```
!pip install ydata-profiling

import kagglehub
import pandas as pd
import numpy as np
from ydata_profiling import ProfileReport
from google.colab import files
import ast
import os
import requests
from bs4 import BeautifulSoup
from tqdm import tqdm
import time
import re
import csv
```

```
Collecting ydata-profiling
  Downloading ydata_profiling-4.16.1-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: scipy<1.16,>=1.4.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: pandas!=1.4.0,<3.0,>1.1 in /usr/local/lib/python3.11/d
Requirement already satisfied: matplotlib<=3.10,>=3.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.11/dist-
Collecting visions<0.8.2,>=0.7.5 (from visions[type_image_path]<0.8.2,>=0.7.5->ydata-
  Downloading visions-0.8.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy<2.2,>=1.16.0 in /usr/local/lib/python3.11/dist-p
Collecting htmlmin==0.1.12 (from ydata-profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling)
  Downloading phik-0.12.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.11/dis
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.11/di
Requirement already satisfied: typeguard<5,>=3 in /usr/local/lib/python3.11/dist-pack
Collecting imagehash==4.3.1 (from ydata-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: wordcloud>=1.9.3 in /usr/local/lib/python3.11/dist-pac
Collecting dacite>=1.8 (from ydata-profiling)
  Downloading dacite-1.9.2-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: numba<=0.61,>=0.56.0 in /usr/local/lib/python3.11/dist
Collecting PyWavelets (from imagehash==4.3.1->ydata-profiling)
  Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-pa
```

```
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: pydantic-core==2.33.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.11/dist-packag
Collecting puremagic (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.
  Downloading puremagic-1.28-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (f
Downloading ydata_profiling-4.16.1-py2.py3-none-any.whl (400 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.1/400.1 kB 7.1 MB/s eta 0:00:00
Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 6.6 MB/s eta 0:00:00
Downloading dacite-1.9.2-py3-none-any.whl (16 kB)
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading phik-0.12.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (6
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 687.8/687.8 kB 21.7 MB/s eta 0:00:00
Downloading visions-0.8.1-py3-none-any.whl (105 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 105.4/105.4 kB 6.9 MB/s eta 0:00:00
Downloading puremagic-1.28-py3-none-any.whl (43 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.2/43.2 kB 2.5 MB/s eta 0:00:00
Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.w
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 15.4 MB/s eta 0:00:00
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27081 sha2
  Stored in directory: /root/.cache/pip/wheels/8d/55/1a/19cd535375ed1ede0c996405ebffe
Successfully built htmlmin
Installing collected packages: puremagic, htmlmin, PyWavelets, multimethod, dacite, i
Successfully installed PyWavelets-1.8.0 dacite-1.9.2 htmlmin-0.1.12 imagehash-4.3.1 m
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection

Download Datasets From Kaggle

```
path = kagglehub.dataset_download("rounakbanik/the-movies-dataset")

path1 = kagglehub.dataset_download("ravineesh/soundtracks-of-top-250-imdb-movies-and-tv-s
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/rounakbanik/the-movi
100%|██████████| 228M/228M [00:01<00:00, 179MB/s]Extracting files...

Downloading from https://www.kaggle.com/api/v1/datasets/download/ravineesh/soundtrack
100%|██████████| 109k/109k [00:00<00:00, 50.4MB/s]Extracting files...
Path to dataset files: /root/.cache/kagglehub/datasets/rounakbanik/the-movies-dataset
Path to dataset files: /root/.cache/kagglehub/datasets/ravineesh/soundtracks-of-top-2
```

Movie Dataset Preprocessing

```
# Read movies_metadata.csv
csv_path = f"{path}/movies_metadata.csv"
df = pd.read_csv(csv_path, low_memory=False)

# Report using ydata-profiling
profile = ProfileReport(df, title="Movies Metadata Profiling Report", explorative=True)
profile.to_notebook_iframe()
```

```
Summarize dataset: 100%                                                    50/50 [00:26<00:00,  3.86it/s, Completed]

  0%|          | 0/24 [00:00<?, ?it/s]
  4%|▌         | 1/24 [00:00<00:08,  2.74it/s]
  8%|▌         | 2/24 [00:00<00:06,  3.32it/s]
 12%|▌         | 3/24 [00:00<00:05,  3.71it/s]
 17%|▌         | 4/24 [00:01<00:11,  1.68it/s]
 21%|▌         | 5/24 [00:02<00:08,  2.35it/s]
 25%|▌         | 6/24 [00:03<00:16,  1.10it/s]
 29%|▌         | 7/24 [00:04<00:13,  1.27it/s]
 33%|▌         | 8/24 [00:04<00:09,  1.63it/s]
 38%|▌         | 9/24 [00:08<00:25,  1.69s/it]
 42%|▌         | 10/24 [00:18<00:59,  4.26s/it]
 54%|▌         | 13/24 [00:19<00:21,  1.94s/it]
 62%|▌         | 15/24 [00:19<00:12,  1.34s/it]
 75%|▌         | 18/24 [00:20<00:05,  1.20it/s]
 83%|▌         | 20/24 [00:20<00:02,  1.49it/s]
100%|▌         | 24/24 [00:21<00:00,  1.12it/s]
Generate report structure: 100%                                            1/1 [00:16<00:00, 16.55s/it]

Render HTML: 100%                                                          1/1 [00:01<00:00,  1.52s/it]
```

Movies Metadata Profiling Report

Overview

Brought to you by YData

Overview | Alerts 14 | Reproduction

Dataset statistics

Number of variables	24
Number of observations	45466
Missing cells	105562
Missing cells (%)	9.7%
Duplicate rows	16
Duplicate rows (%)	< 0.1%
Total size in memory	84.5 MiB
Average record size in memory	1.9 KiB

Variable types

Categorical	2
Text	17
Numeric	4
Boolean	1

```
columns_to_keep = [
    'adult', 'budget', 'genres', 'imdb_id', 'original_language', 'original_title',
    'overview', 'popularity', 'production_companies', 'production_countries',
    'release_date', 'runtime', 'spoken_languages', 'status',
    'title', 'vote_average'
]

df = df[columns_to_keep]


# Clean object-type columns
object_cols = df.select_dtypes(include='object').columns
for col in object_cols:
```

```
    df.loc[:, col] = df[col].replace(['', '[]'], np.nan)
    df.loc[:, col] = df[col].fillna('Unknown')

# Clean numeric-type columns
df['budget'] = pd.to_numeric(df['budget'], errors='coerce')
df['budget'] = df['budget'].apply(lambda x: 'Unknown' if pd.isna(x) or x == 0 else str(in

numeric_cols = ['runtime','vote_average']
for col in numeric_cols:
    df.loc[:, col] = pd.to_numeric(df[col], errors='coerce')
    df.loc[:, col] = df[col].apply(lambda x: 'Unknown' if pd.isna(x) or x == 0 else str(i

# Convert release_date to datetime
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# Create column title_with_year
df['title_with_year'] = df.apply(
    lambda row: f"{row['title']} ({row['release_date'].year})"
    if pd.notnull(row['release_date']) else f"{row['title']} (Unknown)", axis=1
)

# Convert release_date to string for dispaly
df['release_date'] = df['release_date'].dt.strftime('%Y-%m-%d')
df['release_date'] = df['release_date'].fillna('Unknown')

# Remove duplicates
print(" Before removing duplicates:", df.shape[0])
df = df.drop_duplicates(subset='title_with_year', keep='first').reset_index(drop=True)
print(" After duplicates removal:", df.shape[0])
```

Show hidden output

```
lang_map = {
    'en': 'English',
    'fr': 'French',
    'es': 'Spanish',
    'de': 'German',
    'it': 'Italian',
    'ja': 'Japanese',
    'zh': 'Chinese',
    'ko': 'Korean',
    'hi': 'Hindi',
    'ru': 'Russian',
    'pt': 'Portuguese',
    'ar': 'Arabic',
    'tr': 'Turkish',
    'ta': 'Tamil',
    'te': 'Telugu',
    'fa': 'Persian',
```

```
    'pl': 'Polish',
    'nl': 'Dutch',
    'sv': 'Swedish',
    'no': 'Norwegian',
    'cs': 'Czech',
    'ro': 'Romanian',
    'da': 'Danish',
    'he': 'Hebrew',
    'fi': 'Finnish',
    'ur': 'Urdu',
    'xx': 'Unknown'
}

# Replace codes with full names
df['original_language'] = df['original_language'].map(lang_map).fillna('Unknown')



def extract_names_from_column(df, col_name):
    def extract(item):
        try:
            data = ast.literal_eval(item)
            if isinstance(data, list):
                names = [d['name'] for d in data if isinstance(d, dict) and 'name' in d]
                return ','.join(names) if names else 'Unknown'
        except:
            return 'Unknown'
        return 'Unknown'

    df[col_name] = df[col_name].apply(extract)

for col in ['genres', 'production_companies', 'production_countries', 'spoken_languages']
    extract_names_from_column(df, col)

df.loc[:, 'spoken_languages'] = df['spoken_languages'].replace('', np.nan)
df.loc[:, 'spoken_languages'] = df['spoken_languages'].fillna('Unknown')



# Save and download the cleaned dataset
output_file = "cleaned_tmdb_dataset.csv"
df.to_csv(output_file, index=False)
files.download(output_file)

# Final profiling
profile = ProfileReport(df, title="Cleaned TMDB Dataset Profiling", explorative=True)
profile.to_notebook_iframe()
```

Summarize dataset: 100% 26/26 [00:12<00:00, 3.44it/s, Completed]

```
  0%|          | 0/17 [00:00<?, ?it/s]
  6%|▌         | 1/17 [00:00<00:04,  3.63it/s]
 18%|▌▌        | 3/17 [00:00<00:03,  4.63it/s]
 24%|▌▌        | 4/17 [00:01<00:05,  2.39it/s]
 35%|▌▌▌       | 6/17 [00:03<00:07,  1.46it/s]
 41%|▌▌▌       | 7/17 [00:08<00:18,  1.85s/it]
 71%|▌▌▌▌▌     | 12/17 [00:08<00:03,  1.48it/s]
 82%|▌▌▌▌▌▌    | 14/17 [00:08<00:01,  1.90it/s]
 88%|▌▌▌▌▌▌▌   | 15/17 [00:10<00:01,  1.61it/s]
100%|▌▌▌▌▌▌▌▌▌| 17/17 [00:10<00:00,  1.56it/s]
```

Generate report structure: 100% 1/1 [00:11<00:00, 11.73s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.66s/it]

Cleaned TMDB Dataset Profiling

Overview

Brought to you by YData

Overview | Alerts 4 | Reproduction

Dataset statistics

Number of variables	17
Number of observations	45380
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	68.9 MiB
Average record size	1.6 KiB

Variable types

Boolean	1
Text	13
Categorical	3

in memory

Soundtracks Dataset Preprocessing

```
csv_path = os.path.join(path1, 'sound_track_imdb_top_250_movie_tv_series.csv')
df = pd.read_csv(csv_path)

profile = ProfileReport(df, title="Soundtracks Profiling Report", explorative=True)
profile.to_notebook_iframe()

# Copy values from 'written_performed_by' to 'written_by' & 'performed_by' only if it is
if 'written_performed_by' in df.columns:
    mask = df['written_performed_by'].notna() & (df['written_performed_by'] != 'Unknown')
    df.loc[mask, 'written_by'] = df.loc[mask, 'written_performed_by']
    df.loc[mask, 'performed_by'] = df.loc[mask, 'written_performed_by']
    df.drop(columns='written_performed_by', inplace=True)

df.drop(columns='libretto_by',inplace=True)
df.drop(columns='Unnamed: 0',inplace=True)


# Replace NaN with 'Unknown'
df = df.replace({np.nan: 'Unknown'})

# Create title_with_year column
if 'name' in df.columns and 'year' in df.columns:
    df['title_with_year'] = df.apply(
        lambda row: f"{row['name']} ({row['year']})"
        if row['name'] != 'Unknown' and row['year'] != 'Unknown' else 'Unknown',
        axis=1
    )

#  Detect and remove duplicates based on title_with_year and song_name, keeping the most
if 'song_name' in df.columns and 'title_with_year' in df.columns:
    # Count cells that is not 'Unknown'
    df['info_count'] = df.apply(lambda row: sum(row != 'Unknown'), axis=1)

    # Find duplicates
    duplicates = df[df.duplicated(subset=['title_with_year', 'song_name'], keep=False)]
    num_duplicates = duplicates.shape[0]

    if num_duplicates > 0:
        print(f"🔁 Found {num_duplicates} duplicate rows based on 'title_with_year' + 'so
        display(duplicates.sort_values(by=['title_with_year', 'song_name']))

    # In case of a tie in info_count, keep the first occurrence
```

```
    # In case of a tie in info_count, keep the first occurrence
    df['original_index'] = df.index

    df = df.sort_values(by=['title_with_year', 'song_name', 'info_count', 'original_index
                        ascending=[True, True, False, True])

    df = df.drop_duplicates(subset=['title_with_year', 'song_name'], keep='first').reset_

    # Delete rows
    df.drop(columns=['info_count', 'original_index'], inplace=True)

    print("Removed duplicates, keeping the most informative row per group.")
```

```
# Save and download the cleaned dataset
output_file = "cleaned_soundtracks.csv"
df.to_csv(output_file, index=False)
files.download(output_file)

# Final profiling
profile = ProfileReport(df, title="Cleaned Soundtracks Dataset Profiling", explorative=Tr
profile.to_notebook_iframe()
```

Summarize dataset: 100% 28/28 [00:01<00:00, 9.73it/s, Completed]

```
  0%|          | 0/14 [00:00<?, ?it/s]
 29%|▋▋▋       | 4/14 [00:00<00:00, 13.49it/s]
 43%|▋▋▋▋▋     | 6/14 [00:00<00:00, 13.33it/s]
100%|▋▋▋▋▋▋▋▋▋▋| 14/14 [00:00<00:00, 19.79it/s]
```

Generate report structure: 100% 1/1 [00:11<00:00, 11.24s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.29it/s]

Soundtracks Profiling Report

Overview

Brought to you by YData

Overview Alerts 14 Reproduction

Dataset statistics

Variable types

Number of variables	14
Number of observations	3133
Missing cells	25737
Missing cells (%)	58.7%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.8 MiB
Average record size in memory	588.6 B

Numeric	2
Text	11
Categorical	1

Variables

Select Columns

🔁 Found 92 duplicate rows based on 'title_with_year' + 'song_name'

	name	year	song_name	written_by	performed_by	composed_by	lyrics_by
2785	Aspirants	2021	Phod De Ya Chhod De	Rohit Sharma and Deepesh Sumitra Jagdish	Noxious D	Unknown	Unknown
2791	Aspirants	2021	Phod De Ya Chhod De	Unknown	Achint Marwah (Noxious D)	Unknown	Deepesh Sumitra Jagdish
314	Back to the Future	1985	Out The Window	Unknown	Unknown	Unknown	Unknown
319	Back to the Future	1985	Out The Window	Edward Van Halen Played in Marty's walkman	Unknown	Unknown	Unknown
2614	Breaking Bad	2008	Fallacies	Unknown	Twaughthammer	Unknown	Unknown

...	...	...	...	...	...	...	...
747	Toy Story	1995	You've Got a Friend in Me	Unknown	Unknown	Unknown	Unknown
752	Toy Story	1995	You've Got a Friend in Me	Unknown	Randy Newman & Lyle Lovett	Unknown	Unknown
1809	Trainspotting	1996	Temptation	Unknown	Heaven 17	Unknown	Unknown
1811	Trainspotting	1996	Temptation	Stephen Morris / Peter Hook / Bernard Sumner /...	New Order	Unknown	Unknown
1826	Trainspotting	1996	Temptation	Stephen Morris / Peter Hook / Bernard Sumner /...	Kelly Macdonald [When Diane is sitting on Rent...	Unknown	Unknown

92 rows × 13 columns

```
Removed duplicates, keeping the most informative row per group.
```

Summarize dataset: 100% 22/22 [00:01<00:00, 13.79it/s, Completed]

```
  0%|          | 0/12 [00:00<?, ?it/s]
 25%|▌▌▌       | 3/12 [00:00<00:00, 14.23it/s]
 42%|▌▌▌▌▌     | 5/12 [00:00<00:00, 15.55it/s]
 75%|▌▌▌▌▌▌▌   | 9/12 [00:00<00:00, 21.71it/s]
100%|▌▌▌▌▌▌▌▌▌▌| 12/12 [00:00<00:00, 21.09it/s]
```

Generate report structure: 100% 1/1 [00:12<00:00, 12.82s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.09it/s]

Cleaned Soundtracks Dataset Profiling

Overview

Brought to you by YData

Overview Alerts 1 Reproduction

Dataset statistics Variable types

Create csv from the above two

```
df_soundtracks = pd.read_csv("/content/cleaned_soundtracks.csv")
df_tmdb = pd.read_csv("/content/cleaned_tmdb_dataset.csv")

# Find unique title_with_year
titles_soundtracks = set(df_soundtracks['title_with_year'].unique())
titles_tmdb = set(df_tmdb['title_with_year'].unique())

common_titles = titles_soundtracks.intersection(titles_tmdb)

print(f" Common titles found: {len(common_titles)}")
print("Common titles:")
for title in list(sorted(common_titles)):
    print("-", title)

common_songs = df_soundtracks[df_soundtracks['title_with_year'].isin(common_titles)]

# Total songs
print(f"\n Total number of songs from common titles: {len(common_songs)}")


# Filter datasets
df_sound_common = df_soundtracks[df_soundtracks['title_with_year'].isin(common_titles)]
df_tmdb_common = df_tmdb[df_tmdb['title_with_year'].isin(common_titles)]

# Merge based on common title
merged_df = pd.merge(
    df_sound_common,
    df_tmdb_common,
    on='title_with_year',
    suffixes=('_soundtrack', '_tmdb')
)

merged_df.drop(columns='title',inplace=True)
merged_df.drop(columns='name',inplace=True)
merged_df.drop(columns='year',inplace=True)

merged_df.rename(columns={
    'title_with_year': 'movie_title',
    'original_title': 'original_movie_title'
}, inplace=True)


merged_path = "/content/common_titles_merged.csv"
merged_df.to_csv(merged_path, index=False)
```

```
from google.colab import files
files.download(merged_path)

# profiling
profile = ProfileReport(merged_df, title="Merged Dataset Profiling", explorative=True)
profile.to_notebook_iframe()
```

```
 Common titles found: 238
Common titles:
- 12 Angry Men (1957)
- 12 Years a Slave (2013)
- 2001: A Space Odyssey (1968)
- 3 Idiots (2009)
- A Beautiful Mind (2001)
- A Clockwork Orange (1971)
- A Separation (2011)
- Aladdin (1992)
- Alien (1979)
- Aliens (1986)
- All About Eve (1950)
- Amadeus (1984)
- American Beauty (1999)
- American History X (1998)
- Amores perros (2000)
- Amélie (2001)
- Apocalypse Now (1979)
- Back to the Future (1985)
- Band of Brothers (2001)
- Barry Lyndon (1975)
- Batman Begins (2005)
- Beauty and the Beast (1991)
- Before Sunrise (1995)
- Before Sunset (2004)
- Ben-Hur (1959)
- Bicycle Thieves (1948)
- Blade Runner (1982)
- Braveheart (1995)
- Casablanca (1942)
- Casino (1995)
- Catch Me If You Can (2002)
- Children of Heaven (1997)
- Chinatown (1974)
- Cinema Paradiso (1988)
- Citizen Kane (1941)
- City Lights (1931)
- City of God (2002)
- Come and See (1985)
- Cool Hand Luke (1967)
- Dangal (2016)
- Dead Poets Society (1989)
- Death Note (2006)
- Dersu Uzala (1975)
- Dial M for Murder (1954)
- Die Hard (1988)
```

```
- Django Unchained (2012)
- Double Indemnity (1944)
- Downfall (2004)
- Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
- Erased (2016)
- Eternal Sunshine of the Spotless Mind (2004)
- Fargo (1996)
- Fight Club (1999)
- Finding Nemo (2003)
- For a Few Dollars More (1965)
- Forrest Gump (1994)
- From the Earth to the Moon (1998)
- Frozen Planet (2011)
- Full Metal Jacket (1987)
- Gandhi (1982)
- Generation Kill (2008)
- Gladiator (2000)
- Gone Girl (2014)
- Gone with the Wind (1939)
- Good Will Hunting (1997)
- Gran Torino (2008)
- Grave of the Fireflies (1988)
- Groundhog Day (1993)
- Hachi: A Dog's Tale (2009)
- Hacksaw Ridge (2016)
- Happy Valley (2014)
- Harry Potter and the Deathly Hallows: Part 2 (2011)
- Heat (1995)
- High and Low (1963)
- Horace and Pete (2016)
- Hotel Rwanda (2004)
- How to Train Your Dragon (2010)
- Howl's Moving Castle (2004)
- I, Claudius (1976)
- Ikiru (1952)
- In the Name of the Father (1993)
- Incendies (2010)
- Inception (2010)
- Indiana Jones and the Last Crusade (1989)
- Inglourious Basterds (2009)
- Inside Out (2015)
- Interstellar (2014)
- Into the Wild (2007)
- It Happened One Night (1934)
- It's a Wonderful Life (1946)
- Jaws (1975)
- John Adams (2008)
- Judgment at Nuremberg (1961)
- Jurassic Park (1993)
- Kill Bill: Vol. 1 (2003)
- L.A. Confidential (1997)
- La Haine (1995)
- Lawrence of Arabia (1962)
- Life (2009)
```

```
- Life Is Beautiful (1997)
- Life of Brian (1979)
- Lock, Stock and Two Smoking Barrels (1998)
- Logan (2017)
- Long Way Round (2004)
- M (1931)
- Mad Max: Fury Road (2015)
- Mary and Max (2009)
- Memento (2000)
- Memories of Murder (2003)
- Metropolis (1927)
- Million Dollar Baby (2004)
- Modern Times (1936)
- Monsters, Inc. (2001)
- Monty Python and the Holy Grail (1975)
- Mr. Smith Goes to Washington (1939)
- My Father and My Son (2005)
- My Neighbor Totoro (1988)
- Network (1976)
- No Country for Old Men (2007)
- North by Northwest (1959)
- On the Waterfront (1954)
- Once Upon a Time in America (1984)
- Once Upon a Time in the West (1968)
- One Flew Over the Cuckoo's Nest (1975)
- Over the Garden Wall (2014)
- Pan's Labyrinth (2006)
- Pather Panchali (1955)
- Paths of Glory (1957)
- Persona (1966)
- Pirates of the Caribbean: The Curse of the Black Pearl (2003)
- Planet Earth (2006)
- Planet Earth II (2016)
- Platoon (1986)
- Princess Mononoke (1997)
- Prisoners (2013)
- Psycho (1960)
- Pulp Fiction (1994)
- Raging Bull (1980)
- Ran (1985)
- Rashomon (1950)
- Ratatouille (2007)
- Rear Window (1954)
- Rebecca (1940)
- Requiem for a Dream (2000)
- Reservoir Dogs (1992)
- Rocky (1976)
- Room (2015)
- Rush (2013)
- Saving Private Ryan (1998)
- Scarface (1983)
- Schindler's List (1993)
- Se7en (1995)
- Seven Samurai (1954)
- Shutter Island (2010)
```

```
- Shutter Island (2010)
- Singin' in the Rain (1952)
- Snatch (2000)
- Some Like It Hot (1959)
- Spirited Away (2001)
- Spotlight (2015)
- Stand by Me (1986)
- Star Wars (1977)
- Taxi Driver (1976)
- Terminator 2: Judgment Day (1991)
- The 400 Blows (1959)
- The Apartment (1960)
- The Battle of Algiers (1966)
- The Best Years of Our Lives (1946)
- The Big Lebowski (1998)
- The Blue Planet (2001)
- The Bridge on the River Kwai (1957)
- The Civil War (1990)
- The Dark Knight (2008)
- The Dark Knight Rises (2012)
- The Deer Hunter (1978)
- The Departed (2006)
- The Elephant Man (1980)
- The Exorcist (1973)
- The General (1926)
- The Godfather (1972)
- The Godfather: Part II (1974)
- The Gold Rush (1925)
- The Good, the Bad and the Ugly (1966)
- The Grand Budapest Hotel (2014)
- The Grapes of Wrath (1940)
- The Great Dictator (1940)
- The Great Escape (1963)
- The Green Mile (1999)
- The Handmaiden (2016)
- The Help (2011)
- The Hunt (2012)
- The Incredibles (2004)
- The Intouchables (2011)
- The Jinx: The Life and Deaths of Robert Durst (2015)
- The Kid (1921)
- The Lion King (1994)
- The Lives of Others (2006)
- The Lord of the Rings: The Fellowship of the Ring (2001)
- The Lord of the Rings: The Return of the King (2003)
- The Lord of the Rings: The Two Towers (2002)
- The Matrix (1999)
- The Passion of Joan of Arc (1928)
- The Pianist (2002)
- The Prestige (2006)
- The Secret in Their Eyes (2009)
- The Seventh Seal (1957)
- The Shawshank Redemption (1994)
- The Shining (1980)
- The Silence of the Lambs (1991)
```

```
  The Silence of the Lambs (1991)
- The Sixth Sense (1999)
- The Sound of Music (1965)
- The Sting (1973)
- The Terminator (1984)
- The Thing (1982)
- The Third Man (1949)
- The Treasure of the Sierra Madre (1948)
- The Truman Show (1998)
- The Usual Suspects (1995)
- The Wages of Fear (1953)
- The Wizard of Oz (1939)
- The Wolf of Wall Street (2013)
- There Will Be Blood (2007)
- To Be or Not to Be (1942)
- To Kill a Mockingbird (1962)
- Tokyo Story (1953)
- Toy Story (1995)
- Toy Story 3 (2010)
- Trainspotting (1996)
- Unforgiven (1992)
- Up (2009)
- Vertigo (1958)
- WALL·E (2008)
- Warrior (2011)
- Whiplash (2014)
- Wild Strawberries (1957)
- Wild Tales (2014)
- Witness for the Prosecution (1957)
- Yojimbo (1961)
- Your Name. (2016)

 Total number of songs from common titles: 2217
<ipython-input-10-6bcb1fe46d3d>:2: DtypeWarning: Columns (7) have mixed types. Specif
  df_tmdb = pd.read_csv("/content/cleaned_tmdb_dataset.csv")
```

Summarize dataset: 100% 35/35 [00:02<00:00, 8.30it/s, Completed]

```
  0%|          | 0/25 [00:00<?, ?it/s]
  4%|▌         | 1/25 [00:00<00:03,  7.78it/s]
 20%|▉         | 5/25 [00:00<00:01, 19.03it/s]
 28%|▉▌        | 7/25 [00:00<00:00, 19.31it/s]
 52%|▉▉▉▉▌     | 13/25 [00:00<00:00, 29.29it/s]
 68%|▉▉▉▉▉▉▌   | 17/25 [00:00<00:00, 25.25it/s]
100%|▉▉▉▉▉▉▉▉▉▉| 25/25 [00:00<00:00, 27.96it/s]
```

Generate report structure: 100% 1/1 [00:16<00:00, 16.85s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.04it/s]

Merged Dataset Profiling

Overview

Brought to you by YData

Overview | Alerts 9 | Reproduction

Dataset statistics

Number of variables	25
Number of observations	2217
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	4.5 MiB
Average record size in memory	2.1 KiB

Variable types

Text	17
Categorical	4
Boolean	1
Unsupported	1
DateTime	1
Numeric	1

Songs from MusicBrainz API

```
!pip install musicbrainzngs --quiet
import time
import musicbrainzngs
from tqdm.notebook import tqdm
from google.colab import files


# Initiate MusicBrainz API
musicbrainzngs.set_useragent("ColabApp", "1.0", "vaspntl@gmail.com")

df = pd.read_csv("/content/common_titles_merged.csv")

# Create duration_sec column
if 'duration_sec' not in df.columns:
```

```
if 'duration_sec' not in df.columns:
    df['duration_sec'] = 'Unknown'

# Enrich songs for each movie
for i, row in tqdm(df.iterrows(), total=len(df)):
    title = row['song_name']
    artist = row['performed_by'] if pd.notna(row['performed_by']) and row['performed_by']

    try:
        result = musicbrainzngs.search_recordings(recording=title, artist=artist, limit=1
                 else musicbrainzngs.search_recordings(recording=title, limit=1)

        recordings = result.get("recording-list")
        if recordings:
            rec = recordings[0]

            if df.at[i, 'performed_by'] == 'Unknown':
                performer = rec.get('artist-credit', [{}])[0].get('name', '')
                if performer:
                    df.at[i, 'performed_by'] = performer

            if 'length' in rec:
                df.at[i, 'duration_sec'] = round(int(rec['length']) / 1000)

            if 'work-relation-list' in rec:
                work_id = rec['work-relation-list'][0]['work']['id']
                work_data = musicbrainzngs.get_work_by_id(work_id, includes=['artist-rels
                relations = work_data['work'].get('artist-relation-list', [])
                composers = [rel['artist']['name'] for rel in relations if rel['type'] ==
                lyricists = [rel['artist']['name'] for rel in relations if rel['type'] ==

                if composers:
                    if df.at[i, 'composed_by'] == 'Unknown':
                        df.at[i, 'composed_by'] = ', '.join(composers)
                    if df.at[i, 'music_by'] == 'Unknown':
                        df.at[i, 'music_by'] = ', '.join(composers)

                if lyricists:
                    if df.at[i, 'written_by'] == 'Unknown':
                        df.at[i, 'written_by'] = ', '.join(lyricists)
                    if df.at[i, 'lyrics_by'] == 'Unknown':
                        df.at[i, 'lyrics_by'] = ', '.join(lyricists)

    except Exception as e:
        df.at[i, 'duration_sec'] = 'Unknown'
        print(f"Error for {title} / {artist}: {e}")
    time.sleep(1)

# Search and add extra songs for each movie
extra_rows = []
existing_pairs = set(zip(df['movie_title'], df['song_name']))
```

```
columns = df.columns.tolist()

for full_title in tqdm(df['movie_title'].unique()):
    try:
        if '(' in full_title:
            release_title = full_title.rsplit('(', 1)[0].strip()
            release_year = full_title.rsplit('(', 1)[1].replace(')', '').strip()
        else:
            release_title = full_title
            release_year = ''

        releases = musicbrainzngs.search_releases(release=release_title, date=release_yea
        if not releases.get("release-list"):
            continue

        release = releases["release-list"][0]
        release_id = release["id"]
        release_info = musicbrainzngs.get_release_by_id(release_id, includes=["recordings
        mediums = release_info["release"].get("medium-list", [])

        for medium in mediums:
            for track in medium.get("track-list", []):
                track_title = track["recording"]["title"]
                artist = track["recording"].get("artist-credit", [{}])[0].get("name", "Un
                duration = round(int(track["recording"]["length"]) / 1000) if "length" in

                if (full_title, track_title) not in existing_pairs:
                    new_row = {col: 'Unknown' for col in columns}
                    new_row['movie_title'] = full_title
                    new_row['song_name'] = track_title
                    new_row['performed_by'] = artist
                    new_row['duration_sec'] = duration
                    extra_rows.append(new_row)
                    existing_pairs.add((full_title, track_title))

        time.sleep(1)

    except Exception as e:
        print(f"Error while searching tracks for {full_title}: {e}")
        continue

# Merge and save final enriched dataset
extra_df = pd.DataFrame(extra_rows)
full_df = pd.concat([df, extra_df], ignore_index=True)

path = "/content/common_titles_enriched_full.csv"
full_df.to_csv(path, index=False)
files.download(path)

print(f"Found and added {len(extra_df)} additional songs from MusicBrainz.")
print("Total songs after enrichment:", len(full_df))
```

```
print("Total songs after enrichment: ", len(full_df))
```

```
100%                                                    2217/2217 [57:22<00:00,   1.46s/it]
❌ Error for Ubuhuha  / None: caused by: <urlopen error [SSL: DECRYPTION_FAILED_OR_B
100%                                                    238/238 [09:49<00:00,   2.49s/it]
✅ Found and added 3274 additional songs from MusicBrainz.
🎧 Total songs after enrichment: 5491
```

	movie_title	song_name	performed_by	written_by	composed_by	music_by	lyrics_b
0	12 Angry Men (1957)	Dance of the Cuckoos	Christian Blees	Unknown	Unknown	Marvin Hatley	Unknow
1	12 Years a Slave (2013)	Apache Blessing Song	Chesley Wilson	Chesley Wilson	Unknown	Unknown	Unknow
2	12 Years a Slave (2013)	Awake on Foreign Shores	Colin Stetson	Colin Stetson	Unknown	Unknown	Unknow
3	12 Years a Slave (2013)	Cotton Song	Lead Belly	Nicholas Britell	Unknown	Unknown	Unknow
4	12 Years a Slave (2013)	John	Anita Carter	John Davis	Unknown	Unknown	Unknow
...	...	...	...	...	...	...	.
95	Amadeus (1984)	Die Zauberflöte (The Magic	Ensemble Orchestral de	Wolfgang Amadeus	Unknown	Unknown	Unknow

```
# profiling
profile = ProfileReport(full_df, title="Full Dataset Profiling", explorative=True)
profile.to_notebook_iframe()
```

```
Summarize dataset: 100%                                    35/35 [00:02<00:00, 12.37it/s, Completed]

  0%|          | 0/26 [00:00<?, ?it/s]
  4%|▌         | 1/26 [00:00<00:08,  3.11it/s]
 15%|▌         | 4/26 [00:00<00:02, 10.61it/s]
 23%|▌         | 6/26 [00:00<00:01, 12.92it/s]
 38%|▌         | 10/26 [00:00<00:00, 18.18it/s]
 58%|▌         | 15/26 [00:00<00:00, 24.23it/s]
 69%|▌         | 18/26 [00:01<00:00, 21.48it/s]
100%|▌         | 26/26 [00:01<00:00, 20.91it/s]
Generate report structure: 100%                            1/1 [00:15<00:00, 15.57s/it]

Render HTML: 100%                                          1/1 [00:00<00:00,  1.07it/s]
```

Full Dataset Profiling

Overview

Brought to you by YData

Overview | Alerts 9 | Reproduction

Dataset statistics

Number of variables	26
Number of observations	5491
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	9.7 MiB
Average record size in memory	1.8 KiB

Variable types

Text	18
Categorical	3
Unsupported	5

Variables

Wikipedia Scrapping

```
df_tmdb_common = df_tmdb[df_tmdb['title_with_year'].isin(common_titles)].copy()

# Wikipedia scraping for movies
target_fields = [
    "Directed by", "Produced by", "Written by", "Screenplay by",
    "Story by", "Starring", "Music by"
```

```
    "Story by", "Starring", "Music by",
    "Running time", "Budget"
]

def search_wikipedia_page(title_with_year):
    query = f"{title_with_year} film"
    search_url = f"https://en.wikipedia.org/w/index.php?search={query.replace(' ', '+')}"
    response = requests.get(search_url)
    soup = BeautifulSoup(response.text, 'html.parser')
    result = soup.select_one("ul.mw-search-results li a")

    if result:
        return "https://en.wikipedia.org" + result.get("href")
    else:
        fallback_url = f"https://en.wikipedia.org/wiki/{title_with_year.replace(' ', '_')
        r = requests.get(fallback_url)
        if r.status_code == 200:
            return fallback_url
        return None

def get_infobox_data(wiki_url):
    data = {}
    try:
        response = requests.get(wiki_url)
        if response.status_code != 200:
            return data
        soup = BeautifulSoup(response.text, 'html.parser')
        infobox = soup.find('table', {'class': 'infobox'})
        if not infobox:
            return data
        for row in infobox.find_all("tr"):
            header = row.find("th")
            value = row.find("td")
            if header and value:
                key = header.text.strip()
                val = value.text.strip().replace("\xa0", " ").replace("\n", ", ")
                if key in target_fields:
                    data[key] = val
        data['wiki_url'] = wiki_url
    except:
        pass
    return data

# Scrapping
enriched_rows = []

for title in tqdm(df_tmdb_common['title_with_year']):
    wiki_url = search_wikipedia_page(title)
    if not wiki_url:
        enriched_rows.append({})
        continue
```

```
    wiki_data = get_infobox_data(wiki_url)
    wiki_data['title_with_year'] = title
    enriched_rows.append(wiki_data)
    time.sleep(1)

df_wiki = pd.DataFrame(enriched_rows)

# Merge data
df_merged = pd.merge(df_tmdb_common, df_wiki, on='title_with_year', how='left')

# Drop columns
cols_to_drop = ['original_title', 'runtime', 'budget', 'title', 'Box office', 'Release_da
df_merged.drop(columns=[col for col in cols_to_drop if col in df_merged.columns], inplace

# Clean data from refs etc
def clean_text(val):
    if isinstance(val, str):
        val = re.sub(r'\[\d+\]', '', val)  # remove [1], [23], etc
        return (
            val.replace('\xa0', ' ')
               .replace('\n', ', ')
               .strip()
               .replace(',,', ',')
               .strip(' ,')
        )
    return val

df_merged = df_merged.applymap(clean_text)

# Complete empty cells with Unknown
df_merged.fillna("Unknown", inplace=True)

# Save file
df_merged.to_csv("/content/enriched_tmdb_common_cleaned.csv", index=False)

# Profiling
profile = ProfileReport(df_merged, title="Enriched Wikipedia Dataset", explorative=True)
profile.to_notebook_iframe()

# Count Unknown values
unknown_count_total = (df_merged == "Unknown").sum().sum()

# Count Unknown values per column
unknown_per_column = (df_merged == "Unknown").sum()

print(f"\nTotal number of Unknown values: {unknown_count_total}")
print("\nUnknown values per column:")
print(unknown_per_column[unknown_per_column > 0].sort_values(ascending=False))
```

```
100%|██████████| 238/238 [08:06<00:00,  2.04s/it]
```

```
100%|██████████| 238/238 [00:00<00:00,  2.045/it]
<ipython-input-18-766710acdf1f>:92: FutureWarning: DataFrame.applymap has been deprec
  df_merged = df_merged.applymap(clean_text)
Καθαρισμένο αρχείο αποθηκεύτηκε ως: enriched_tmdb_common_cleaned.csv
```

Summarize dataset: 100% 32/32 [00:01<00:00, 20.59it/s, Completed]

```
  0%|          | 0/23 [00:00<?, ?it/s]
 22%|██        | 5/23 [00:00<00:00, 36.86it/s]
 52%|└└└└└└    | 12/23 [00:00<00:00, 49.22it/s]
 74%|└└└└└└└   | 17/23 [00:00<00:00, 44.40it/s]
100%|└└└└└└└└└| 23/23 [00:00<00:00, 43.96it/s]
```

Generate report structure: 100% 1/1 [00:14<00:00, 14.45s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.37s/it]

Enriched Wikipedia Dataset

Overview

Brought to you by YData

Overview | Alerts 8 | Reproduction

Dataset statistics

Number of variables	23
Number of observations	238
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	500.7 KiB

Variable types

Boolean	1
Text	16
Categorical	4
Unsupported	1
DateTime	1

Average record size in memory	2.1 KiB

Variables

```
Συνολικός αριθμός Unknown τιμών: 647

Unknown ανά στήλη:
Story by                202
Written by              154
Screenplay by           107
Budget                   40
Music by                 38
Produced by              30
Starring                 24
Directed by              22
Running time             17
production_countries      5
production_companies      4
wiki_url                  2
```

```
# Get csv from google sheet
sheet_url = "https://docs.google.com/spreadsheets/d/11U6qIgu24iLTI__cUhJUnRQkftwSb7kod_TB
df = pd.read_csv(sheet_url)

# Wikipedia scrapping for songs
song_fields = [
    "Artist", "Album", "Released", "Recorded", "Studio",
    "Genre", "Length", "Label", "Songwriter(s)", "Producer(s)"
]


def search_wikipedia_page(title):
    query = f"{title} song"
    search_url = f"https://en.wikipedia.org/w/index.php?search={query.replace(' ', '+')}"
    response = requests.get(search_url)
    soup = BeautifulSoup(response.text, 'html.parser')
    result = soup.select_one("ul.mw-search-results li a")

    if result:
        return "https://en.wikipedia.org" + result.get("href")
    else:
        fallback_url = f"https://en.wikipedia.org/wiki/{title.replace(' ', '_')}"
        r = requests.get(fallback_url)
        if r.status_code == 200:
            return fallback_url
        return None
```

```
def get_song_infobox(wiki_url):
    data = {}
    try:
        response = requests.get(wiki_url)
        if response.status_code != 200:
            return data
        soup = BeautifulSoup(response.text, 'html.parser')
        infobox = soup.find('table', {'class': 'infobox'})
        if not infobox:
            return data
        for row in infobox.find_all("tr"):
            header = row.find("th")
            value = row.find("td")
            if header and value:
                key = header.text.strip()
                val = value.text.strip().replace("\xa0", " ").replace("\n", ", ")
                if key in song_fields:
                    data[key] = val
        data["wiki_url"] = wiki_url
    except:
        pass
    return data

# Scrapping
enriched_data = []

for title in tqdm(df["song_name"]):
    wiki_url = search_wikipedia_page(title)
    if not wiki_url:
        enriched_data.append({})
        continue
    wiki_data = get_song_infobox(wiki_url)
    wiki_data["song_name"] = title
    enriched_data.append(wiki_data)
    time.sleep(1)

df_wiki = pd.DataFrame(enriched_data)

# Merge
df_merged = pd.merge(df, df_wiki, on="song_name", how="left")

# Cleaning
def clean_text(val):
    if isinstance(val, str):
        val = re.sub(r"\[\d+\]", "", val)
        return (
            val.replace('\xa0', ' ')
               .replace('\n', ', ')
               .strip()
               .replace('  ', ' ')
```

```
                .replace(' ,, ', ' , ')
                .strip(' ,')
        )
    return val

df_merged = df_merged.applymap(clean_text)

# Complete the Unknown values if information is found
for col in song_fields:
    if col in df.columns:
        df_merged[col] = df_merged[col].mask(df_merged[col] == "Unknown", df_merged[f"{co

# Complete empty cells with Unknown
df_merged.fillna("Unknown", inplace=True)

# Save
df_merged.to_csv("/content/enriched_songs_dataset.csv", index=False)

# Profiling
profile = ProfileReport(df_merged, title="Enriched Songs Dataset", explorative=True)
profile.to_notebook_iframe()

# Count Unknown values
unknown_count_total = (df_merged == "Unknown").sum().sum()
unknown_per_column = (df_merged == "Unknown").sum()

print(f"\n Total number of Unknown values: {unknown_count_total}")
print("\nUnknown values per column:")
print(unknown_per_column[unknown_per_column > 0].sort_values(ascending=False))
```

Upgrade to ydata-sdk

Improve your data and profiling with ydata-sdk, featuring data quality scoring, redundancy detection, outlier identification, text validation, and synthetic data generation.

```
100%|██████████| 5491/5491 [3:42:26<00:00,  2.43s/it]
<ipython-input-2-ffb5cf0aae05>:91: FutureWarning: DataFrame.applymap has been depreca
  df_merged = df_merged.applymap(clean_text)
✅ Εμπλουτισμένο αρχείο αποθηκεύτηκε ως: enriched_songs_dataset.csv
```

Summarize dataset: 100% 45/45 [00:05<00:00, 3.85it/s, Completed]

```
  0%|          | 0/36 [00:00<?, ?it/s]
  3%|▌         | 1/36 [00:00<00:13,  2.56it/s]
 11%|▉         | 4/36 [00:00<00:03,  8.77it/s]
 17%|▉         | 6/36 [00:00<00:02, 10.02it/s]
 22%|▉         | 8/36 [00:00<00:02, 11.34it/s]
 28%|▉         | 10/36 [00:01<00:02, 11.35it/s]
 33%|▉         | 12/36 [00:01<00:02, 11.42it/s]
 39%|▉         | 14/36 [00:01<00:02, 10.72it/s]
 44%|▉         | 16/36 [00:01<00:01, 10.36it/s]
 50%|▉         | 18/36 [00:01<00:01, 10.04it/s]
```

```
 56%|                | 20/36 [00:01<00:01, 10.58it/s]
 61%|                | 22/36 [00:02<00:01, 10.99it/s]
 67%|                | 24/36 [00:02<00:01, 11.45it/s]
 72%|                | 26/36 [00:02<00:00, 11.84it/s]
 78%|                | 28/36 [00:02<00:00,  9.46it/s]
 83%|                | 30/36 [00:02<00:00,  9.39it/s]
 89%|                | 32/36 [00:03<00:00,  9.04it/s]
 94%|                | 34/36 [00:03<00:00,  9.66it/s]
100%|                | 36/36 [00:03<00:00, 10.16it/s]
```

Generate report structure: 100% 1/1 [00:31<00:00, 31.89s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.13s/it]

Enriched Songs Dataset

Overview

Brought to you by YData

Overview | Alerts 8 | Reproduction

Dataset statistics

Number of variables	36
Number of observations	7713
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	658
Duplicate rows (%)	8.5%
Total size in memory	19.6 MiB
Average record size in memory	2.6 KiB

Variable types

Text	30
Categorical	6

Variables

```
🔍 Συνολικός αριθμός 'Unknown' τιμών: 165552

📊 'Unknown' ανά στήλη:
Artist                  7711
conducted_by            7569
under_license_from      7559
composed_by             7518
lyrics_by               7462
music_by                7183
courtesy_of             6675
Studio                  6610
written_by              6203
Recorded                5899
Producer(s)             5795
Songwriter(s)           5663
Length                  5103
Label                   5041
Released                4964
Genre                   4815
budget                  4343
production_countries    3882
production_companies    3847
performed_by            3745
original_language       3742
spoken_languages        3742
overview                3741
genres                  3741
adult                   3741
imdb_id                 3741
original_movie_title    3741
vote_average            3741
popularity              3741
release_date            3741
runtime                 3741
```

```
df = pd.read_csv("/content/enriched_songs_dataset.csv")  # άλλαξε path αν χρειάζεται

# Count Unknown per row
df["unknown_count"] = (df == "Unknown").sum(axis=1)

# Keep the entry with the less "Unknown" values for each (song_name, movie_title)
df_cleaned = df.sort_values("unknown_count").drop_duplicates(subset=["song_name", "movie_t

df_cleaned.drop(columns=["unknown_count"], inplace=True)

# Total number of Unknown cells
```

```
total_unknowns = (df_cleaned == "Unknown").sum().sum()

print(f" Unique entries: {len(df_cleaned)}")
print(f" Total Unknown cells: {total_unknowns}")

df_cleaned.to_csv("/content/enriched_songs_cleaned_best.csv", index=False)
```

```
✅ Τελικές μοναδικές εγγραφές: 5288
❓ Συνολικά 'Unknown' κελιά: 118739
💾 Αποθηκεύτηκε ως enriched_songs_cleaned_best.csv
```

Final Merge

```
songs_df = pd.read_csv("/content/enriched_songs_cleaned_best.csv - enriched_songs_cleaned_b
tmdb_df = pd.read_csv("/content/enriched_tmdb_common_cleaned.csv - enriched_tmdb_common_cle

# Select all coulumns from thw tmdb csv except title_with_year column
tmdb_columns_to_keep = [col for col in tmdb_df.columns if col != "title_with_year"]

# Select columns from songs csv that are not in the tmdb csv
songs_columns_to_keep = [col for col in songs_df.columns if col not in tmdb_columns_to_keep

# Merge based on movie_title and title_with_year
merged_df = pd.merge(
    songs_df[songs_columns_to_keep],
    tmdb_df[tmdb_columns_to_keep + ["title_with_year"]],
    how="left",
    left_on="movie_title",
    right_on="title_with_year"
)

merged_df.drop(columns=["title_with_year"], inplace=True)

# Rename columns
merged_df.rename(columns={
    "Genre": "music_genre",
    "genres": "movie_genre",
    "production_companies": "production_company",
    "production_countries": "production_country",
    "spoken_languages": "spoken_language",
    "Music by": "movie_music_by",
    "wiki_url": "movie_url"
}, inplace=True)

# Cleaning
merged_df = merged_df.applymap(
    lambda x: re.sub(r'\[[^\]]*\]', '', x).strip() if isinstance(x, str) else x
)
```

```
# Count Unknown values
unknown_count = (merged_df == "Unknown").sum().sum()
print(f"Total Unknown values: {unknown_count}")

# Profiling
profile = ProfileReport(merged_df, title="Merged TMDB and Songs", explorative=True)
profile.to_notebook_iframe()

merged_df.to_csv("merged_enriched_songs.csv", index=False)
```

```
<ipython-input-10-194c5a6413c7>:39: FutureWarning: DataFrame.applymap has been deprec
  merged_df = merged_df.applymap(
Σύνολο εμφανίσεων της τιμής 'Unknown': 81706
```

Summarize dataset: 100% 51/51 [00:02<00:00, 15.86it/s, Completed]

```
  0%|          | 0/41 [00:00<?, ?it/s]
  7%|▉         | 3/41 [00:00<00:01, 24.72it/s]
 17%|▉▉        | 7/41 [00:00<00:01, 27.01it/s]
 27%|▉▉▉       | 11/41 [00:00<00:01, 23.09it/s]
 39%|▉▉▉▉      | 16/41 [00:00<00:00, 28.29it/s]
 46%|▉▉▉▉▉     | 19/41 [00:00<00:00, 26.33it/s]
 59%|▉▉▉▉▉▉    | 24/41 [00:00<00:00, 31.49it/s]
 68%|▉▉▉▉▉▉▉   | 28/41 [00:00<00:00, 33.18it/s]
 83%|▉▉▉▉▉▉▉▉  | 34/41 [00:01<00:00, 37.52it/s]
100%|▉▉▉▉▉▉▉▉▉▉| 41/41 [00:01<00:00, 32.42it/s]
```

Generate report structure: 100% 1/1 [00:21<00:00, 21.01s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.37s/it]

Merged TMDB and Songs

Overview

Brought to you by YData

Overview | Alerts 5 | Reproduction

Dataset statistics

Number of variables	41

Variable types

Text	33
Categorical	5

Number of observations	5288
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	16.8 MiB
Average record size in memory	3.2 KiB

Boolean	1
Numeric	1
DateTime	1

Final formatiing for mapping

```
# Load data from Google Sheet
sheet_url = "https://docs.google.com/spreadsheets/d/1SlmKMDsG4zjLMPOn5qPHhAIXDWQsL0BviipP
csv_url = sheet_url.replace("/edit?usp=sharing", "").replace("/edit", "") + "/gviz/tq?tqx

df = pd.read_csv(csv_url)

# Define columns
person_columns = [
    "written_by", "performed_by", "composed_by", "lyrics_by", "music_by",
    "conducted_by", "Songwriter", "Producer", "Directed by", "Screenplay by",
    "Produced by", "Story by", "Written by", "movie_music_by", "Starring"
]
company_columns = ["courtesy_of", "under_license_from", "Label", "production_company"]
studio_columns = ["Studio"]
genre_columns = ["music_genre", "movie_genre"]
country_columns = ["production_country"]
date_columns = ["Released", "Recorded", "release_date"]
duration_columns = ["Length", "Running time"]
value_columns = ["Budget"]
numeric_columns = ["Length", "Running time", "Budget", "popularity", "vote_average"]

def remove_parentheses(value):
    return re.sub(r"\s*\(.*?\)", "", value).strip()

def replace_and_ampersand(val):
```

```
def replace_and_ampersand(val):
    return re.sub(r"\s+(and|&|/)\s+", ", ", val)

def normalize_list(val):
    val = replace_and_ampersand(val)
    return ", ".join([item.strip() for item in val.split(",")])

def capitalize_genre(val):
    if pd.isna(val):
        return val
    return ", ".join([word.strip().capitalize() for word in val.split(",")])

def normalize_date(val, fmt_out):
    try:
        parsed = pd.to_datetime(val, errors='coerce')
        if pd.isna(parsed):
            return np.nan
        return parsed.strftime(fmt_out)
    except:
        return np.nan

def normalize_duration_to_seconds(val):
    if isinstance(val, str):
        val = remove_parentheses(val)
        match = re.search(r"(\d+)", val)
        if match:
            return float(match.group(1)) * 60.0 #in sec
    return np.nan

def normalize_budget_to_usd(val):
    if not isinstance(val, str):
        return np.nan
    val = val.replace(",", "")
    currency = "USD"
    if "€" in val:
        currency = "EUR"
    elif "£" in val:
        currency = "GBP"

    range_match = re.findall(r"(\d+\.?\d*)", val)
    if len(range_match) == 2:
        mid = (float(range_match[0]) + float(range_match[1])) / 2
        num = mid
    else:
        match = re.search(r"(\d+\.?\d*)", val)
        if not match:
            return np.nan
        num = float(match.group(1))

    if "million" in val.lower():
        num *= 1_000_000
```

```
    elif "billion" in val.lower():
        num *= 1_000_000_000

    return round(num, 2)

def clean_studio(val):
    if pd.isna(val) or val.strip() == "":
        return ""
    if val.strip().lower() == "Unknown":
        return "Unknown_Studio"
    return re.split(r"[,(]", val)[0].strip()

# Cleaning
for col in df.columns:
    if col in person_columns + company_columns + country_columns:
        df[col] = df[col].astype(str).apply(remove_parentheses).apply(normalize_list)

    if col in studio_columns:
        df[col] = df[col].astype(str).apply(clean_studio)

    if col in genre_columns:
        df[col] = df[col].astype(str).apply(remove_parentheses).apply(replace_and_ampersa

# Dates
df["Released"] = df["Released"].astype(str).apply(lambda x: normalize_date(x, "%m/%Y"))
df["Recorded"] = df["Recorded"].astype(str).apply(lambda x: normalize_date(x, "%m/%Y"))
df["release_date"] = df["release_date"].astype(str).apply(lambda x: normalize_date(x, "%d

# Durations in sec
df["Length"] = df["Length"].astype(str).apply(normalize_duration_to_seconds)
df["Running time"] = df["Running time"].astype(str).apply(normalize_duration_to_seconds)

# Budget in USD
df["Budget"] = df["Budget"].astype(str).apply(normalize_budget_to_usd)

# Convert numeric columns in NaN if "Unknown" or ""
for col in numeric_columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# Save
output_file = "final_cleaned_songs.csv"
df.to_csv(output_file, index=False)

from google.colab import files
files.download(output_file)
```

Formatting the merged csv from the two datasets before

Formatting the merged csv from the two datasets before scrapping or MusicBrainz for mapping

```
# Load data from Google Sheet
sheet_url = "https://docs.google.com/spreadsheets/d/17jFvi91-yUHyBRNThX_cK2o6Zw2KAGeICBu5
csv_url = sheet_url.replace("/edit?usp=sharing", "").replace("/edit", "") + "/gviz/tq?tqx

df = pd.read_csv(csv_url)

df.drop("original_movie_title", axis=1, inplace=True)

# Format release_date
def normalize_date(val, fmt_out):
    try:
        parsed = pd.to_datetime(val, errors='coerce')
        if pd.isna(parsed):
            return np.nan
        return parsed.strftime(fmt_out)
    except:
        return np.nan

# Format runtime
def normalize_duration_to_seconds(val):
    if isinstance(val, str):
        val = remove_parentheses(val)
        match = re.search(r"(\d+)", val)
        if match:
            return float(match.group(1)) * 60.0 #in sec
    return np.nan

def remove_parentheses(value):
    return re.sub(r"\s*\(.*?\)", "", value).strip()


def normalize_budget_to_usd(val):
    try:
        if pd.isna(val) or val == "" or str(val).lower() == "unknown":
            return np.nan
        return float(str(val).replace(",", "").strip())
    except:
        return np.nan

# Cleaning
df["release_date"] = df["release_date"].astype(str).apply(lambda x: normalize_date(x, "%d
df["runtime"] = df["runtime"].astype(str).apply(normalize_duration_to_seconds)
df["budget"] = df["budget"].apply(normalize_budget_to_usd)

# Convert to numbers
df["popularity"] = pd.to_numeric(df["popularity"], errors="coerce")
```

```
df["vote_average"] = pd.to_numeric(df["vote_average"], errors="coerce")

df.rename(columns={"budget": "Budget"}, inplace=True)
df.rename(columns={"runtime": "Running time"}, inplace=True)

# Save
output_path = "common_titles_mergeed_cleaned.csv"
df.to_csv(output_path, index=False)
files.download(output_path)
```