,039' '76,657,000' '43,000,000' '35,000,000' '132,088,635'
 '959,000' '696,690' '1,378,435' '141,843,612' '13,780,024' '30,500,000']

 Valores únicos coluna Certificate:
['A' 'UA' 'U' 'PG-13' 'R' nan 'PG' 'G' 'Passed' 'TV-14' '16' 'TV-MA'
 'Unrated' 'GP' 'Approved' 'TV-PG' 'U/A']
```

[ ]:
```
#Verificando os valores únicos de cada coluna
release_pg = df[df['Released_Year']=='PG']
```

```
[ ]: #Como o filme foi rodado em 1995, optamos por substituir essa informação
     df['Released_Year'] = df['Released_Year'].replace('PG', '1995')
```

```
[ ]: df.dtypes
```

```
[ ]: Series_Title      object
     Released_Year     object
     Certificate       object
     Runtime           object
     Genre             object
     IMDB_Rating      float64
     Overview          object
     Meta_score       float64
     Director          object
     Star1             object
     Star2             object
     Star3             object
     Star4             object
     No_of_Votes        int64
     Gross             object
     dtype: object
```

```
[ ]: # Verificando quantidade de linhas com zero
     (df['Released_Year'] == 0).sum()
```

```
[ ]: 0
```

```
[ ]: df['Gross'] = df['Gross'].str.replace(',', '').astype(float, errors='ignore')
     df['Runtime'] = df['Runtime'].str.replace(' min', '').astype(float)
     df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')
```

```
[ ]: df.dtypes
```

```
[ ]: Series_Title      object
     Released_Year      int64
     Certificate       object
     Runtime          float64
     Genre             object
     IMDB_Rating      float64
     Overview          object
     Meta_score       float64
     Director          object
     Star1             object
     Star2             object
     Star3             object
     Star4             object
     No_of_Votes        int64
```

```
Gross            float64
dtype: object
```

```
[ ]: #Converte a coluna 'Released_Year' em um formato de data e hora, mantendo␣
      ↪apenas o ano.
     df['Released_Year'] = pd.to_datetime(df['Released_Year'], format='%Y')
     df['Released_Year'] = df['Released_Year'].dt.year
```

```
[ ]: #Insere a média dos valores não ausentes na coluna 'Meta_score' e Gross e moda␣
      ↪na  Certicate
     df['Meta_score'].fillna(df['Meta_score'].mean(),inplace=True)
     df['Gross'].fillna(df['Gross'].mean(),inplace=True)
     df['Certificate'].fillna(df['Certificate'].mode()[0],inplace=True)
```

```
[ ]: # Analisando os tipos de variáveis
     df.dtypes
```

```
[ ]: Series_Title      object
     Released_Year      int32
     Certificate       object
     Runtime          float64
     Genre             object
     IMDB_Rating      float64
     Overview          object
     Meta_score       float64
     Director          object
     Star1             object
     Star2             object
     Star3             object
     Star4             object
     No_of_Votes        int64
     Gross            float64
     dtype: object
```

```
[ ]: #Tratando outliers
     variaveis = ['Gross']
```

```
[ ]: #Vamos usar IQR (Interquartile Range) para removê-los.
     def remove_outliers_iqr(df, column):
         Q1 = df[column].quantile(0.25)
         Q3 = df[column].quantile(0.75)
         IQR = Q3 - Q1
         lower_bound = Q1 - 1.5 * IQR
         upper_bound = Q3 + 1.5 * IQR
         return df[(df[column] > lower_bound) & (df[column] < upper_bound)]
```

[ ]:
```
# Aplicar a função remove_outliers_iqr a cada variável na lista
for column in variaveis:
    df = remove_outliers_iqr(df, column)
```

[ ]:
```
df.dtypes
```

[ ]:
```
Series_Title      object
Released_Year      int32
Certificate       object
Runtime          float64
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross            float64
dtype: object
```

[ ]:
```
df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')
```

[ ]:
```
# Após finalizado a etapa de limpeza e tratamento dos dados, vamos realizar a␣
 ↪EDA - EXPLORATORY DATA ANALYSIS
df.describe()
```

[ ]:
```
       Released_Year     Runtime  IMDB_Rating  Meta_score   No_of_Votes  \
count     883.000000  883.000000   883.000000  883.000000  8.830000e+02
mean     1989.430351  121.996602     7.938279   78.055665  2.111446e+05
std        23.747406   28.111718     0.260224   11.474185  2.517367e+05
min      1920.000000   45.000000     7.600000   28.000000  2.508800e+04
25%      1973.000000  102.000000     7.700000   72.000000  5.133750e+04
50%      1997.000000  118.000000     7.900000   77.969121  1.083990e+05
75%      2008.000000  135.000000     8.100000   86.000000  2.653800e+05
max      2020.000000  321.000000     9.200000  100.000000  1.854740e+06

              Gross
count  8.830000e+02
mean   3.900290e+07
std    3.804462e+07
min    1.305000e+03
25%    3.990500e+06
50%    2.826257e+07
75%    6.808257e+07
```

```
max    1.611978e+08
```

[ ]:
```
dfr = df.copy()
print(dfr.isnull().sum())
```

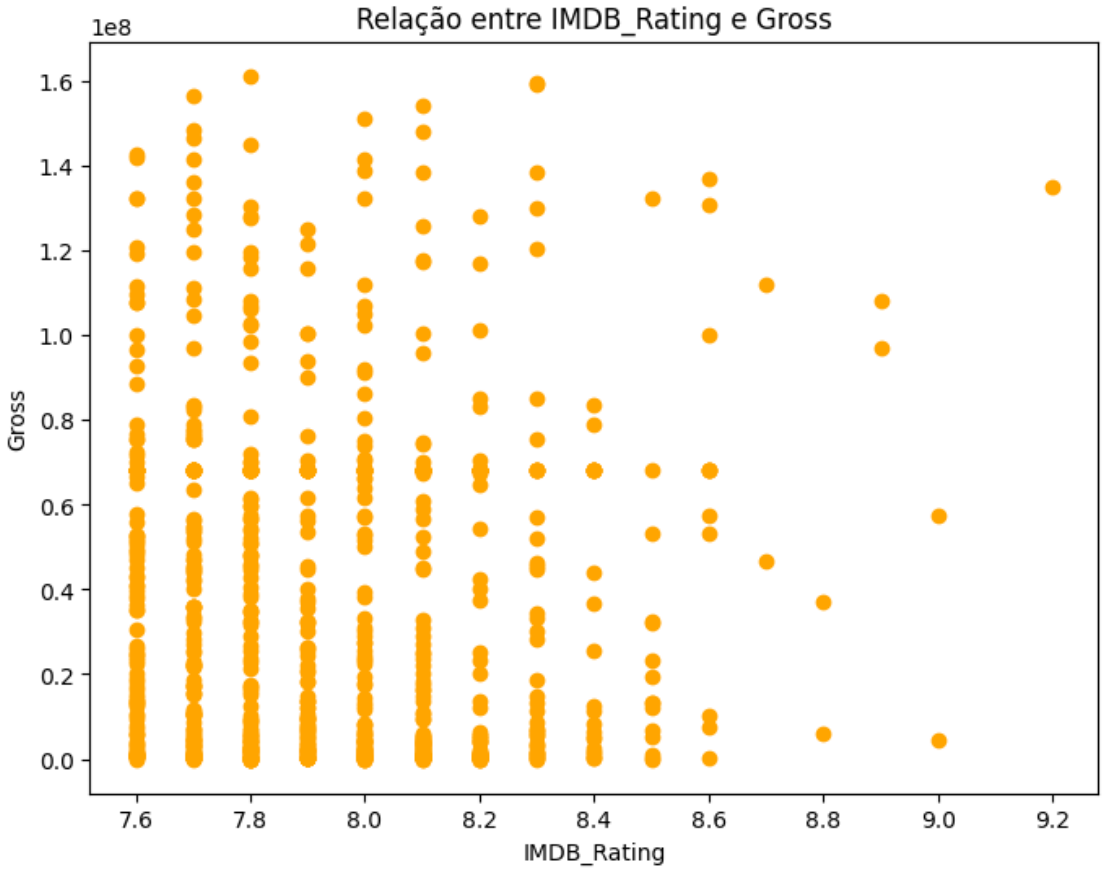
```
Series_Title     0
Released_Year    0
Certificate      0
Runtime          0
Genre            0
IMDB_Rating      0
Overview         0
Meta_score       0
Director         0
Star1            0
Star2            0
Star3            0
Star4            0
No_of_Votes      0
Gross            0
dtype: int64
```

[ ]:
```
# Analisando os tipos de variáveis
df.dtypes
```

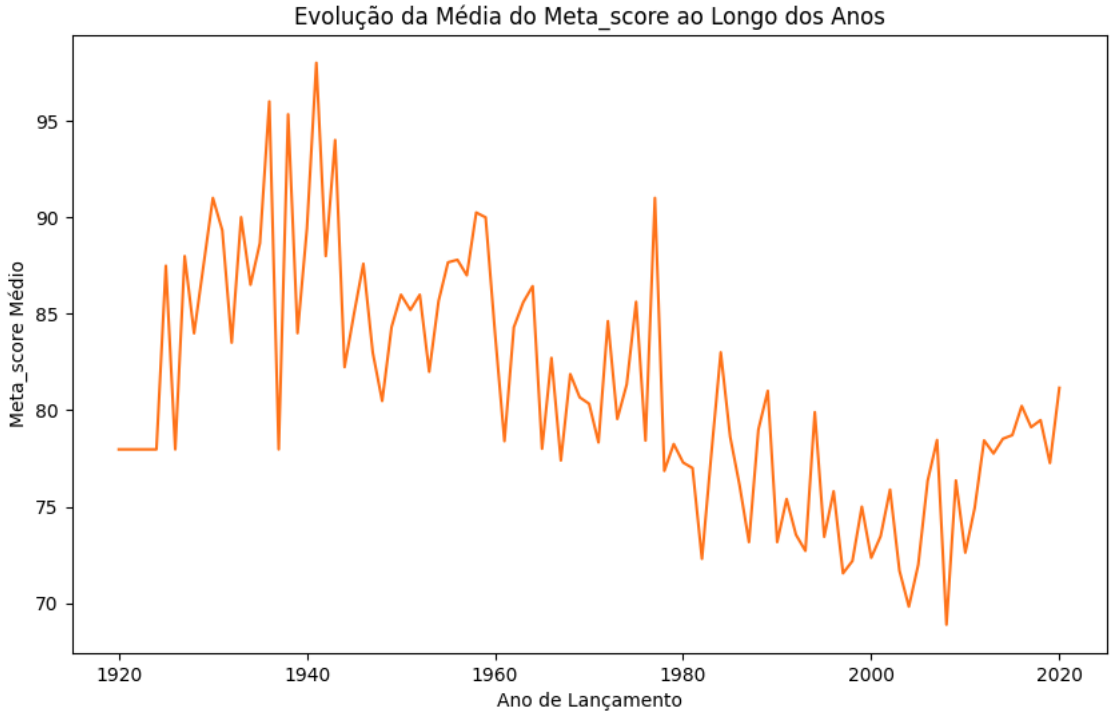
[ ]:
```
Series_Title      object
Released_Year      int32
Certificate       object
Runtime          float64
Genre             object
IMDB_Rating      float64
Overview          object
Meta_score       float64
Director          object
Star1             object
Star2             object
Star3             object
Star4             object
No_of_Votes        int64
Gross            float64
dtype: object
```

[ ]:
```
# Este gráfico mostra a relação entre a classificação do IMDB e a receita bruta␣
 ↪dos filmes
# Definindo a cor em tom de laranja
orange_color = "#FFA500"  # Laranja padrão
```

```
# Este gráfico mostra a relação entre a classificação do IMDB e a receita bruta␣
  ↪dos filmes
plt.figure(figsize=(8, 6))
plt.scatter(df['IMDB_Rating'], df['Gross'], color=orange_color)
plt.title('Relação entre IMDB_Rating e Gross')
plt.xlabel('IMDB_Rating')
plt.ylabel('Gross')
plt.show()
```



[ ]:
```
#Este gráfico de linha mostra a evolução da média da pontuação do Meta Score ao␣
  ↪longo dos anos de lançamento.
plt.figure(figsize=(10, 6))
df.groupby('Released_Year')['Meta_score'].mean().plot(kind='line',␣
  ↪color='#fe7218')
plt.title('Evolução da Média do Meta_score ao Longo dos Anos')
plt.xlabel('Ano de Lançamento')
plt.ylabel('Meta_score Médio')
plt.show()
```



[ ]:
```
#Chamando a função para criação do nosso relatório EDA
profile = ProfileReport(df, title = "Profiling Report")
```

[ ]:
```
#Visualizando o profile
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[ ]:

** INFORMAÇÕES SOBRE O DATASET

- Desafios:

1. O dataset possuiam 999 linhas e 16 colunas
2. Os dados se dividiam em float, int e object
3. Encontramos valores ausentes nas colunas Certificate, MetaScore e Gross.
4. Foi identificado presença dado com informação divergente na coluna 5. Released_Year
5. Haviam colunas com dados categóricos e quantitativos
6. Também identificamos vírgulas nos numerais
7. Encontramos outliers na coluna gross

- Ações tomadas:

1. Notamos que a coluna Unnamed não agrega ao estudo, portanto, vamos eliminá-la
2. Na coluna Released_Year, tratamos com a inclusão da informação correta - referente ao ano de lançamento
3. Inserimos a média dos valores ausentes na coluna gross e meta score.
4. Na certificate inserimos a moda, ou seja o valor mais comum à variável.
5. Converte a coluna 'Released_Year' em um formato de data e hora, mantendo apenas o ano.
6. Extraímos valores categóricos da coluna runtime, excluindo as vírgulas da variável gross
7. Transformamos a variável gross de object para float
8. Tratamos outlier na coluna gross

Produto A: EDA

[ ]:
```
#Baixando o EDA para html
profile.to_file('EDAFINAL.html')
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

Produto B: Achados e insights

- Qual filme você recomendaria para uma pessoa que você não conhece? *texto em itálico*

Eu usaria duas estratégias: ou indicaria o filme com a melhor classificação ou o filme com o maior número de votos, sendo assim:

Maior nota do IMDB: Filme: The Godfather, Nota: 9.2

Filme com o maior número de votos: Filme: Fight Club, Número de Votos: 1854740

[ ]:
```
# Encontrar a maior nota do IMDB
maior_imdb_rating = df['IMDB_Rating'].max()
filme_maior_imdb = df[df['IMDB_Rating'] == maior_imdb_rating]

# Encontrar o filme com o maior número de votos
indice_max_votos = df['No_of_Votes'].idxmax()
filme_maior_votos = df.loc[indice_max_votos]

# Exibir informações dos filmes encontrados
print("Maior nota do IMDB:")
print(f"Filme: {filme_maior_imdb['Series_Title'].values[0]}, Nota:␣
 ↪{maior_imdb_rating}")

print("\nFilme com o maior número de votos:")
print(f"Filme: {filme_maior_votos['Series_Title']}, Número de Votos:␣
 ↪{filme_maior_votos['No_of_Votes']}")
```

```
Maior nota do IMDB:
Filme: The Godfather, Nota: 9.2

Filme com o maior número de votos:
Filme: Fight Club, Número de Votos: 1854740
```

Quais são os principais fatores que estão relacionados com alta expectativa de faturamento de um filme?

Conforme observado no EDA - a maior correlação de faturamento de um filme é ("Relação entre notas IMDB e Metascore e faturamento"), o Metascore pode influenciar a receita bruta, mesmo sem haver correlação linear entre Metascore e Gross, como visto na análise de correlação.

Quais insights podem ser tirados com a coluna Overview? É possível inferir o gênero do filme a partir dessa coluna?

Alguns caminhos a partir da coluna Overwiew:

- "young" (jovem): Insight: A presença da palavra "young" sugere que histórias envolvendo personagens jovens ou temas relacionados à juventude são populares. Você pode considerar criar um filme que explore a vida de jovens em diferentes contextos, como amizade, amor, desafios e crescimento pessoal.
- "man" (homem) e "woman" (mulher): Insight: A frequência de "man" e "woman" indica que personagens masculinos e femininos são centrais nas narrativas. Isso sugere a possibilidade de explorar dinâmicas de gênero, relacionamentos ou até mesmo histórias de empoderamento. Um enredo que desafie estereótipos de gênero pode ressoar bem com o público.
- "two" (dois): Insight: A palavra "two" pode indicar histórias que envolvem duplas ou relacionamentos. Isso pode ser explorado em um filme que foca em parcerias, como amizade, amor ou rivalidade, onde a interação entre dois personagens é central para a narrativa.
- "life" (vida): Insight: "Life" é uma palavra poderosa que sugere temas de existência, experiências e reflexões. Um filme que explore a jornada da vida, com seus altos e baixos, pode ser atraente. Isso pode incluir histórias de superação, autodescoberta ou a busca por propósito.
- "world" (mundo): Insight: A menção a "world" sugere a possibilidade de explorar temas globais ou universais. Você pode considerar um enredo que aborde questões sociais, culturais ou ambientais, mostrando como diferentes personagens interagem com o mundo ao seu redor.
- "new" (novo): Insight: A palavra "new" pode indicar a introdução de novos conceitos, ideias ou personagens. Isso pode ser uma oportunidade para criar uma narrativa que desafie o status quo ou que introduza um novo elemento no mundo do filme.
- "story" (história): Insight: A ênfase em "story" sugere que o enredo é fundamental. Um filme que se concentre em narrativas envolventes e bem construídas, talvez com várias camadas ou reviravoltas, pode atrair o público.
- "war" (guerra): Insight: A presença de "war" pode indicar interesse em histórias de conflito, heroísmo ou drama humano em tempos de crise. Um filme que explore as consequências da guerra ou a luta pela paz pode ser impactante.
- "love" (amor): Insight: "Love" é um tema universal que ressoa com muitos públicos. Um enredo romântico ou que explore diferentes formas de amor (familiar, platônico, romântico) pode ser muito atrativo.

Sugestões de Roteiro

- Drama Juvenil: Um filme que segue a vida de um grupo de jovens amigos enfrentando desafios em um mundo em mudança, explorando temas de amor, amizade e autodescoberta.

- Romance com Conflito: Uma história de amor entre um homem e uma mulher de mundos diferentes, que deve superar barreiras sociais ou culturais, com um pano de fundo de guerra ou conflito.
- Aventura de Dupla: Um filme que segue duas pessoas em uma jornada para descobrir algo novo sobre si mesmas e o mundo, enfrentando desafios e construindo um vínculo forte ao longo do caminho.
- Reflexão sobre a Vida: Uma narrativa que explora a vida de um personagem ao longo de várias décadas, refletindo sobre suas experiências, amores e as lições aprendidas.
- História de Superação: Um filme que aborda a luta de um jovem para encontrar seu lugar no mundo, lidando com questões de identidade, aceitação e amor.

[ ]:
```
#Definindo um novo dataset  para extrair o melhor da variável overview
#Padronizar a coluna overview
df['Overview']= df['Overview'].str.lower()
df['Overview'] = df['Overview'].str.replace('[^\w\s]','')
```

[ ]:
```
nltk.download('stopwords')
stop = set(stopwords.words("english"))

def processamento_texto(text):
    filtered_words = [word.lower() for word in text.split() if word.lower() not␣
 ↪in stop]
    return " ".join(filtered_words)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
df['Overview']= df.Overview.map(processamento_texto)
```

[ ]:
```
# Download the stopwords corpus
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

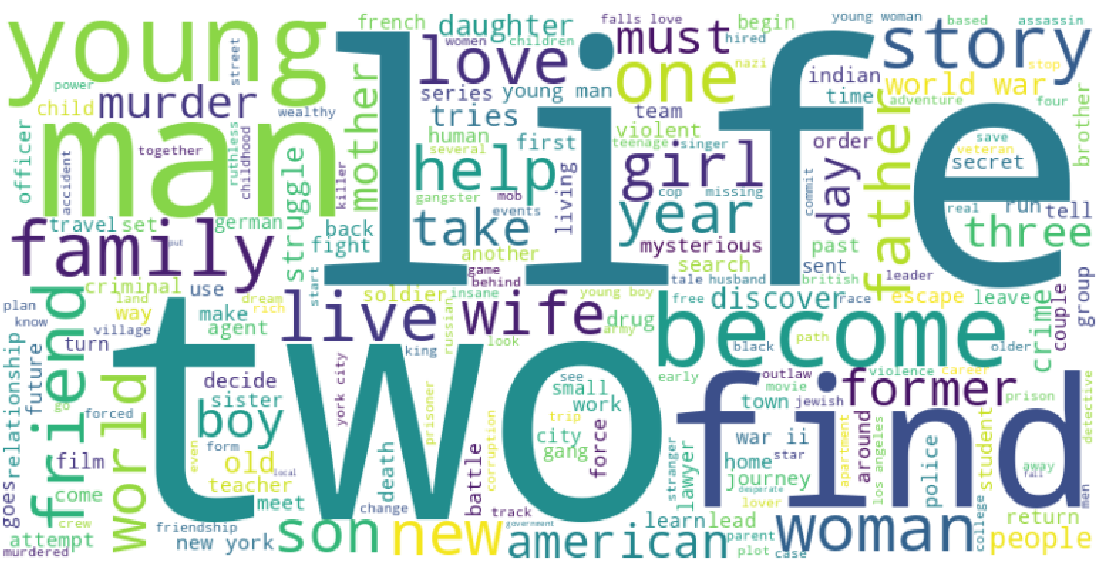
[ ]:
```
True
```

[ ]:
```
# Tokenização e remoção de stopwords
stop_words = set(stopwords.words('english'))
all_words = []

for overview in df['Overview']:
    # Tokenizar
    words = word_tokenize(overview.lower())

    # Converte para minúsculas
```

```
    # Filtrar stopwords
    filtered_words = [word for word in words if word.isalnum() and word not in stop_words]
    all_words.extend(filtered_words)
```

[ ]:
```
# Criar uma nuvem de palavras
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(all_words))
```

[ ]:
```
# Exibir a nuvem de palavras
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



[ ]:
```
# Calcular a frequência das palavras
word_freq = pd.Series(all_words).value_counts()
```

[ ]:
```
# Extrair as palavras mais relevantes
top_words = word_freq.head(10)
print("Palavras mais relevantes:")
print(top_words)
```

```
Palavras mais relevantes:
young    122
man      108
two       99
life      95
```

```
world     67
new       62
woman     61
story     60
war       58
love      57
Name: count, dtype: int64
```

DATASET ADICIONAL OBS: #Para saber mais sobre esse dataset consulte: https://www.kaggle.com/datasets/luisfredgs/imdb-ptbr?select=imdb-reviews-pt-br.csv #OBS: Não consegui enviar a base de dados para o github devido ao tamanho. (infelizmente)

[ ]:
```
#Para tornar a nossa análise mais avançada, foi inserido nesse estudo um␣
 ↪conjunto de dados cedidos pelo IMDb com avaliações e classicadas em␣
 ↪sentimentos positivos e negativos.
df_sentiment = pd.read_csv("/content/drive/MyDrive/Indicium/imdb-reviews-pt-br.
 ↪csv")
```

[ ]:
```
df_sentiment
```

[ ]:
```
           id                                            text_en  \
0           1  Once again Mr. Costner has dragged out a movie…
1           2  This is an example of why the majority of acti…
2           3  First of all I hate those moronic rappers, who…
3           4  Not even the Beatles could write songs everyon…
4           5  Brass pictures movies is not a fitting word fo…
…         …                                                …
49454   49456  Seeing as the vote average was pretty low, and…
49455   49457  The plot had some wretched, unbelievable twist…
49456   49458  I am amazed at how this movieand most others h…
49457   49459  A Christmas Together actually came before my t…
49458   49460  Working-class romantic drama from director Mar…

                                              text_pt sentiment
0      Mais uma vez, o Sr. Costner arrumou um filme p…       neg
1      Este é um exemplo do motivo pelo qual a maiori…       neg
2      Primeiro de tudo eu odeio esses raps imbecis, …       neg
3      Nem mesmo os Beatles puderam escrever músicas …       neg
4      Filmes de fotos de latão não é uma palavra apr…       neg
…                                                    …         …
49454  Como a média de votos era muito baixa, e o fat…       pos
49455  O enredo teve algumas reviravoltas infelizes e…       pos
49456  Estou espantado com a forma como este filme e …       pos
49457  A Christmas Together realmente veio antes do m…       pos
49458  O drama romântico da classe trabalhadora do di…       pos
```

```
[49459 rows x 4 columns]
```

```
[ ]: df_sentiment.drop(columns=['id'], inplace=True)
```

```
[ ]: df_sentiment.dtypes
```

```
[ ]: text_en      object
     text_pt      object
     sentiment    object
     dtype: object
```

```
[ ]: df_sentiment.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49459 entries, 0 to 49458
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   text_en    49459 non-null  object
 1   text_pt    49459 non-null  object
 2   sentiment  49459 non-null  object
dtypes: object(3)
memory usage: 1.1+ MB
```

```
[ ]: df_sentiment.query("sentiment == 'pos'").shape[0]
```

```
[ ]: 24694
```

```
[ ]: df_sentiment.query("sentiment == 'neg'").shape[0]
```

```
[ ]: 24765
```

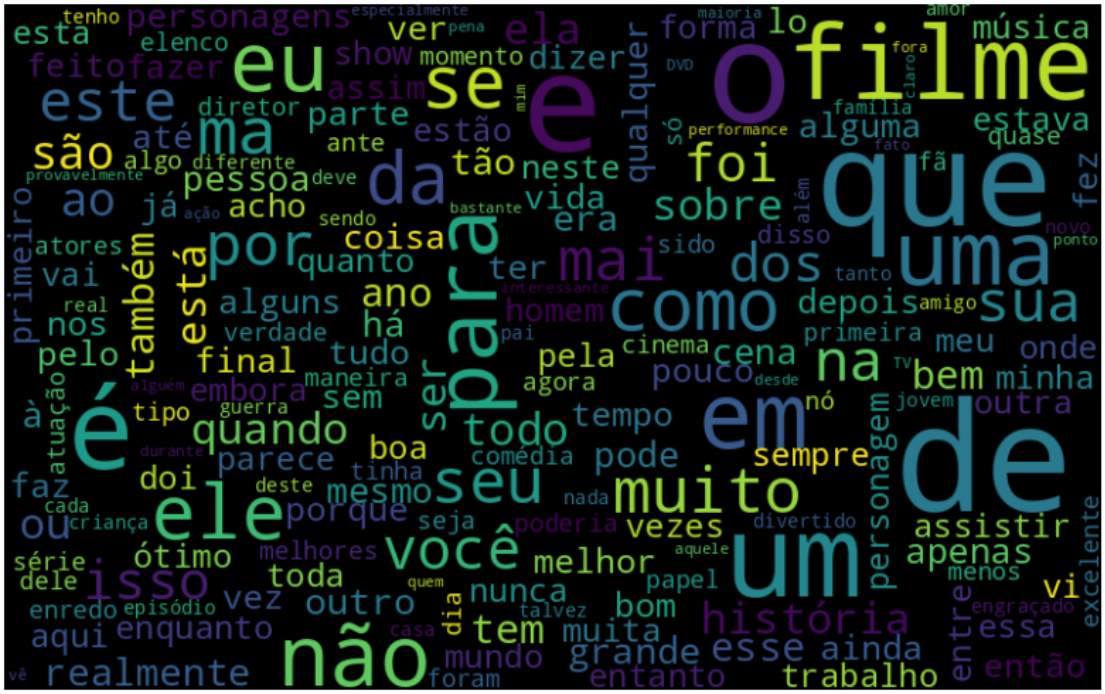
```
[ ]: positivo_total = df_sentiment.query("sentiment == 'pos'").sample(6810)
     negativo_total = df_sentiment.query("sentiment == 'neg'")
     df_sentiment = pd.concat([positivo_total, negativo_total])
     df_sentiment.shape
```

```
[ ]: (31575, 3)
```

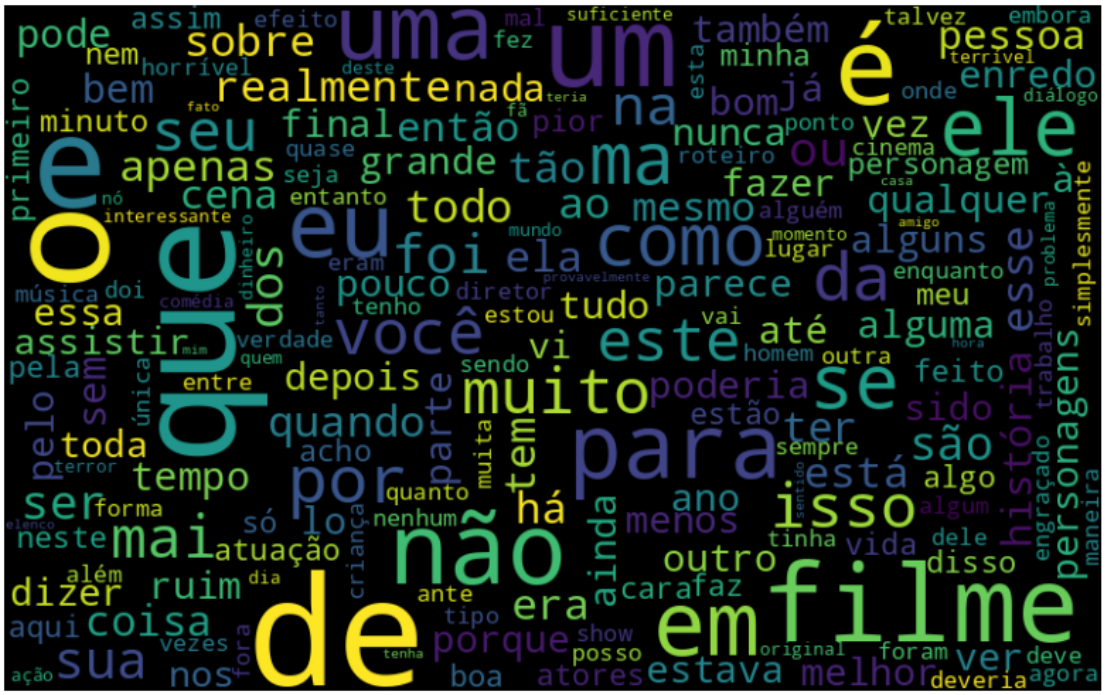
```
[ ]: def generate_wordcloud(dataframe: pd.DataFrame, text_column: str):
         all_words = ' '.join([ text for text in dataframe[text_column] ])
         word_cloud = \
             WordCloud(width=800, height=500,
                       max_font_size=110, collocations=False).generate(all_words)
         plt.figure(figsize=(10,7))
         plt.imshow(word_cloud, interpolation='bilinear')
         plt.axis('off')
```

```
    plt.show()
```

[ ]:
```
generate_wordcloud(df_sentiment.query("sentiment == 'pos'"), "text_pt")
```

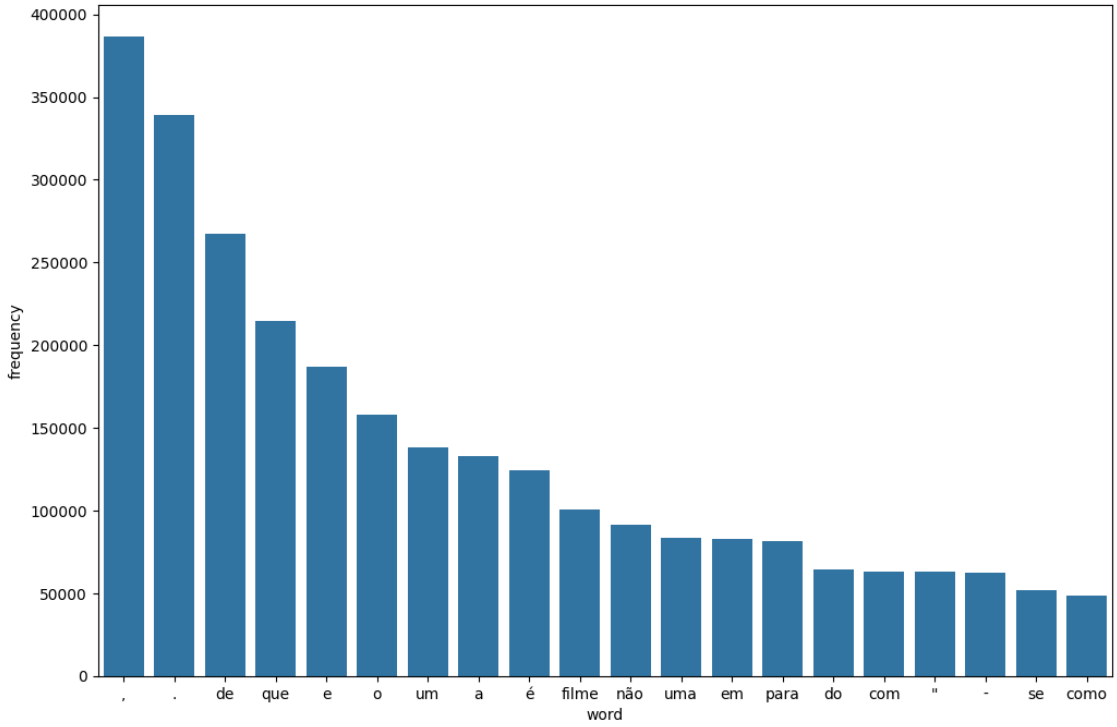


[ ]:
```
generate_wordcloud(df_sentiment.query("sentiment == 'neg'"), "text_pt")
```



[ ]:
```
def word_frequencies_graphic(tokenized_data, n_largest):
    frequency = nltk.FreqDist(tokenized_data)
    frequency_df = pd.DataFrame({
        "word": list(frequency.keys()),
        "frequency": list(frequency.values())
    })
    plt.figure(figsize=(12, 8))
    ax = sns.barplot(
        data = frequency_df.nlargest(columns = "frequency", n = n_largest),
        x = "word",
        y = "frequency"
    )
    plt.show()
```

[ ]:
```
tokenized_data = tokenizer(df_sentiment, "text_pt", False)
word_frequencies_graphic(tokenized_data, 20)
```

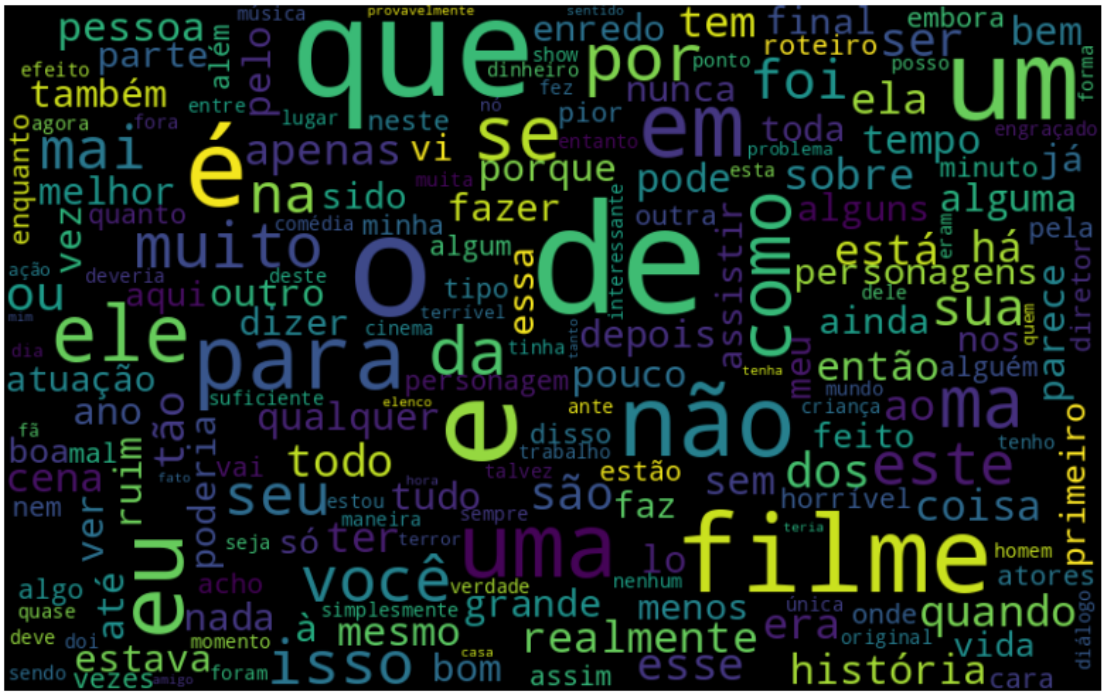


[ ]:
```
!pip install unidecode
import string
import unidecode


puncts = [ punct for punct in string.punctuation ]
stopwords = list(set([ unidecode.unidecode(stopword) for stopword in nltk.
 ↪corpus.stopwords.words("portuguese")]))
stopwords_puncts = sorted(stopwords + puncts)
print(stopwords_puncts)
```

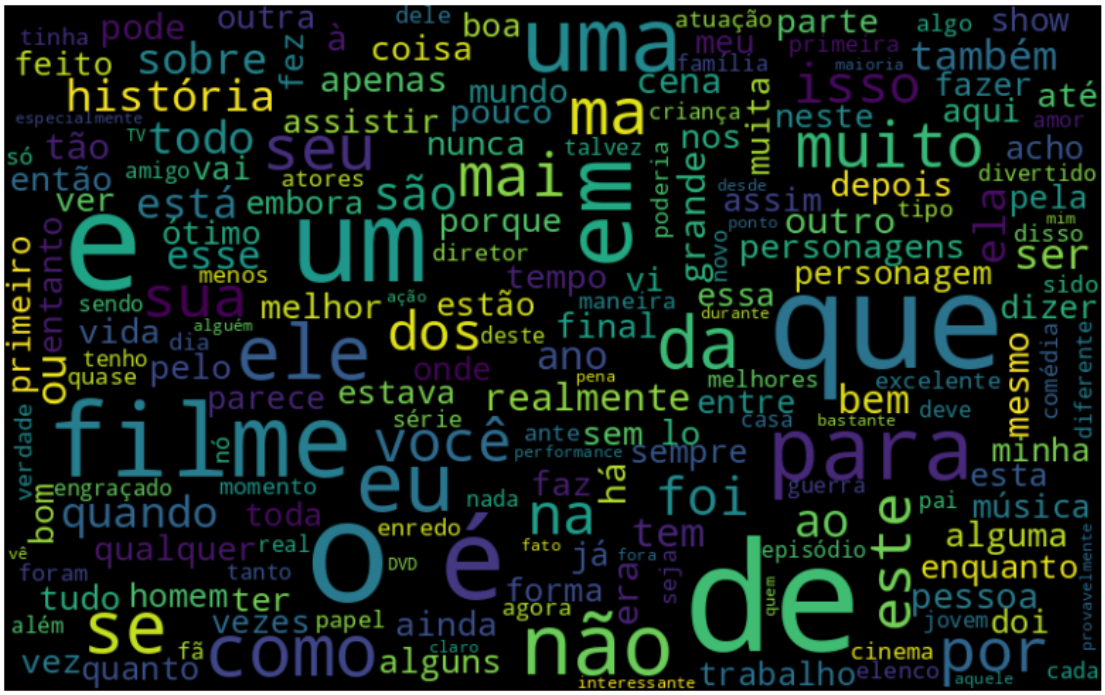
```
Requirement already satisfied: unidecode in /usr/local/lib/python3.10/dist-
packages (1.3.8)
['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':',
';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', 'a', 'ao', 'aos',
'aquela', 'aquelas', 'aquele', 'aqueles', 'aquilo', 'as', 'ate', 'com', 'como',
'da', 'das', 'de', 'dela', 'delas', 'dele', 'deles', 'depois', 'do', 'dos', 'e',
'ela', 'elas', 'ele', 'eles', 'em', 'entre', 'era', 'eram', 'eramos', 'essa',
'essas', 'esse', 'esses', 'esta', 'estamos', 'estao', 'estar', 'estas',
'estava', 'estavam', 'estavamos', 'este', 'esteja', 'estejam', 'estejamos',
'estes', 'esteve', 'estive', 'estivemos', 'estiver', 'estivera', 'estiveram',
'estiveramos', 'estiverem', 'estivermos', 'estivesse', 'estivessem',
'estivessemos', 'estou', 'eu', 'foi', 'fomos', 'for', 'fora', 'foram',
'foramos', 'forem', 'formos', 'fosse', 'fossem', 'fossemos', 'fui', 'ha',
```

```
'haja', 'hajam', 'hajamos', 'hao', 'havemos', 'haver', 'hei', 'houve',
'houvemos', 'houver', 'houvera', 'houveram', 'houveramos', 'houverao',
'houverei', 'houverem', 'houveremos', 'houveria', 'houveriam', 'houveriamos',
'houvermos', 'houvesse', 'houvessem', 'houvessemos', 'isso', 'isto', 'ja',
'lhe', 'lhes', 'mais', 'mas', 'me', 'mesmo', 'meu', 'meus', 'minha', 'minhas',
'muito', 'na', 'nao', 'nas', 'nem', 'no', 'nos', 'nossa', 'nossas', 'nosso',
'nossos', 'num', 'numa', 'o', 'os', 'ou', 'para', 'pela', 'pelas', 'pelo',
'pelos', 'por', 'qual', 'quando', 'que', 'quem', 'sao', 'se', 'seja', 'sejam',
'sejamos', 'sem', 'ser', 'sera', 'serao', 'serei', 'seremos', 'seria', 'seriam',
'seriamos', 'seu', 'seus', 'so', 'somos', 'sou', 'sua', 'suas', 'tambem', 'te',
'tem', 'temos', 'tenha', 'tenham', 'tenhamos', 'tenho', 'tera', 'terao',
'terei', 'teremos', 'teria', 'teriam', 'teriamos', 'teu', 'teus', 'teve',
'tinha', 'tinham', 'tinhamos', 'tive', 'tivemos', 'tiver', 'tivera', 'tiveram',
'tiveramos', 'tiverem', 'tivermos', 'tivesse', 'tivessem', 'tivessemos', 'tu',
'tua', 'tuas', 'um', 'uma', 'voce', 'voces', 'vos', '{', '|', '}', '~']
```

[ ]:
```
generate_wordcloud(df_sentiment.query("sentiment == 'neg'"), "text_pt")
```



[ ]:
```
generate_wordcloud(df_sentiment.query("sentiment == 'pos'"), "text_pt")
```



Insights dos sentimentos positivos e negativos

Negativos

- Desenvolver uma história original e cativante: Evite clichês e tramas previsíveis. Crie reviravoltas inesperadas e um arco de personagem convincente. Focar em um enredo bem estruturado: Certifique-se de que a história tenha um início, meio e fim claros, com transições suaves entre as cenas.
- Evitar buracos na trama ou inconsistências: Examine o enredo de perto para garantir que tudo faça sentido e que não haja lacunas lógicas. Insights sobre a Atuação e os Personagens
- Criar personagens multidimensionais e realistas: Evite estereótipos e dê aos personagens profundidade emocional e motivações convincentes. Garantir que os diálogos soem naturais e relevantes: Certifique-se de que os diálogos avancem a trama e revelem informações sobre os personagens.
- Treinar os atores adequadamente e dar-lhes direção clara: Certifique-se de que os atores entendam seus personagens e possam transmitir as emoções corretamente.
- Garantir uma direção coesa e uma visão clara para o filme: O diretor deve ter uma compreensão sólida da história e transmitir isso efetivamente para a equipe. Escolher atores que se encaixem perfeitamente em seus papéis: A química entre os atores e a credibilidade de seus personagens são essenciais.

Positivos

- Crie uma narrativa envolvente: O enredo deve ser intrigante e manter o público interessado. Considere estruturas narrativas que incluam reviravoltas, conflitos e resoluções satisfatórias.

- Desenvolvimento de temas universais: Histórias que abordam temas como amor, amizade, superação ou questões sociais tendem a ressoar mais com o público. Pense em como seu enredo pode explorar esses temas de maneira única.
- Escolha de elenco: A atuação é crucial para a credibilidade do filme. Certifique-se de que os atores escolhidos possam trazer profundidade e autenticidade aos seus personagens.
- Direção de atores: Forneça direções claras e construtivas para os atores, ajudando-os a entender as motivações de seus personagens e a construir performances memoráveis.
- Crie personagens complexos: Personagens bem desenvolvidos com motivações claras e arcos de crescimento são mais cativantes. Considere dar a cada personagem uma jornada pessoal que se entrelaça com a narrativa principal. Diversidade e representação: Personagens de diferentes origens, etnias e experiências de vida podem enriquecer a história e torná-la mais inclusiva. Isso pode atrair um público mais amplo.
- Estilo de direção coeso: A visão do diretor deve ser clara e consistente ao longo do filme. Isso inclui a escolha de ângulos de câmera, ritmo e estilo visual que complementem a história.
- Trabalho em equipe: A colaboração entre o diretor, roteirista e equipe de produção é essencial para garantir que todos os elementos do filme estejam alinhados com a visão criativa.
- Trilha sonora impactante: A música pode evocar emoções e realçar momentos-chave da narrativa. Considere como a trilha sonora pode ser usada para intensificar a experiência do espectador. Integração com a narrativa: A música deve complementar a história e os personagens, ajudando a contar a história de maneira mais eficaz.
- Humor apropriado: Se o seu filme permitir, incluir elementos engraçados pode torná-lo mais acessível e divertido. Certifique-se de que o humor se encaixe no tom geral do filme.
- Momentos leves em meio ao drama: Mesmo em histórias sérias, momentos de alívio cômico podem ajudar a equilibrar a narrativa e proporcionar uma experiência mais rica para o público.

0.4 4. Modelagem (Modeling)

A fase de modelagem envolve a aplicação de técnicas e algoritmos de modelagem de dados aos dados preparados. Selecionamos as técnicas mais adequadas, como regressão, classificação ou agrupamento, e ajustamos e avaliamos os modelos para garantir sua precisão e eficácia.

Produto C: Modelagem Preditiva

#Variáveis e transformação

- Inicialmente, a base de dados foi dividida em dois subconjuntos distintos: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento foi utilizado para ajustar e calibrar os modelos preditivos, enquanto o conjunto de teste serviu para avaliar o desempenho desses modelos treinados. Essa abordagem de divisão dos dados foi crucial para garantir que os resultados obtidos fossem generalizáveis e para evitar o problema de overfitting, no qual o modelo se ajusta excessivamente aos dados de treinamento, perdendo sua capacidade de generalização. Removemos as colunas 'Series_Title', 'Released_Year', 'Runtime', 'Overview', 'Star1', 'Star2', 'Star3', 'Star4' porque entendemos que não agregavam ao modelo.
- Avaliação da Performance dos Modelos A performance dos modelos de regressão foi avaliada com base em métricas recomendadas pela literatura científica relevante, incluindo: Coefi-

ciente de Determinação (R^2): Indicava a proporção da variabilidade na variável dependente que era explicada pelas variáveis independentes no modelo. Erro Quadrático Médio (MSE): Mediu a média dos quadrados dos erros de previsão, penalizando erros maiores de forma mais acentuada. Raiz do Erro Quadrático Médio (RMSE): Forneceu a raiz quadrada da média dos quadrados dos erros de previsão, apresentando os erros na mesma escala da variável dependente. Erro Absoluto Médio (MAE): Forneceu a média dos valores absolutos dos erros de previsão, oferecendo uma medida mais intuitiva dos erros. Essas métricas permitiram analisar a capacidade preditiva dos modelos e selecionar aquele que apresentava o melhor desempenho geral.

- Variáveis Utilizadas As variáveis selecionadas para a modelagem incluíam: released_year: Ano de lançamento do filme certificate: Classificação indicativa, após transformação em variáveis dummy runtime: Duração do filme genre: Gênero do filme, após transformação em variáveis dummy meta_score: Pontuação do filme no Metacritic no_of_votes: Número de votos recebidos gross: Receita bruta do filme

- Essas variáveis foram escolhidas com base em uma análise exploratória prévia dos dados, realizada durante a etapa de pré-processamento, transformação e análise exploratória da base de dados. Nessa etapa, foram aplicadas as transformações necessárias para preparar os dados para a modelagem, como a conversão da variável runtime para um formato numérico adequado, o tratamento de valores ausentes e a normalização de variáveis, quando apropriado. Ao empregar esses modelos de regressão e as variáveis selecionadas, esperava-se obter previsões precisas da nota do IMDb, contribuindo para uma compreensão mais profunda dos fatores que influenciam a avaliação dos filmes pelos usuários.

0.5 Tipo de Problema

- Estamos lidando com um problema de regressão, pois o objetivo é prever uma variável contínua: a nota do IMDb dos filmes. A regressão é apropriada aqui, uma vez que estamos tentando estimar valores numéricos com base em diferentes características dos filmes.

0.6 O Modelo

- O modelo que apresentou o melhor desempenho na métrica de RMSE foi o Random Forest, pois possui menores valores de MSE e RMSE, além de um R^2 mais alto, indicando que ele é mais eficaz em capturar a variância nos dados. Regressão Linear teve um desempenho razoável, mas não conseguiu explicar bem a variância dos dados, como indicado pelo R^2 baixo. Rede Neural apresentou resultados variáveis dependendo da configuração, mas a configuração com 40 neurônios foi a mais promissora. No entanto, o desempenho geral ainda não superou o Random Forest.

- Vantagens do Random Forest Robustez e Estabilidade: O Random Forest é um modelo de aprendizado de máquina que combina múltiplas árvores de decisão, o que o torna menos suscetível a variações nos dados de entrada. Isso resulta em previsões mais estáveis e confiáveis. Capacidade de Lidar com Dados Não Lineares: O modelo é eficaz em capturar relações não lineares entre as variáveis, o que é uma vantagem em muitos conjuntos de dados complexos. Importância das Variáveis: O Random Forest fornece uma medida de importância das variáveis, permitindo identificar quais fatores têm maior impacto nas previsões. Isso pode ser útil para interpretação e análise de dados. Menos Necessidade de Pré-processamento: O modelo pode lidar com dados faltantes e não requer normalização ou padronização das variáveis,

o que simplifica o processo de preparação dos dados.

- Desvantagens do Random Forest Complexidade e Tempo de Treinamento: Embora o Random Forest seja robusto, ele pode ser mais lento para treinar em comparação com modelos mais simples, especialmente em conjuntos de dados muito grandes. Menos Interpretável: Apesar de fornecer informações sobre a importância das variáveis, o Random Forest é considerado menos interpretável do que modelos simples, como a regressão linear. A complexidade do modelo pode dificultar a compreensão de como as previsões são feitas. Tendência ao Overfitting em Conjuntos Pequenos: Em conjuntos de dados pequenos, o Random Forest pode se ajustar excessivamente aos dados de treinamento, levando a um desempenho inferior em dados não vistos. Conclusão O Random Forest é uma escolha poderosa para problemas de regressão, especialmente quando se busca robustez e capacidade de lidar com dados complexos. No entanto, suas desvantagens, como a complexidade e o tempo de treinamento, devem ser consideradas ao escolher o modelo mais apropriado para uma tarefa específica.

[ ]:
```
#Fizemos uma cópia do DataFrame df e a armazenamos em uma nova variável e␣
 ↪também verificamos se havia ficado algum valor nulo
dfr = df.copy()
print(dfr.isnull().sum())
```

```
Series_Title     0
Released_Year    0
Certificate      0
Runtime          0
Genre            0
IMDB_Rating      0
Overview         0
Meta_score       0
Director         0
Star1            0
Star2            0
Star3            0
Star4            0
No_of_Votes      0
Gross            0
dtype: int64
```

[ ]:
```
print(dfr.columns)
```

```
Index(['Series_Title', 'Released_Year', 'Certificate', 'Runtime', 'Genre',
       'IMDB_Rating', 'Overview', 'Meta_score', 'Director', 'Star1', 'Star2',
       'Star3', 'Star4', 'No_of_Votes', 'Gross'],
      dtype='object')
```

[ ]:
```
#As variáveis não deve apresentar relevância para o modelo por isso serão␣
 ↪removidas
dfr = dfr.drop(columns=['Series_Title', 'Released_Year','Runtime', 'Overview',␣
 ↪'Star1', 'Star2', 'Star3', 'Star4'])
```

[ ]:
```
# Confirmando as nossas variáveis no dataset
dfr.info( )
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 883 entries, 0 to 998
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Certificate  883 non-null    object
 1   Genre        883 non-null    object
 2   IMDB_Rating  883 non-null    float64
 3   Meta_score   883 non-null    float64
 4   Director     883 non-null    object
 5   No_of_Votes  883 non-null    int64
 6   Gross        883 non-null    float64
dtypes: float64(3), int64(1), object(3)
memory usage: 87.5+ KB
```

[ ]:
```
#Converte as colunas categóricas 'Certificate', 'Genre' e 'Director' em valores␣
 ↪numéricos usando LabelEncoder.
# Adicionando 1 a cada valor numérico para que os valores comecem em 1 para␣
 ↪facilitar a interpretação do algoritmo
# Criando dicionários de mapeamento para cada coluna, para saber qual número␣
 ↪corresponde a cada categoria original.
label_encoder = LabelEncoder()
dfr['Certificate'] = label_encoder.fit_transform(dfr['Certificate']) + 1
certificate_mapping = dict(zip(label_encoder.classes_, label_encoder.
 ↪transform(label_encoder.classes_) + 1))
dfr['Genre'] = label_encoder.fit_transform(dfr['Genre']) + 1
genre_mapping = dict(zip(label_encoder.classes_, label_encoder.
 ↪transform(label_encoder.classes_) + 1))
dfr['Director'] = label_encoder.fit_transform(dfr['Director']) + 1
director_mapping = dict(zip(label_encoder.classes_, label_encoder.
 ↪transform(label_encoder.classes_) + 1))
```

[ ]:
```
#Observando o nosso mapeamento
print("Mapeamento para 'Certificate':")
print(certificate_mapping)
print("\nMapeamento para 'Genre':")
print(genre_mapping)
print("\nMapeamento para 'Director':")
print(director_mapping)
```

```
Mapeamento para 'Certificate':
{'16': 1, 'A': 2, 'Approved': 3, 'G': 4, 'GP': 5, 'PG': 6, 'PG-13': 7, 'Passed':
8, 'R': 9, 'TV-14': 10, 'TV-MA': 11, 'TV-PG': 12, 'U': 13, 'U/A': 14, 'UA': 15,
'Unrated': 16}
```

Mapeamento para 'Genre':
{'Action, Adventure': 1, 'Action, Adventure, Biography': 2, 'Action, Adventure, Comedy': 3, 'Action, Adventure, Crime': 4, 'Action, Adventure, Drama': 5, 'Action, Adventure, Fantasy': 6, 'Action, Adventure, History': 7, 'Action, Adventure, Horror': 8, 'Action, Adventure, Romance': 9, 'Action, Adventure, Sci-Fi': 10, 'Action, Adventure, Thriller': 11, 'Action, Adventure, War': 12, 'Action, Adventure, Western': 13, 'Action, Biography, Crime': 14, 'Action, Biography, Drama': 15, 'Action, Comedy, Crime': 16, 'Action, Comedy, Fantasy': 17, 'Action, Comedy, Mystery': 18, 'Action, Comedy, Romance': 19, 'Action, Crime, Comedy': 20, 'Action, Crime, Drama': 21, 'Action, Crime, Mystery': 22, 'Action, Crime, Thriller': 23, 'Action, Drama': 24, 'Action, Drama, History': 25, 'Action, Drama, Mystery': 26, 'Action, Drama, Sci-Fi': 27, 'Action, Drama, Sport': 28, 'Action, Drama, Thriller': 29, 'Action, Drama, War': 30, 'Action, Drama, Western': 31, 'Action, Mystery, Thriller': 32, 'Action, Sci-Fi': 33, 'Action, Sci-Fi, Thriller': 34, 'Action, Thriller': 35, 'Adventure, Biography, Crime': 36, 'Adventure, Biography, Drama': 37, 'Adventure, Comedy, Crime': 38, 'Adventure, Comedy, Drama': 39, 'Adventure, Comedy, Family': 40, 'Adventure, Comedy, Fantasy': 41, 'Adventure, Comedy, Film-Noir': 42, 'Adventure, Comedy, Sci-Fi': 43, 'Adventure, Comedy, War': 44, 'Adventure, Drama': 45, 'Adventure, Drama, Fantasy': 46, 'Adventure, Drama, History': 47, 'Adventure, Drama, Musical': 48, 'Adventure, Drama, Romance': 49, 'Adventure, Drama, Sci-Fi': 50, 'Adventure, Drama, Thriller': 51, 'Adventure, Drama, War': 52, 'Adventure, Drama, Western': 53, 'Adventure, Family, Fantasy': 54, 'Adventure, History, War': 55, 'Adventure, Horror, Sci-Fi': 56, 'Adventure, Mystery, Thriller': 57, 'Adventure, Sci-Fi': 58, 'Animation, Action, Adventure': 59, 'Animation, Action, Crime': 60, 'Animation, Action, Drama': 61, 'Animation, Action, Fantasy': 62, 'Animation, Action, Sci-Fi': 63, 'Animation, Adventure, Comedy': 64, 'Animation, Adventure, Drama': 65, 'Animation, Adventure, Family': 66, 'Animation, Adventure, Fantasy': 67, 'Animation, Biography, Crime': 68, 'Animation, Biography, Drama': 69, 'Animation, Comedy, Drama': 70, 'Animation, Comedy, Fantasy': 71, 'Animation, Crime, Mystery': 72, 'Animation, Drama, Family': 73, 'Animation, Drama, Fantasy': 74, 'Animation, Drama, Romance': 75, 'Animation, Drama, War': 76, 'Animation, Family, Fantasy': 77, 'Animation, Sci-Fi': 78, 'Biography, Comedy, Drama': 79, 'Biography, Crime, Drama': 80, 'Biography, Drama': 81, 'Biography, Drama, Family': 82, 'Biography, Drama, History': 83, 'Biography, Drama, Music': 84, 'Biography, Drama, Romance': 85, 'Biography, Drama, Sport': 86, 'Biography, Drama, Thriller': 87, 'Biography, Drama, War': 88, 'Comedy': 89, 'Comedy, Crime': 90, 'Comedy, Crime, Drama': 91, 'Comedy, Crime, Mystery': 92, 'Comedy, Crime, Romance': 93, 'Comedy, Crime, Sport': 94, 'Comedy, Crime, Thriller': 95, 'Comedy, Drama': 96, 'Comedy, Drama, Family': 97, 'Comedy, Drama, Fantasy': 98, 'Comedy, Drama, Music': 99, 'Comedy, Drama, Musical': 100, 'Comedy, Drama, Romance': 101, 'Comedy, Drama, Thriller': 102, 'Comedy, Drama, War': 103, 'Comedy, Family': 104, 'Comedy, Family, Fantasy': 105, 'Comedy, Family, Romance': 106, 'Comedy, Fantasy, Romance': 107, 'Comedy, Horror': 108, 'Comedy, Music': 109, 'Comedy, Music, Musical': 110, 'Comedy, Music, Romance': 111, 'Comedy, Musical, Romance': 112, 'Comedy, Musical, War': 113, 'Comedy, Mystery, Romance': 114, 'Comedy, Romance': 115, 'Comedy, War':

116, 'Comedy, Western': 117, 'Crime, Drama': 118, 'Crime, Drama, Fantasy': 119, 'Crime, Drama, Film-Noir': 120, 'Crime, Drama, History': 121, 'Crime, Drama, Horror': 122, 'Crime, Drama, Music': 123, 'Crime, Drama, Musical': 124, 'Crime, Drama, Mystery': 125, 'Crime, Drama, Romance': 126, 'Crime, Drama, Sci-Fi': 127, 'Crime, Drama, Thriller': 128, 'Crime, Film-Noir, Mystery': 129, 'Crime, Film-Noir, Thriller': 130, 'Crime, Mystery, Thriller': 131, 'Crime, Thriller': 132, 'Drama': 133, 'Drama, Family': 134, 'Drama, Family, Fantasy': 135, 'Drama, Family, Musical': 136, 'Drama, Family, Sport': 137, 'Drama, Fantasy': 138, 'Drama, Fantasy, History': 139, 'Drama, Fantasy, Horror': 140, 'Drama, Fantasy, Music': 141, 'Drama, Fantasy, Mystery': 142, 'Drama, Fantasy, Romance': 143, 'Drama, Fantasy, War': 144, 'Drama, Film-Noir': 145, 'Drama, Film-Noir, Mystery': 146, 'Drama, Film-Noir, Romance': 147, 'Drama, History': 148, 'Drama, History, Music': 149, 'Drama, History, Mystery': 150, 'Drama, History, Romance': 151, 'Drama, History, Thriller': 152, 'Drama, History, War': 153, 'Drama, Horror': 154, 'Drama, Horror, Mystery': 155, 'Drama, Horror, Sci-Fi': 156, 'Drama, Horror, Thriller': 157, 'Drama, Music': 158, 'Drama, Music, Musical': 159, 'Drama, Music, Mystery': 160, 'Drama, Music, Romance': 161, 'Drama, Musical': 162, 'Drama, Mystery': 163, 'Drama, Mystery, Romance': 164, 'Drama, Mystery, Sci-Fi': 165, 'Drama, Mystery, Thriller': 166, 'Drama, Mystery, War': 167, 'Drama, Romance': 168, 'Drama, Romance, Sci-Fi': 169, 'Drama, Romance, Thriller': 170, 'Drama, Romance, War': 171, 'Drama, Sci-Fi': 172, 'Drama, Sci-Fi, Thriller': 173, 'Drama, Sport': 174, 'Drama, Thriller': 175, 'Drama, Thriller, War': 176, 'Drama, Thriller, Western': 177, 'Drama, War': 178, 'Drama, War, Western': 179, 'Drama, Western': 180, 'Family, Fantasy, Musical': 181, 'Fantasy, Horror': 182, 'Fantasy, Horror, Mystery': 183, 'Film-Noir, Mystery': 184, 'Film-Noir, Mystery, Thriller': 185, 'Film-Noir, Thriller': 186, 'Horror': 187, 'Horror, Mystery, Sci-Fi': 188, 'Horror, Mystery, Thriller': 189, 'Horror, Sci-Fi': 190, 'Horror, Thriller': 191, 'Mystery, Romance, Thriller': 192, 'Mystery, Sci-Fi, Thriller': 193, 'Mystery, Thriller': 194, 'Thriller': 195, 'Western': 196}

Mapeamento para 'Director':

{'Aamir Khan': 1, 'Aaron Sorkin': 2, 'Abdellatif Kechiche': 3, 'Abhishek Chaubey': 4, 'Abhishek Kapoor': 5, 'Adam Elliot': 6, 'Adam McKay': 7, 'Aditya Chopra': 8, 'Aditya Dhar': 9, 'Akira Kurosawa': 10, 'Alain Resnais': 11, 'Alan J. Pakula': 12, 'Alan Parker': 13, 'Alejandro Amenábar': 14, 'Alejandro G. Iñárritu': 15, 'Alejandro Jodorowsky': 16, 'Alex Garland': 17, 'Alex Proyas': 18, 'Alexander Mackendrick': 19, 'Alexander Payne': 20, 'Alfonso Cuarón': 21, 'Alfonso Gomez-Rejon': 22, 'Alfred Hitchcock': 23, 'Amit Ravindernath Sharma': 24, 'Anders Thomas Jensen': 25, 'Andrei Tarkovsky': 26, 'Andrew Lau': 27, 'Andrew Niccol': 28, 'Andrey Zvyagintsev': 29, 'Aneesh Chaganty': 30, 'Ang Lee': 31, 'Aniruddha Roy Chowdhury': 32, 'Anthony Harvey': 33, 'Antoine Fuqua': 34, 'Anton Corbijn': 35, 'Anurag Basu': 36, 'Anurag Kashyap': 37, 'Arthur Penn': 38, 'Asghar Farhadi': 39, 'Ashutosh Gowariker': 40, 'Barry Levinson': 41, 'Ben Affleck': 42, 'Bennett Miller': 43, 'Bernardo Bertolucci': 44, 'Billy Bob Thornton': 45, 'Billy Wilder': 46, 'Blake Edwards': 47, 'Boaz Yakin': 48, 'Bob Clark': 49, 'Bob Fosse': 50, 'Bob Gale': 51, 'Bong Joon Ho': 52, 'Brad Anderson': 53, 'Brad Bird': 54, 'Brian De Palma': 55, 'Brian G. Hutton': 56,

'Brian Henson': 57, 'Bruce Robinson': 58, 'Bryan Singer': 59, 'Buster Keaton': 60, 'Cameron Crowe': 61, 'Can Ulkay': 62, 'Carl Theodor Dreyer': 63, 'Carol Reed': 64, 'Cary Joji Fukunaga': 65, 'Cecil B. DeMille': 66, 'Chan-wook Park': 67, 'Charles Chaplin': 68, 'Charles Laughton': 69, 'Charles Vidor': 70, 'Charlie Kaufman': 71, 'Christian Carion': 72, 'Christophe Barratier': 73, 'Christopher Nolan': 74, 'Clint Eastwood': 75, 'Clyde Bruckman': 76, 'Cristian Mungiu': 77, 'Curtis Hanson': 78, 'Cy Endfield': 79, 'Céline Sciamma': 80, 'Damien Chazelle': 81, 'Damián Szifron': 82, 'Dan Gilroy': 83, 'Daniel Monzón': 84, 'Danis Tanovic': 85, 'Danny Boyle': 86, 'Darius Marder': 87, 'Darren Aronofsky': 88, 'David Ayer': 89, 'David Cronenberg': 90, 'David Fincher': 91, 'David Lean': 92, 'David Lynch': 93, 'David Mackenzie': 94, 'David Mickey Evans': 95, 'David O. Russell': 96, 'David Zucker': 97, 'Denis Villeneuve': 98, 'Deniz Gamze Ergüven': 99, 'Dennis Gansel': 100, 'Destin Daniel Cretton': 101, 'Don Siegel': 102, 'Dorota Kobiela': 103, 'Doug Liman': 104, 'Duncan Jones': 105, 'Edgar Wright': 106, 'Edward Zwick': 107, 'Elem Klimov': 108, 'Elia Kazan': 109, 'Emir Kusturica': 110, 'Eric Bress': 111, 'Ericson Core': 112, 'Ernst Lubitsch': 113, 'Ethan Coen': 114, 'F. Gary Gray': 115, 'F.W. Murnau': 116, 'Fabián Bielinsky': 117, 'Farhan Akhtar': 118, 'Fatih Akin': 119, 'Federico Fellini': 120, 'Felix van Groeningen': 121, 'Fernando Meirelles': 122, 'Florian Henckel von Donnersmarck': 123, 'Francis Ford Coppola': 124, 'Francis Lee': 125, 'Francis Veber': 126, 'Frank Capra': 127, 'Frank Darabont': 128, 'Frank Miller': 129, 'Franklin J. Schaffner': 130, 'François Truffaut': 131, 'Fred Zinnemann': 132, 'Fritz Lang': 133, 'Gabriele Muccino': 134, 'Gareth Evans': 135, 'Garth Davis': 136, 'Gauri Shinde': 137, "Gavin O'Connor": 138, 'Gayatri': 139, 'Gene Saks': 140, 'George A. Romero': 141, 'George Cukor': 142, 'George Miller': 143, 'George P. Cosmatos': 144, 'George Roy Hill': 145, 'George Seaton': 146, 'George Sluizer': 147, 'George Stevens': 148, 'Georges Franju': 149, 'Gillo Pontecorvo': 150, 'Giuseppe Tornatore': 151, 'Gregg Araki': 152, 'Gregory Hoblit': 153, 'Greta Gerwig': 154, 'Guillermo del Toro': 155, 'Gus Van Sant': 156, 'Guy Hamilton': 157, 'Guy Ritchie': 158, 'Hal Ashby': 159, 'Hannes Holm': 160, 'Harold Ramis': 161, 'Hayao Miyazaki': 162, 'Henri-Georges Clouzot': 163, 'Henry Koster': 164, 'Henry Selick': 165, 'Hideaki Anno': 166, 'Hirokazu Koreeda': 167, 'Hiromasa Yonebayashi': 168, 'Hong-jin Na': 169, 'Howard Hawks': 170, 'Hrishikesh Mukherjee': 171, 'Imtiaz Ali': 172, 'Ingmar Bergman': 173, 'Isao Takahata': 174, 'J. Lee Thompson': 175, 'Jack Clayton': 176, 'Jaco Van Dormael': 177, 'Jacques Audiard': 178, 'Jacques Tourneur': 179, 'Jae-young Kwak': 180, 'James Algar': 181, 'James Cameron': 182, 'James Foley': 183, 'James Frawley': 184, 'James Ivory': 185, 'James L. Brooks': 186, 'James Mangold': 187, 'James Marsh': 188, 'James McTeigue': 189, 'James Simone': 190, 'James Wan': 191, 'James Whale': 192, 'Je-kyu Kang': 193, 'Jean Renoir': 194, 'Jean-Jacques Annaud': 195, 'Jean-Luc Godard': 196, 'Jean-Marc Vallée': 197, 'Jean-Pierre Jeunet': 198, 'Jean-Pierre Melville': 199, 'Jee-woon Kim': 200, 'Jeethu Joseph': 201, 'Jemaine Clement': 202, 'Jeong-beom Lee': 203, 'Jessie Nelson': 204, 'Jim Abrahams': 205, 'Jim Jarmusch': 206, 'Jim Sheridan': 207, 'Joe Johnston': 208, 'Joe Wright': 209, 'Joel Coen': 210, 'Joel Schumacher': 211, 'John Boorman': 212, 'John Cameron Mitchell': 213, 'John Carney': 214, 'John Carpenter': 215, 'John Ford': 216, 'John Frankenheimer': 217, 'John G. Avildsen': 218, 'John Hughes': 219, 'John Huston': 220, 'John Landis': 221, 'John McTiernan': 222,

'John Schlesinger': 223, 'John Singleton': 224, 'John Sturges': 225, 'John Woo': 226, 'Jon Avnet': 227, 'Jonathan Dayton': 228, 'Jonathan Demme': 229, 'Jonathan Levine': 230, 'Jonathan Lynn': 231, 'Joseph Kosinski': 232, 'Joseph L. Mankiewicz': 233, 'Joseph Sargent': 234, 'Josh Boone': 235, 'Joss Whedon': 236, 'José Padilha': 237, 'Juan José Campanella': 238, 'Jules Dassin': 239, 'Julian Schnabel': 240, 'Kabir Khan': 241, 'Kaige Chen': 242, 'Kar-Wai Wong': 243, 'Karan Johar': 244, 'Katsuhiro Ôtomo': 245, 'Ken Annakin': 246, 'Ken Loach': 247, 'Kenneth Branagh': 248, 'Kenneth Lonergan': 249, 'Kevin Altieri': 250, 'Kevin Macdonald': 251, 'Kevin Reynolds': 252, 'Kevin Smith': 253, 'Ki-duk Kim': 254, 'Kinji Fukasaku': 255, 'Krzysztof Kieslowski': 256, 'Lars von Trier': 257, 'Lasse Hallström': 258, 'Lee Tamahori': 259, 'Lenny Abrahamson': 260, 'Leo McCarey': 261, 'Levent Semerci': 262, 'Lewis Milestone': 263, 'Louis Malle': 264, 'Luc Besson': 265, 'Luca Guadagnino': 266, 'Luis Buñuel': 267, 'Lukas Moodysson': 268, 'Majid Majidi': 269, 'Makoto Shinkai': 270, 'Mamoru Hosoda': 271, 'Mamoru Oshii': 272, 'Marc Caro': 273, 'Marc Forster': 274, 'Marc Webb': 275, 'Mark Herman': 276, 'Mark Osborne': 277, 'Mark Rydell': 278, 'Martin Brest': 279, 'Martin McDonagh': 280, 'Martin Rosen': 281, 'Martin Scorsese': 282, 'Martin Zandvliet': 283, 'Mary Harron': 284, 'Masaki Kobayashi': 285, 'Mathieu Kassovitz': 286, 'Matt Ross': 287, 'Matthew Vaughn': 288, 'Matthew Warchus': 289, 'Meghna Gulzar': 290, 'Mehmet Ada Öztekin': 291, 'Mel Brooks': 292, 'Mel Gibson': 293, 'Mel Stuart': 294, 'Merian C. Cooper': 295, 'Michael Cimino': 296, 'Michael Curtiz': 297, 'Michael Haneke': 298, 'Michael Mann': 299, 'Michael Powell': 300, 'Michael Radford': 301, 'Michel Gondry': 302, 'Michel Hazanavicius': 303, 'Michelangelo Antonioni': 304, 'Mikael Håfström': 305, 'Mike Judge': 306, 'Mike Leigh': 307, 'Mike Newell': 308, 'Mike Nichols': 309, 'Milos Forman': 310, 'Morten Tyldum': 311, 'Moustapha Akkad': 312, 'Mukesh Chhabra': 313, 'Nadine Labaki': 314, 'Naoko Yamada': 315, 'Neeraj Pandey': 316, 'Neil Burger': 317, 'Neill Blomkamp': 318, 'Nicholas Meyer': 319, 'Nicholas Ray': 320, 'Nick Cassavetes': 321, 'Nicolas Winding Refn': 322, 'Niels Arden Oplev': 323, 'Nikkhil Advani': 324, 'Nishikant Kamat': 325, 'Nitesh Tiwari': 326, 'Noah Baumbach': 327, 'Norman Jewison': 328, 'Nuri Bilge Ceylan': 329, 'Oliver Hirschbiegel': 330, 'Oliver Stone': 331, 'Olivier Dahan': 332, 'Olivier Nakache': 333, 'Oriol Paulo': 334, 'Orson Welles': 335, 'Otto Preminger': 336, 'Paolo Genovese': 337, 'Paolo Sorrentino': 338, 'Paul Greengrass': 339, 'Paul Haggis': 340, 'Paul King': 341, 'Paul McGuigan': 342, 'Paul Thomas Anderson': 343, 'Paul Verhoeven': 344, 'Pedro Almodóvar': 345, 'Penny Marshall': 346, 'Pete Docter': 347, 'Peter Bogdanovich': 348, 'Peter Farrelly': 349, 'Peter Mullan': 350, 'Peter Weir': 351, 'Philip Kaufman': 352, 'Pierre Morel': 353, 'Prashanth Neel': 354, 'Priyadarshan': 355, 'Quentin Tarantino': 356, 'Rahi Anil Barve': 357, 'Raja Menon': 358, 'Rajkumar Hirani': 359, 'Rajkumar Santoshi': 360, 'Rakeysh Omprakash Mehra': 361, 'Ramesh Sippy': 362, 'Raoul Walsh': 363, 'René Laloux': 364, 'Richard Attenborough': 365, 'Richard Brooks': 366, 'Richard Curtis': 367, 'Richard Donner': 368, 'Richard Kelly': 369, 'Richard Lester': 370, 'Richard Linklater': 371, 'Richard Schenkman': 372, 'Ridley Scott': 373, 'Ritesh Batra': 374, 'Rob Reiner': 375, 'Robert Aldrich': 376, 'Robert Altman': 377, 'Robert Benton': 378, 'Robert Clouse': 379, 'Robert De Niro': 380, 'Robert Hamer': 381, 'Robert Mulligan': 382, 'Robert Redford': 383, 'Robert Rossen': 384, 'Robert Stevenson': 385, 'Robert Wiene': 386, 'Robert Wise': 387, 'Robert

```
Zemeckis': 388, 'Roberto Benigni': 389, 'Roger Donaldson': 390, 'Roland Joffé':
391, 'Roman Polanski': 392, 'Ron Clements': 393, 'Ron Howard': 394, 'Ronny Yu':
395, 'Ruben Fleischer': 396, 'Ryan Coogler': 397, 'S.S. Rajamouli': 398, 'Sam
Mendes': 399, 'Sam Peckinpah': 400, 'Sam Raimi': 401, 'Sam Wood': 402, 'Sanjay
Leela Bhansali': 403, 'Satoshi Kon': 404, 'Scott Hicks': 405, 'Sean Baker': 406,
'Sean Penn': 407, 'Sebastian Schipper': 408, 'Sergei M. Eisenstein': 409,
'Sergio Leone': 410, 'Sergio Pablos': 411, 'Shane Meadows': 412, 'Shimit Amin':
413, "Shin'ichirô Watanabe": 414, 'Shoojit Sircar': 415, 'Shûsuke Kaneko': 416,
'Sidney Lumet': 417, 'Sofia Coppola': 418, 'Spike Jonze': 419, 'Spike Lee': 420,
'Sriram Raghavan': 421, 'Stanley Donen': 422, 'Stanley Kramer': 423, 'Stanley
Kubrick': 424, 'Stephen Chbosky': 425, 'Stephen Chow': 426, 'Stephen Daldry':
427, 'Stephen Frears': 428, 'Steve McQueen': 429, 'Steven Spielberg': 430,
'Stuart Rosenberg': 431, 'Sudha Kongara': 432, 'Sujoy Ghosh': 433, 'Susanne
Bier': 434, 'Sylvain Chomet': 435, 'Taika Waititi': 436, 'Takeshi Kitano': 437,
'Tarsem Singh': 438, 'Taylor Hackford': 439, 'Taylor Sheridan': 440, 'Ted
Demme': 441, 'Ted Kotcheff': 442, 'Terence Young': 443, 'Terrence Malick': 444,
'Terry George': 445, 'Terry Gilliam': 446, 'Terry Jones': 447, 'Tetsuya
Nakashima': 448, 'Thomas Jahn': 449, 'Thomas Kail': 450, 'Thomas Vinterberg':
451, 'Tigmanshu Dhulia': 452, 'Tim Burton': 453, 'Tod Browning': 454, 'Todd
Haynes': 455, 'Todd Solondz': 456, 'Tom Hooper': 457, 'Tom McCarthy': 458, 'Tom
Tykwer': 459, 'Tomas Alfredson': 460, 'Tomm Moore': 461, 'Tony Bancroft': 462,
'Tony Kaye': 463, 'Tony Scott': 464, 'Travis Knight': 465, 'Trey Parker': 466,
'Troy Duffy': 467, 'Tyler Nilson': 468, 'Umesh Shukla': 469, 'Victor Fleming':
470, 'Vikas Bahl': 471, 'Vikramaditya Motwane': 472, 'Vincent Paronnaud': 473,
'Vishal Bhardwaj': 474, 'Vittorio De Sica': 475, 'W.S. Van Dyke': 476, 'Walter
Hill': 477, 'Walter Salles': 478, 'Werner Herzog': 479, 'Wes Anderson': 480,
'William Friedkin': 481, 'William Wyler': 482, 'Wilson Yip': 483, 'Wim Wenders':
484, 'Wolfgang Becker': 485, 'Wolfgang Petersen': 486, 'Wolfgang Reitherman':
487, 'Woody Allen': 488, 'Xavier Dolan': 489, 'Yann Samuell': 490, 'Yash
Chopra': 491, 'Yasujirô Ozu': 492, 'Yavuz Turgul': 493, 'Yilmaz Erdogan': 494,
'Yimou Zhang': 495, 'Yoshiaki Kawajiri': 496, 'Yoshifumi Kondô': 497, 'Yôjirô
Takita': 498, 'Zack Snyder': 499, 'Zaza Urushadze': 500, 'Zoya Akhtar': 501,
'Çagan Irmak': 502, 'Ömer Faruk Sorak': 503}
```

[ ]:
```
#Checando nossas colunas
dfr.columns
```

[ ]:
```
Index(['Certificate', 'Genre', 'IMDB_Rating', 'Meta_score', 'Director',
       'No_of_Votes', 'Gross'],
      dtype='object')
```

[ ]:
```
dfr.dtypes
```

[ ]:
```
Certificate      int64
Genre            int64
IMDB_Rating    float64
Meta_score     float64
```

```
Director         int64
No_of_Votes      int64
Gross          float64
dtype: object
```

[ ]:
```
#Selecionando a nossa variável alvo -  que vamos prever.
X = dfr.drop('IMDB_Rating', axis=1)
# variável dependente
y = dfr['IMDB_Rating']
```

[ ]:
```
# Definindo tamanhos para os conjuntos de teste que podem ser utilizados␣
 ↪posteriormente ao dividir os dados em conjuntos de treinamento e teste.
test_sizes = [0.2, 0.3]
```

[ ]:
```
#Dividir um conjunto de dados em conjuntos de treinamento e teste, treinar um␣
 ↪modelo de regressão linear com os dados de treinamento
def train_test_regression(X, y, test_size=0.2, random_state=7):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, random_state=random_state)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    return mse, r2, mape, rmse
```

[ ]:
```
#Conhecendo os valores
print('Regressão Linear')
print()
for test_size in test_sizes:
    mse, r2, mape, rmse = train_test_regression(X, y, test_size=test_size)
    print(f'Test Size: {test_size}')
    print('MAPE (Mean Absolute Percentage Error):', mape)
    print('MSE (Mean Squared Error):', mse)
    print('R2:', r2)
    print('RMSE:', rmse)
    print('-----------------------------------------------------------------')
```

```
Regressão Linear

Test Size: 0.2
MAPE (Mean Absolute Percentage Error): 2.2870287639534412
MSE (Mean Squared Error): 0.04673564743482177
R2: 0.21880343465995955
RMSE: 0.21618429044410642
```

```
------------------------------------------------------------
Test Size: 0.3
MAPE (Mean Absolute Percentage Error): 2.2753319663185905
MSE (Mean Squared Error): 0.047190430525477695
R2: 0.19414542969412296
RMSE: 0.21723358516923136
------------------------------------------------------------
```

[ ]:
```
#Treinando modelo de regressão usando o algoritmo de Random Forest
def train_test_random_forest(X, y, test_size=0.3, random_state=7,␣
 ↪n_estimators=100):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, random_state=random_state)
    model = RandomForestRegressor(n_estimators=n_estimators)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    rmse = np.sqrt(mse)
    return mse, r2, mape, rmse
```

[ ]:
```
#Conhecendo os valores
print('Random Forest')
print()
for test_size in test_sizes:
    mse, r2, mape, rmse = train_test_random_forest(X, y, test_size=test_size)
    print(f'Test Size: {test_size}')
    print('MAPE (Mean Absolute Percentage Error):', mape)
    print('MSE (Mean Squared Error):', mse)
    print('R2:', r2)
    print('RMSE:', np.sqrt(mse))
    print('------------------------------------------------------------')
```

```
Random Forest

Test Size: 0.2
MAPE (Mean Absolute Percentage Error): 2.104200750150485
MSE (Mean Squared Error): 0.04024870621468929
R2: 0.32723407548498595
RMSE: 0.20062080204876384
------------------------------------------------------------
Test Size: 0.3
MAPE (Mean Absolute Percentage Error): 2.025545576259571
MSE (Mean Squared Error): 0.03908587924528313
R2: 0.3325440333240912
RMSE: 0.19770149024547876
------------------------------------------------------------
```

[ ]:
```
#Selecionando a nossa variável alvo -  que vamos prever.
X = dfr.drop('IMDB_Rating', axis=1)
y = dfr['IMDB_Rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=7)
```

[ ]:
```
# Treinando a nossa rede neural
def train_test_neural_network_1_layer(X, y, hidden_layers=1, neurons=10,␣
 ↪activation='relu', test_size=0.2, epochs=20, batch_size=32):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, random_state=7)
    model = Sequential()
    model.add(Dense(neurons, input_dim=X_train.shape[1], activation=activation))
    for _ in range(hidden_layers - 1):
        model.add(Dense(neurons, activation=activation))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,␣
 ↪validation_data=(X_test, y_test), verbose=0)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    return mse, rmse
```

[ ]:
```
# Configurações para testar
configurations = [
    (1, 10, 'relu'),  # 1 camada escondida com 10 neurônios e função de␣
 ↪ativação ReLU
    (1, 20, 'relu'),  # 1 camada escondida com 20 neurônios e função de␣
 ↪ativação ReLU
    (1, 30, 'relu'),  # 1 camada escondida com 30 neurônios e função de␣
 ↪ativação ReLU
    (1, 40, 'relu'),  # 1 camada escondida com 40 neurônios e função de␣
 ↪ativação ReLU
    (1, 50, 'relu'),  # 1 camada escondida com 50 neurônios e função de␣
 ↪ativação ReLU
]

# Parâmetros de teste
test_size = 0.2
epochs = 20
batch_size = 32

print('Rede neural (configuração: nº camadas, nº neurônios, função de␣
 ↪ativação)')
for config in configurations:
```

```
    hidden_layers, neurons, activation = config
    mse, rmse = train_test_neural_network_1_layer(X, y,␣
↪hidden_layers=hidden_layers, neurons=neurons, activation=activation,␣
↪test_size=test_size, epochs=epochs, batch_size=batch_size)
    print(f'Configuração: {config}')
    print('MSE:', mse)
    print('RMSE:', rmse)
  ␣
↪print('----------------------------------------------------------------------------------------
```

```
Rede neural (configuração: nº camadas, nº neurônios, função de ativação)
6/6 [==============================] - 0s 4ms/step
Configuração: (1, 10, 'relu')
MSE: 368171267.37541884
RMSE: 19187.789538542966
--------------------------------------------------------------------------------
----------
6/6 [==============================] - 0s 6ms/step
Configuração: (1, 20, 'relu')
MSE: 7685330969.666211
RMSE: 87666.01946972504
--------------------------------------------------------------------------------
----------
6/6 [==============================] - 0s 3ms/step
Configuração: (1, 30, 'relu')
MSE: 1596886526.7180402
RMSE: 39961.06263249315
--------------------------------------------------------------------------------
----------
6/6 [==============================] - 0s 4ms/step
Configuração: (1, 40, 'relu')
MSE: 1945305.3499314315
RMSE: 1394.7420370561115
--------------------------------------------------------------------------------
----------
6/6 [==============================] - 0s 8ms/step
Configuração: (1, 50, 'relu')
MSE: 13528992.693400275
RMSE: 3678.1779039900007
--------------------------------------------------------------------------------
----------
```

0.7 5. Avaliação (Evaluation)

A avaliação dos modelos desenvolvidos é crucial para medir sua qualidade e desempenho. Nesta fase, utilizamos métodos como validação cruzada e métricas de desempenho para avaliar o quão bem os modelos se saem em dados não vistos. Com base nessa avaliação, podemos ajustar e aprimorar os modelos, se necessário.

[ ]:
```
# Definindo o modelo
model = RandomForestRegressor(random_state=7)

# Definindo os hiperparâmetros a serem testados
param_grid = {
    'n_estimators': [50, 100, 200],  # Número de árvores na floresta
    'max_depth': [None, 10, 20, 30],  # Profundidade máxima da árvore
    'min_samples_split': [2, 5, 10],  # Número mínimo de amostras necessárias␣
 ↪para dividir um nó
    'min_samples_leaf': [1, 2, 4]  # Número mínimo de amostras necessárias em␣
 ↪um nó folha
}

# Configurando o Grid Search com validação cruzada
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           scoring='neg_mean_squared_error',
                           cv=5,  # Número de folds para validação cruzada
                           verbose=2,
                           n_jobs=-1)  # Usar todos os núcleos disponíveis

# Executando o Grid Search
grid_search.fit(X_train, y_train)

# Resultados do Grid Search
print("Melhores hiperparâmetros encontrados:")
print(grid_search.best_params_)

# Previsões com o melhor modelo encontrado
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)

# Avaliação do modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MSE: {mse:.4f}")
print(f"R²: {r2:.4f}")
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Melhores hiperparâmetros encontrados:
{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators':
200}
MSE: 0.0395
R²: 0.3401
```

Produto D: Teste do modelo

Amostra * {‘Series_Title’: ‘The Shawshank Redemption’, * ‘Released_Year’: ‘1994’, * ‘Certificate’:

‘A’, * ‘Runtime’: ‘142 min’, * ‘Genre’: ‘Drama’, * ‘Overview’: ‘Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.’, * ‘Meta_score’: 80.0, * ‘Director’: ‘Frank Darabont’, * ‘Star1’: ‘Tim Robbins’, * ‘Star2’: ‘Morgan Freeman’, * ‘Star3’: ‘Bob Gunton’, * ‘Star4’: ‘William Sadler’, * ‘No_of_Votes’: 2343110, * ‘Gross’: ‘28,341,469’} »>Qual seria a nota do IMDB? 8.626

[ ]:
```
#Convertendo para poder mapear a amostra com o nosso dataset
print("Número associado a 'Certificate':", certificate_mapping['A'])
```

```
Número associado a 'Certificate': 2
```

[ ]:
```
print("Número associado a 'Director':", director_mapping['Frank Darabont'])
```

```
Número associado a 'Director': 128
```

[ ]:
```
amostra = {
 'Released_Year':1994,
 'Certificate': 2,
 'Runtime': 142,
 'Genre': 7,
 'Meta_score': 80.0,
 'Director': 128,
 'No_of_Votes': 2343110,
 'Gross': 28341469
  }
```

[ ]:
```
#Aplicando o random forest melhor resultado na amostra sugerida
def train_test_random_forest(X, y, test_size=0.2, random_state=7,␣
 ↪n_estimators=100):
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_size, random_state=random_state)
    modelRF = RandomForestRegressor(n_estimators=n_estimators)
    modelRF.fit(X_train, y_train)
    joblib.dump(modelRF, 'finalcompleto_model.pkl')
    return modelRF
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=7)
model = train_test_random_forest(X_train, y_train)
```

[ ]:
```
new_df = pd.DataFrame(amostra, columns=X.columns, index=[0])
predicted_rating = model.predict(new_df)
print("IMDB_Rating:", predicted_rating)
```

```
IMDB_Rating: [8.59]
```

0.8 6. Implantação (Deployment)

A fase final da metodologia CRISP-DM é a implantação do modelo em um ambiente de produção. Integramos o modelo aos sistemas existentes, monitoramos seu desempenho contínuo e garantimos a adoção pela equipe de negócios.

Produto E: Arquivo salvo.pkl

[ ]:
```
#Salvando o nosso modelo
with open('LH_CD_MiriamAguiarSobral_MODELOFINALCOMPLETO.pkl', 'wb') as arquivo:
    pickle.dump(df, arquivo)
```

Produto F: GitHUB https://github.com/eumoas/LH_CD_MiriamAguiarSobral_IMDb/tree/main

Qual tipo de filme deve ser o próximo a ser desenvolvido?

Com base nas informações que você forneceu sobre os fatores que influenciam a expectativa de faturamento de um filme e as palavras-chave relacionadas a sentimentos positivos e negativos, aqui estão algumas sugestões para um roteiro que pode ter um bom desempenho nas bilheteiras e tirar a indústria cinematográfica da crise que se encontra.

> Drama de Crescimento Pessoal: Tema: Um jovem (ou um grupo de jovens) enfrenta desafios em sua jornada de autodescoberta. Elementos: Inclua temas de amizade, amor e superação, com personagens bem desenvolvidos. A história pode explorar a transição da juventude para a vida adulta, lidando com expectativas sociais e pessoais. Palavras-chave: “young”, “life”, “story”.
>
> Romance Épico com Conflito: Tema: Um romance entre um homem e uma mulher de mundos diferentes, enfrentando barreiras sociais ou culturais. Elementos: O pano de fundo pode incluir um conflito maior, como uma guerra ou uma crise social, que desafia o relacionamento. O enredo deve incluir momentos de tensão e alívio cômico. Palavras-chave: “love”, “war”, “two”.
>
> Aventura Fantástica: Tema: Dois protagonistas (um homem e uma mulher) embarcam em uma jornada em um mundo mágico ou futurista. Elementos: A história pode incluir elementos de ação e humor, com desafios que testam a amizade e o amor entre os personagens. A narrativa pode ter reviravoltas inesperadas e um clímax emocionante. Palavras-chave: “world”, “new”, “adventure”.
>
> História de Superação em Tempos Difíceis: Tema: Um personagem principal luta contra adversidades, como a perda de um ente querido ou uma crise pessoal, enquanto busca um novo propósito na vida. Elementos: O filme pode incluir flashbacks que revelam a vida anterior do personagem e como ele se transforma ao longo da história. A música deve ser usada para intensificar as emoções. Palavras-chave: “life”, “story”, “performance”.
>
> Comédia Dramática: Tema: Uma comédia leve que aborda questões sérias de forma engraçada, como a vida cotidiana de um grupo de

amigos. Elementos: Inclua situações engraçadas que refletem as dificuldades da vida, mas que também oferecem uma mensagem positiva sobre amizade e apoio mútuo. O humor deve ser equilibrado com momentos de reflexão. Palavras-chave: “funny”, “episode”, “performance”.

Considerações Finais:

Os temas de amor, amizade, superação e conflitos sociais são universais e ressoam com um amplo público. Certifique-se de que a narrativa tenha um apelo emocional forte. Desenvolvimento de Personagens: Personagens bem desenvolvidos e relacionáveis são cruciais. O público deve se importar com o que acontece com eles. Uso da Música: A trilha sonora pode ser uma ferramenta poderosa para evocar emoções e realçar momentos-chave da narrativa. Equilíbrio entre Humor e Drama: Um bom equilíbrio entre momentos leves e sérios pode tornar o filme mais acessível e envolvente. Essas sugestões de roteiro, combinadas com os insights sobre palavras-chave e sentimentos, têm o potencial de criar um filme que não só atraia o público, mas que também se destaque nas bilheteiras.

[ ]:
```
!jupyter nbconvert --to pdf /content/KNN.ipynb
```

[ ]:
```

```