Data Science

Explanations about how to do the second phases of the final project of the data science course.

Helia Ranjbar
Mehrnaz Hosseini

University of Tehran
Spring 2025

STEP 01

Data Scraping

Data Scraping

We used [The Movie Database (TMDB)] for scraping data.

This is the columns of our dataset:

```
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 18 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   movie_name                   9994 non-null   object 
 1   release_date                 9994 non-null   object 
 2   rating                       9986 non-null   float64
 3   genre                        9994 non-null   object 
 4   run_time                     9984 non-null   object 
 5   certification                9904 non-null   object 
 6   overview                     9986 non-null   object 
 7   tagline                      8414 non-null   object 
 8   director                     9994 non-null   object 
 9   language                     9983 non-null   object 
 10  budget                       9983 non-null   object 
 11  revenue                      9983 non-null   object 
 12  normal_keyword_(rounded)     9994 non-null   object 
 13  tone_keyword_(bold)          9994 non-null   object 
 14  cast                         9994 non-null   object 
 15  reviews                      9994 non-null   object 
 16  content_score                9973 non-null   float64
 17  content_score_description    9973 non-null   object 
dtypes: float64(2), object(16)
```

STEP 02

Database Implementation

- First, we should prepare the data before saving it into the database.

Steps:

1. Runtime Parsing:

   - converts runtime strings into consistent numeric durations. Like '2h 30m' to total minutes.

2. Budget and Revenue Cleaning:

   - Removes dollar signs and commas, handles missing values, and converts to float.

3. Missing Value Normalization:

   - Replaces empty lists and dashes for some columns with NaN for better handling in transformations.

4. Release Date Correction:

   - Converts release dates to datetime objects. Like 17-Feb-95 to 1995-02-17.

5. Extracts and explodes columns with list:

   - Extract columns like director, cast, genre, normal_keyword_(rounded), tone_keyword_(bold) which has multiple values for each row.

Tables Created with SQlite:

1. movies:

   Stores primary metadata for each movie.

   primary key: (movie_id)

Column	Type	Description
movie_id (PK)	INT	Unique identifier
movie_name	VARCHAR(255)	Movie title
release_date	DATE	Official release date
rating	FLOAT	viewer rating
run_time_minutes	INT	Runtime in minutes
certification	VARCHAR(20)	Age-based content
overview, tagline	TEXT	Description and tagline
language	VARCHAR(50)	Original language
budget, revenue	BIGINT	Financial data in USD
content_score	FLOAT	Computed content relevance score
content_score_description	TEXT	Qualitative description of content

2. movie_genres:

Links each movie to one or more genres.

Composite primary key: (movie_id, genre)

Column	Type	Description
movie_id	INT (FK)	References movies.movie_id
genre	VARCHAR(100)	Genre name (e.g., Drama)

3. movie_casts:

Links actors to movies (many-to-many mapping).

Composite Primary Key: (movie_id, actor)

Column	Type	Description
movie_id	INT (FK)	References movies.movie_id
actor	VARCHAR(100)	Actor's name

4. movie_directors:

Stores the directors of each movie (similar to cast structure).

Composite Primay Key: (movie_id, director)

Column	Type	Description
movie_id	INT (FK)	References movies.movie_id
director	VARCHAR(100)	Director's name

5. movie_keywords:

Captures both thematic and tonal keywords per movie.

Composite Primay Key: (normal_keyword_rounded, tone_keyword_bold)

Column	Type	Description
movie_id	INT (FK)	References movies.movie_id
normal_keyword_rounded	VARCHAR(100)	Rounded/general keyword
tone_keyword_bold	VARCHAR(100)	Highlighted tonal keyword

6. movie_reviews:

Stores user reviews for each movie.

Composite Primay Key: (movie_id, writer)

Column	Type	Description
movie_id	INT (FK)	References movies.movie_id
writer	VARCHAR(255)	Name or ID of reviewer
score	FLOAT	Review score as a percentage
review	TEXT	Text content of the review
most_watched_genres	TEXT (JSON)	User's frequently watched genres

STEP 03

Data Querying

1. Find the Top 10 Actors Who Have Played in the Most Different Genres

SELECT mc.cast, COUNT(DISTINCT mg.genre) AS genre_count

FROM movie_casts mc

JOIN movie_genres mg ON mc.movie_id = mg.movie_id

GROUP BY mc.cast

ORDER BY genre_count DESC

LIMIT 10;

	cast	genre_count
0	Samuel L. Jackson	19
1	Joel Edgerton	18
2	Ewan McGregor	18
3	Zac Efron	17
4	Woody Harrelson	17
5	Val Kilmer	17
6	Tom Hanks	17
7	Steve Zahn	17
8	Stephen Root	17
9	Stanley Tucci	17

2. Find the Directors with the Highest Average Revenue

```
SELECT md.director, AVG(m.revenue) AS avg_revenue
FROM movie_directors md
JOIN movies m ON md.movie_id = m.movie_id
GROUP BY md.director
HAVING COUNT(*) > 3
ORDER BY avg_revenue DESC
LIMIT 5;
```

	director	avg_revenue
0	Joe Russo	1.142125e+09
1	Anthony Russo	1.142125e+09
2	James Cameron	1.094838e+09
3	Pierre Coffin	9.270770e+08
4	David Yates	7.995218e+08

3. Yearly Movie Count Trend

```
SELECT STRFTIME('%Y', release_date) AS year, COUNT(*) AS movie_count
FROM movies
GROUP BY year
ORDER BY year DESC;
```

	year	movie_count
0	2025	33
1	2024	203
2	2023	254
3	2022	323
4	2021	368
...	...	...
99	1926	1
100	1925	2
101	1916	1
102	1896	1
103	1895	1

4. Keywords That Drive High Content Scores in Movies

```
SELECT k."tone_keyword_(bold)",
AVG(m.content_score) AS avg_score
FROM movie_keywords k
JOIN movies m ON k.movie_id = m.movie_id
GROUP BY k."tone_keyword_(bold)“
HAVING COUNT(*) > 5
ORDER BY avg_score DESC
LIMIT 10;
```

	tone_keyword_(bold)	avg_score
0	zealous	100.0
1	wry	100.0
2	witty	100.0
3	wistful	100.0
4	whimsical	100.0
5	vindictive	100.0
6	vexed	100.0
7	urgent	100.0
8	understated	100.0
9	unassuming	100.0

5. Actors Who Have Worked Most Often with Christopher Nolan

```
SELECT mc.cast, COUNT(*) AS movie_count
FROM movie_directors md
JOIN movies m ON md.movie_id = m.movie_id
JOIN movie_casts mc ON m.movie_id =
mc.movie_id
WHERE md.director = 'Christopher Nolan’
GROUP BY mc.cast
ORDER BY movie_count DESC
LIMIT 5;
```

	cast	movie_count
0	Michael Caine	6
1	Cillian Murphy	4
2	Christian Bale	4
3	Tom Hardy	3
4	Kenneth Branagh	3

6. Count of Movies by Certification Type

```
SELECT certification, COUNT(*) AS movie_count
FROM movies
GROUP BY certification
ORDER BY movie_count DESC;
```

	certification	movie_count
0	15	2897
1	PG	1242
2	18	998
3	12A	863
4	12	847
...	...	...
82	B	1
83	AA	1
84	21+	1
85	15+	1
86	12PG	1

7. Get movies released after 2015 with a budget over $200 million

```
SELECT movie_name, release_date, budget
FROM movies
WHERE release_date > '2015-01-01' AND
budget > 200000000
ORDER BY release_date DESC;
```

	movie_name	release_date	budget
0	The Electric State	2025-03-14 00:00:00	320000000.0
1	Gladiator II	2024-11-15 00:00:00	310000000.0
2	Red One	2024-11-06 00:00:00	250000000.0
3	Aquaman and the Lost Kingdom	2023-12-21 00:00:00	205000000.0
4	The Marvels	2023-11-10 00:00:00	274800000.0
5	Mission: Impossible - Dead Reckoning Part One	2023-07-10 00:00:00	291000000.0
6	Indiana Jones and the Dial of Destiny	2023-06-30 00:00:00	294700000.0
7	The Flash	2023-06-14 00:00:00	220000000.0
8	The Little Mermaid	2023-05-26 00:00:00	297000000.0
9	Fast X	2023-05-19 00:00:00	340000000.0
10	Guardians of the Galaxy Vol. 3	2023-05-03 00:00:00	250000000.0
11	Ant-Man and the Wasp: Quantumania	2023-02-17 00:00:00	388369742.0
12	Avatar: The Way of Water	2022-12-16 00:00:00	460000000.0
13	Black Panther: Wakanda Forever	2022-11-11 00:00:00	250000000.0
14	Thor: Love and Thunder	2022-07-08 00:00:00	250000000.0
15	No Time to Die	2021-09-30 00:00:00	250000000.0
16	Tenet	2020-08-26 00:00:00	205000000.0
17	Star Wars: The Rise of Skywalker	2019-12-19 00:00:00	416000000.0
18	The Lion King	2019-07-19 00:00:00	260000000.0
19	Avengers: Endgame	2019-04-25 00:00:00	356000000.0
20	Solo: A Star Wars Story	2018-05-25 00:00:00	250000000.0
21	Avengers: Infinity War	2018-04-29 00:00:00	300000000.0
22	Justice League	2017-11-17 00:00:00	300000000.0
23	Transformers: The Last Knight	2017-06-22 00:00:00	217000000.0
24	Pirates of the Caribbean: Dead Men Tell No Tales	2017-05-26 00:00:00	230000000.0
25	The Fate of the Furious	2017-04-14 00:00:00	250000000.0
26	Captain America: Civil War	2016-04-29 00:00:00	250000000.0
27	Batman v Superman: Dawn of Justice	2016-03-25 00:00:00	250000000.0
28	Star Wars: The Force Awakens	2015-12-17 00:00:00	245000000.0
29	Spectre	2015-10-26 00:00:00	245000000.0
30	Avengers: Age of Ultron	2015-04-24 00:00:00	365000000.0

8. Find the Top 5 Most Common Genres

```
SELECT genre, COUNT(*) AS count
FROM movie_genres
GROUP BY genre
ORDER BY count DESC
LIMIT 5;
```

	genre	count
0	Drama	4513
1	Comedy	3584
2	Thriller	2700
3	Action	2313
4	Romance	1689

9. Actors Who Appeared in Movies Released in 2020

```
SELECT mc.cast, m.movie_name,
m.release_date
FROM movie_casts mc
JOIN movies m ON mc.movie_id = m.movie_id
WHERE m.release_date BETWEEN
'2020-01-01' AND '2020-12-31';
```

	cast	movie_name	release_date
0	Cho Yeo-jeong	Parasite	2020-02-07 00:00:00
1	Choi Woo-shik	Parasite	2020-02-07 00:00:00
2	Jang Hye-jin	Parasite	2020-02-07 00:00:00
3	Jung Ji-so	Parasite	2020-02-07 00:00:00
4	Lee Jung-eun	Parasite	2020-02-07 00:00:00
...	...	...	...
2812	Natalie Eva Marie	Hard Kill	2020-09-14 00:00:00
2813	Sergio Rizzuto	Hard Kill	2020-09-14 00:00:00
2814	Swen Temmel	Hard Kill	2020-09-14 00:00:00
2815	Texas Battle	Hard Kill	2020-09-14 00:00:00
2816	Tyler Jon Olson	Hard Kill	2020-09-14 00:00:00

10. Genres by Avg Rating

```
SELECT mg.genre, ROUND(AVG(m.rating), 2)
AS avg_rating
FROM movie_genres mg
JOIN movies m ON mg.movie_id = m.movie_id
GROUP BY mg.genre
ORDER BY avg_rating DESC;
```

	genre	avg_rating
0	Documentary	73.76
1	War	71.10
2	History	71.02
3	Animation	70.51
4	Music	69.41
5	Western	69.13
6	Drama	68.97
7	TV Movie	68.22
8	None	68.00
9	Romance	67.37
10	Fantasy	67.29
11	Family	67.20
12	Crime	66.85
13	Adventure	66.85
14	Mystery	65.55
15	Action	65.51
16	Comedy	65.46
17	Science Fiction	65.30
18	Thriller	64.83
19	Horror	62.70

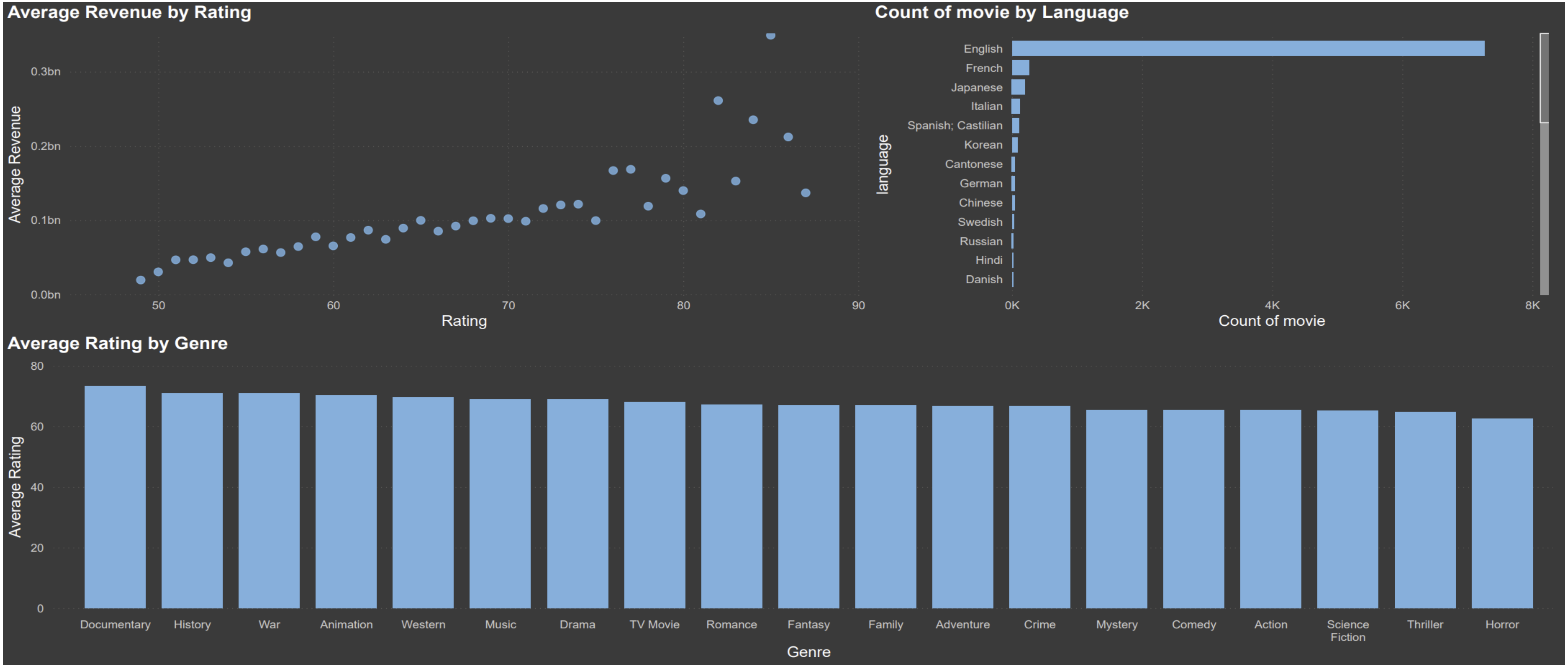
STEP 04

Explanatory Data Analysis(EDA)

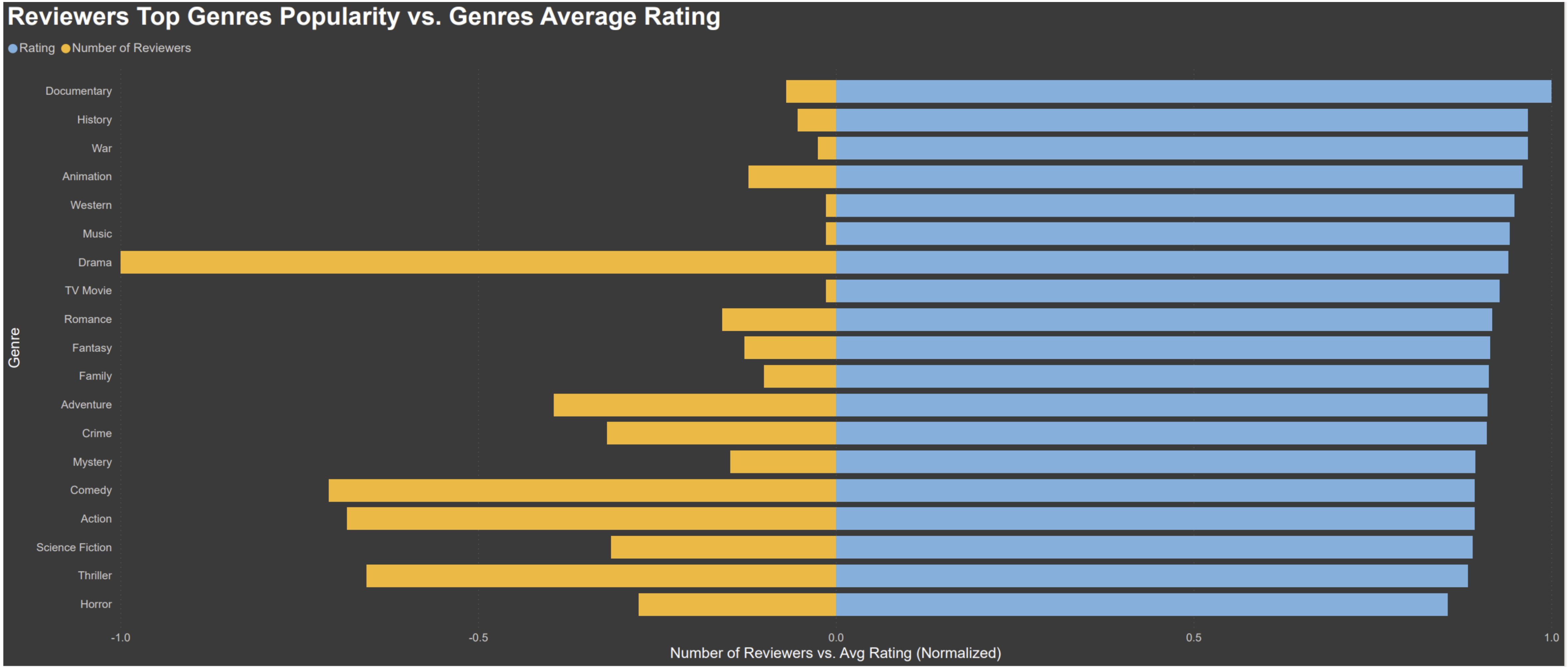
1. analyze how key movie attributes like budget, revenue, and run time vary across different genres for deeper insight and comparison.

	budget					revenue					run_time				
	mean	max	min	std	median	mean	max	min	std	median	mean	max	min	std	median
genre															
Action	5.565472e+07	460000000.0	92.0	6.051682e+07	33000000.0	1.525050e+08	2.923706e+09	428.0	2.576000e+08	57400547.0	108.702983	247.0	4.0	22.529267	107.0
Adventure	6.920311e+07	460000000.0	150.0	6.853590e+07	45000000.0	2.145768e+08	2.923706e+09	3822.0	3.138782e+08	85196485.0	107.613582	242.0	4.0	25.260168	105.0
Animation	5.949356e+07	260000000.0	4986.0	5.589851e+07	45000000.0	1.815432e+08	1.698864e+09	1465.0	2.641466e+08	64197205.0	80.620044	155.0	2.0	27.955468	86.0
Comedy	3.171636e+07	250000000.0	5.0	3.573986e+07	20000000.0	8.950897e+07	1.698864e+09	13.0	1.497581e+08	37801936.0	99.616802	237.0	2.0	19.752763	99.0
Crime	2.787840e+07	340000000.0	120.0	3.253979e+07	19375000.0	6.446329e+07	1.515400e+09	428.0	1.159535e+08	26243131.5	109.364736	247.0	4.0	19.285155	107.0
Documentary	6.000830e+06	60000000.0	5.0	1.052147e+07	2500000.0	2.567601e+07	2.612000e+08	34664.0	4.904969e+07	4606199.0	96.356164	174.0	1.0	22.801615	95.5
Drama	2.408691e+07	310000000.0	1.0	2.892769e+07	15000000.0	5.645946e+07	2.264162e+09	6.0	1.120731e+08	19334145.0	112.915356	367.0	5.0	22.333633	110.0
Family	5.637918e+07	297000000.0	5.0	5.296437e+07	38000000.0	1.773627e+08	1.698864e+09	13.0	2.419069e+08	79300000.0	89.362177	182.0	2.0	25.452002	93.0
Fantasy	5.669824e+07	379000000.0	80.0	5.917628e+07	35000000.0	1.610715e+08	2.923706e+09	3822.0	2.557157e+08	57400000.0	101.466607	242.0	3.0	24.868558	101.0
History	3.389870e+07	200000000.0	100000.0	3.316092e+07	24350000.0	6.136210e+07	9.520000e+08	1281.0	9.290885e+07	26340172.5	129.492986	367.0	32.0	29.345906	125.0
Horror	1.577757e+07	200000000.0	120.0	2.207750e+07	8000000.0	4.598349e+07	7.030000e+08	428.0	7.446504e+07	15187789.0	98.047051	183.0	3.0	15.428976	96.0
Music	2.482494e+07	175000000.0	5.0	2.634738e+07	15000000.0	7.773677e+07	9.183559e+08	3746.0	1.256358e+08	32935319.0	106.391304	188.0	4.0	25.377103	106.0
Mystery	2.544497e+07	340000000.0	275.0	3.034066e+07	15100000.0	5.868945e+07	7.723193e+08	722.0	9.479154e+07	22719750.5	105.935079	188.0	3.0	18.887396	104.0
Romance	2.482163e+07	297000000.0	119.0	2.725965e+07	17000000.0	6.966601e+07	2.264162e+09	921.0	1.262842e+08	28126646.0	108.849023	367.0	4.0	21.555595	106.0
Science Fiction	5.696624e+07	460000000.0	2000.0	6.715280e+07	30000000.0	1.669813e+08	2.923706e+09	3822.0	3.041261e+08	43379092.0	103.549085	192.0	3.0	23.823309	102.0
TV Movie	9.333812e+06	40000000.0	93.0	1.053978e+07	6000000.0	3.746000e+03	3.746000e+03	3746.0	NaN	3746.0	85.765217	140.0	21.0	22.734675	90.0
Thriller	3.005841e+07	340000000.0	120.0	3.663576e+07	18000000.0	7.241940e+07	1.671537e+09	428.0	1.321151e+08	25792310.0	107.082593	247.0	6.0	17.436813	105.0
War	3.480044e+07	175000000.0	20000.0	3.685493e+07	20000000.0	7.387753e+07	5.792000e+08	881.0	1.123853e+08	24911670.0	124.215805	254.0	62.0	26.331243	121.0
Western	2.770487e+07	215000000.0	93.0	3.836005e+07	12500000.0	5.071533e+07	5.329505e+08	9617.0	8.820007e+07	13143056.0	118.114094	218.0	12.0	28.442814	116.0

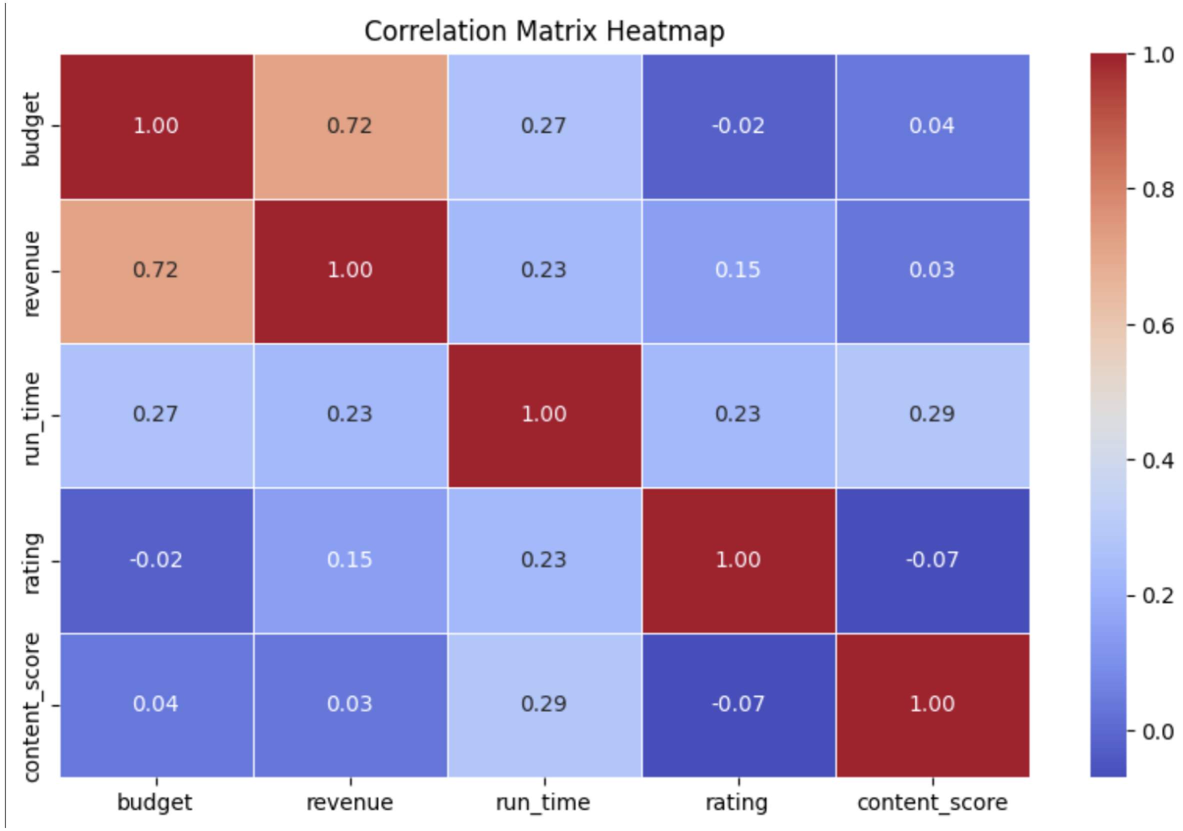
3. evaluate audience performance by connecting movie ratings, revenue outcomes, language distribution, and genre preferences.



4. What does the balance (or lack thereof) between popularity and critical acclaim suggest about audience preferences across different genres?



2. identify and visualize the strength and direction of relationships between numerical movie features, helping uncover patterns and potential dependencies.



STEP 05

Select the appropriate features

movie_id

movie_name

rating

budget_revenue_ratio (New Feature)

certification

overview → Text vectorization

review → Text vectorization

casts → 100 most repeated actors

directors → 100 most repeated directors

normal_keyword_(rounded)→ Text vectorization

tone_keyword_(bold) → Text vectorization

round_keyword → Text vectorization

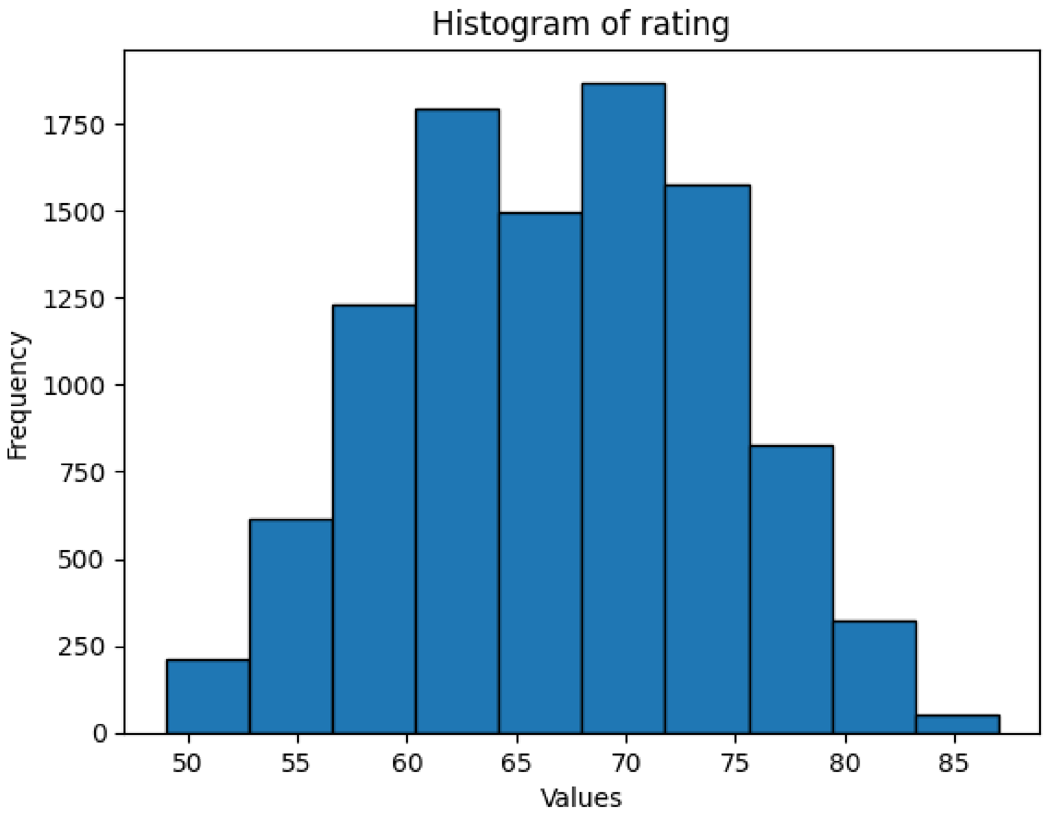
genre (Target Column) → Multi Labeling

STEP 06

Preprocessing

Rating column

- Since this column should be normalized or standardized, it is better to first see the data distribution and then decide which technique to use.
- Because it has a Normal Distribution, it is better to use Standardization. If it had a Uniform Distribution, we would use Normalization.
- We fill the Null values(8 values) with the mean of the column.



Certification column

- This column shows the age limit of each movie. Since the number of its unique values is large, we divided it into 7 subgroups and used ordinal encoding to label it.
- Subgroups:
  1. General Audience (Suitable for all ages)
  2. Parental Guidance Recommended (~7+)
  3. Teen Audiences (Mild to Moderate content, ~12-16)
  4. Older Teens / Restricted (~14-16)
  5. Mature / Restricted (~16-18)
  6. Adults Only (18+)
  7. Special / Regional Ratings/ Null values (Uncategorized)

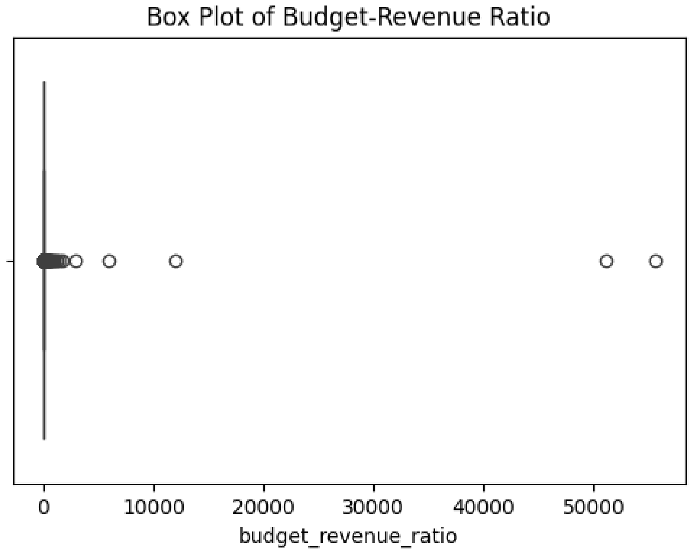
STEP 07

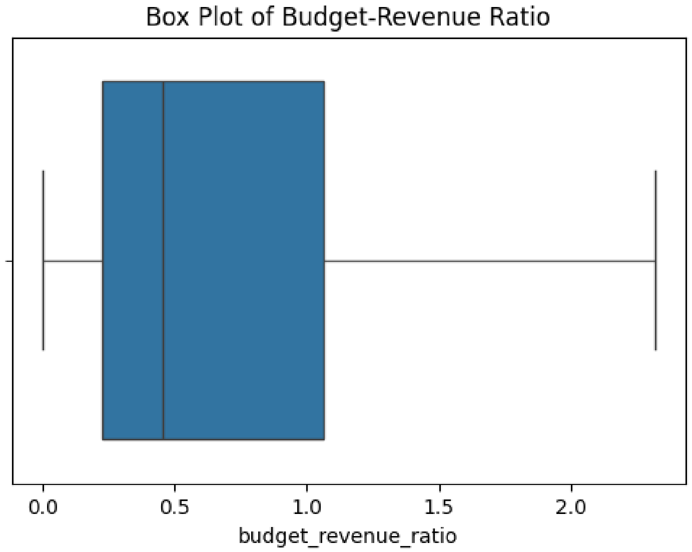
Feature Enginnering

budget_revenue_ratio column

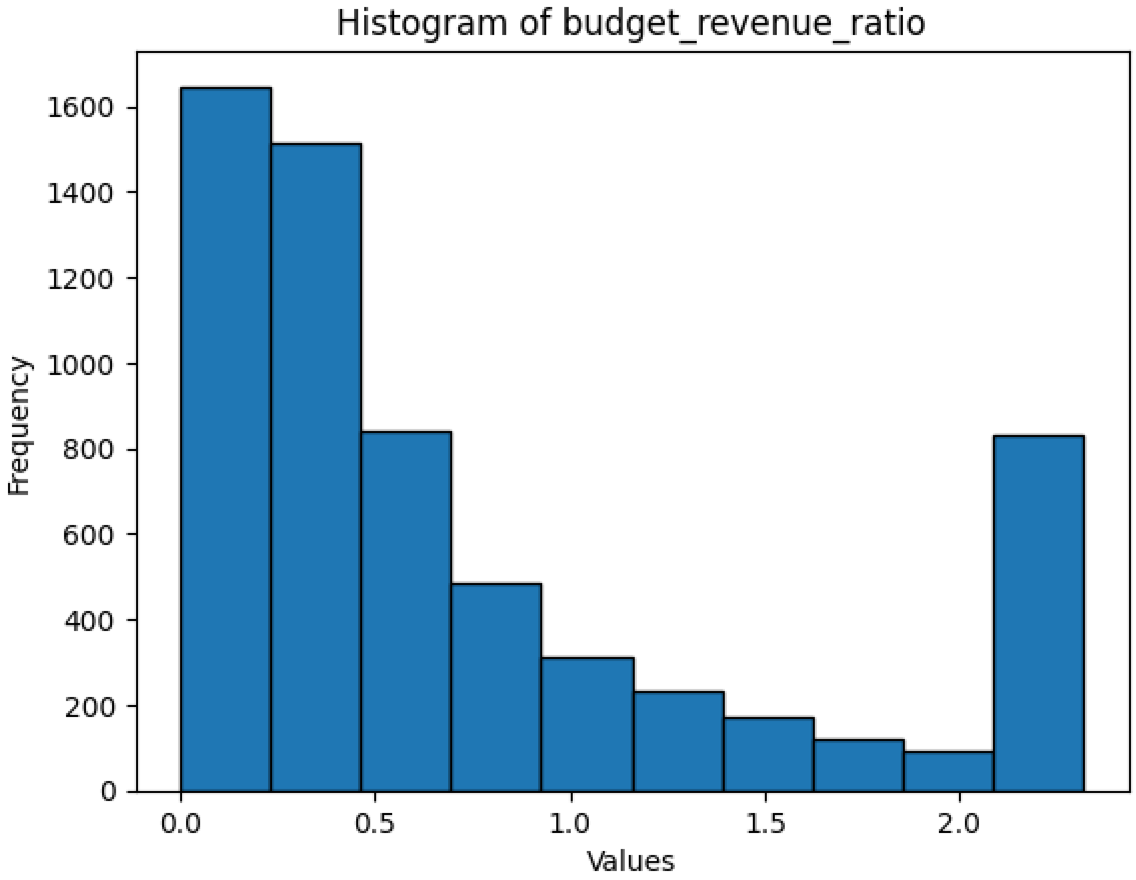
- This is a new column whose values are obtained by dividing the budget column by revenue.
- If we take a look at the box plot of this column, we can see that it has some outlier data, we used IQR to correct this and replaced the values that were more or less than 1.5 times.
- We also fill the Null values of this column with the mean of this column.

```
Q1 = final_df['budget_revenue_ratio'].quantile(0.25)
Q3 = final_df['budget_revenue_ratio'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```





- We also used standardization for this column.



Overview and Review column

- We embedded these two columns that contain text using sentence-bert, which is suitable for semantic extraction.
- Before starting the process, we filled null values with ' ' and removed extra spaces and \n
- Since each movie may have several reviews, we connected all the reviews related to one movie and then embedded them.

Actors and directors columns

- Since there are many actors and directors in the dataset, the approach we have taken to use them is to find 100 actors and 100 directors who have been repeated the most in the dataset and use them to perform one hot encoding for them.
- This means that in each movie, the actors and directors that exist have a value of 1 and otherwise a value of 0.

normal_keyword_(rounded) and tone_keyword_(bold)

- We embedded these two columns, which are the keywords of each movie, using TF-IDF

Genre column

- Since in the genre column, each movie can have several genres, we used multi labeling to encode it.
- For example, if a movie has drama and crime genres, these values will be 1 in the genre of that movie and 0 in the rest of the genres.
- Also, if the movie genre is null, it gets the label unknown.
- In total, we have 20 genres in this dataset.

Checking null valus

- Since we handled null values during the preprocessing and feature engineering, as we can see, none of the columns have null values.

```python
flag = True
for col in final_df.columns.to_list():
    n = final_df[col].isna().sum()
    if n>0:
        flag = False
        print(f"{col} has {n} Null values.")
if flag:
    print('No column contains null values.')
```

✓ 0.0s

```
No column contains null values.
```

STEP 08

Creating an AI Pipeline

```
Command Prompt

Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Helia>cd C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject

C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject>tree /f /a
Folder PATH listing for volume OS
Volume serial number is 7CB9-21C9
C:.
|   .gitattributes
|   dockerfile
|   pipeline.py
|   README.md
|   requirements.txt
|
+---.github
|   \---workflows
|           main.yml
|
+---data
|       finall_df.csv
|       movies_data.csv
|
+---database
|       dataset.db
|
\---scripts
    |   database_connection.py
    |   feature_engineering.py
    |   import_to_db.py
    |   load_data.py
    |   preprocess.py
    |
    \---__pycache__
            database_connection.cpython-39.pyc
            load_data.cpython-39.pyc
            preprocess.cpython-39.pyc
```

This is our directory structure, which is similar to what is set in the project.

This is our pipeline that first loads the dataset from the database and then runs the preprocess and finally feature engineering is run and the final result is saved in a csv file (final_df).

pipeline.py

```python
import subprocess

subprocess.run(["python", "scripts/load_data.py"])
subprocess.run(["python", "scripts/preprocess.py"])
subprocess.run(["python", "scripts/feature_engineering.py"])
```

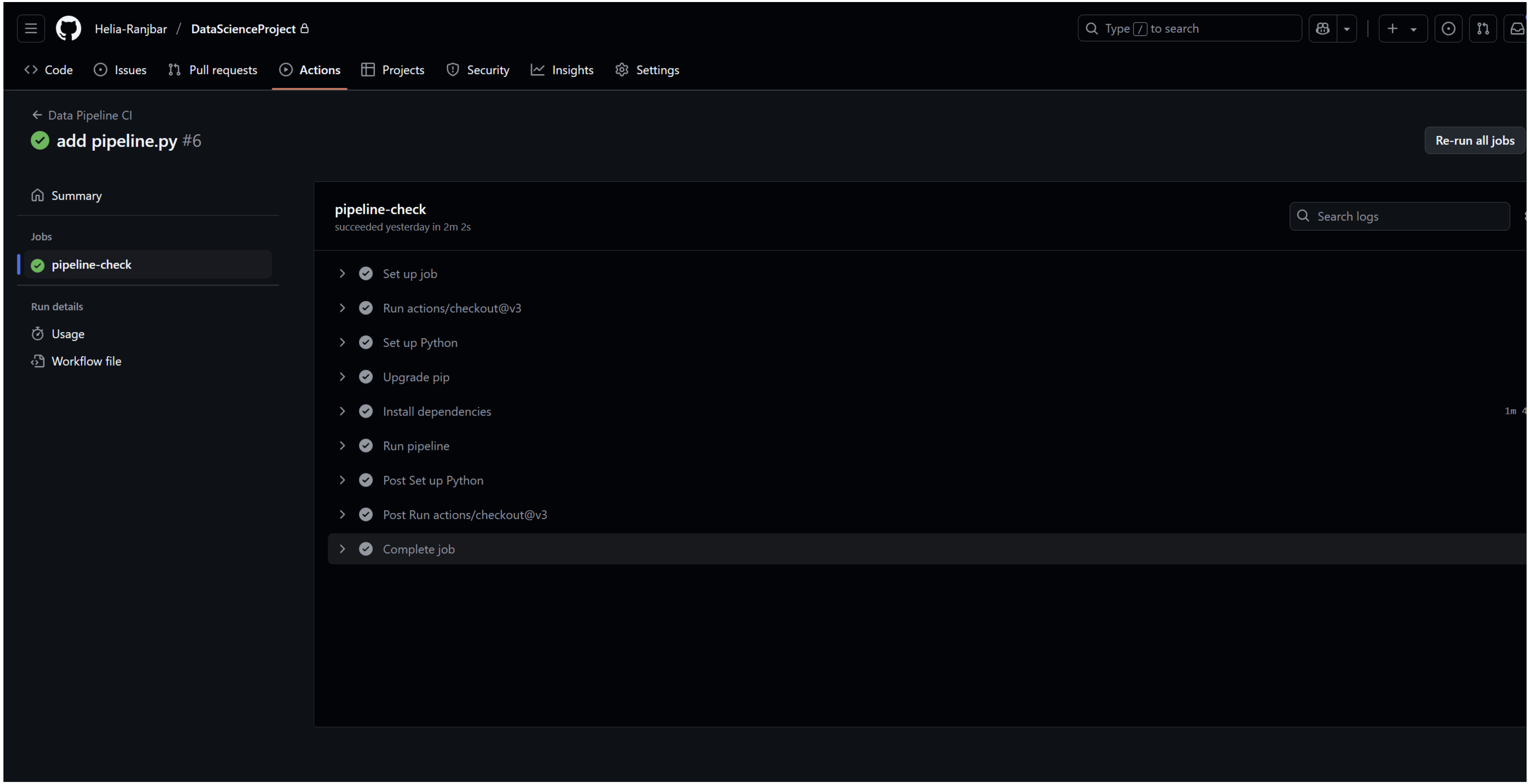
STEP 09

CI/CD Implementation

GitHub Actions workflow

Code Blame 28 lines (21 loc) · 528 Bytes Code 55% faster with GitHub Copilot Raw

```yaml
name: Data Pipeline CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  pipeline-check:
    runs-on: windows-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Upgrade pip
        run: python -m pip install --upgrade pip

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run pipeline
        run: python pipeline.py
```


Helia-Ranjbar / DataScienceProject
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
Data Pipeline CI
add pipeline.py #6
Re-run all jobs
Summary
Jobs
pipeline-check
Run details
Usage
Workflow file
pipeline-check
succeeded yesterday in 2m 2s
Search logs
Set up job
Run actions/checkout@v3
Set up Python
Upgrade pip
Install dependencies
Run pipeline
Post Set up Python
Post Run actions/checkout@v3
Complete job

STEP 10

Docker

building Docker image

```
PS C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject> docker build -t ds_project .
[+] Building 5459.5s (11/11) FINISHED                                                                                              docker:desktop-linux
 => [internal] load build definition from dockerfile                                                                                              0.1s
 => => transferring dockerfile: 1.45kB                                                                                                            0.1s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                                                1.9s
 => [internal] load .dockerignore                                                                                                                 0.0s
 => => transferring context: 2B                                                                                                                   0.0s
 => [internal] load build context                                                                                                                 0.2s
 => => transferring context: 38B                                                                                                                  0.2s
 => [1/6] FROM docker.io/library/python:3.9-slim@sha256:bef8d69306a7905f55cd523f5604de1dde45bbf745ba896dbb89f6d15c727170                          0.0s
 => => resolve docker.io/library/python:3.9-slim@sha256:bef8d69306a7905f55cd523f5604de1dde45bbf745ba896dbb89f6d15c727170                          0.0s
 => CACHED [2/6] WORKDIR /app                                                                                                                     0.0s
 => [3/6] RUN apt-get update && apt-get install -y --no-install-recommends     build-essential     gcc     g++     git     curl     wget     gnupg     unixodbc-dev     214.6s
 => [4/6] COPY requirements.txt .                                                                                                                 0.2s
 => [5/6] RUN pip install --upgrade pip                                                                                                           9.2s
 => [6/6] RUN pip install -r requirements.txt                                                                                                  3810.2s
 => exporting to image                                                                                                                         1421.1s
 => => exporting layers                                                                                                                        1155.9s
 => => exporting manifest sha256:ca74418568ba2f569448795f745540ff70e11c3da1dd3e005f95efbad12e0578                                                 0.1s
 => => exporting config sha256:c541fe56e8ef0b024884cf4452de608316e57709321338e0460986cf19608ed8                                                   0.2s
 => => exporting attestation manifest sha256:2721bde17ee99d1469baabaee67104dd39fe51b9be96f26c8ef0c896ab152b66                                     0.2s
 => => exporting manifest list sha256:b3f88d58414358dd26356b789aa83dcc9e7a1ba8ff31bb20efe272bdbb8c6395                                            0.1s
 => => naming to docker.io/library/ds_project:latest                                                                                              0.0s
 => => unpacking to docker.io/library/ds_project:latest                                                                                         264.4s
```

Run Container from Image

```
PS C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject> docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
ds_project   latest    b3f88d584143   26 minutes ago   19.9GB
neo4j        latest    ca5da0c4cb59   6 months ago     855MB
neo4j        5.13.0    6c6003e890c7   17 months ago    802MB
```

```
PS C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject> docker run -d --name ds_p_container ds_pr
oject
241b1cdc137ab584a9e10e31675beec4d353bd0d133e82d873a6d6322830ed1f
PS C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject> docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED          STATUS          PORTS                                                      NAMES
241b1cdc137a   ds_project      "jupyter notebook --…"   13 seconds ago   Up 11 seconds   8888/tcp                                                   ds_p_container
ebe6e4ee2dbc   neo4j:5.13.0    "tini -g -- /startup…"   4 months ago     Up 3 days       0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp   string-neo4j
PS C:\Users\Helia\OneDrive\Desktop\Uni\Computer\DataScience\Project_P2_220701062_220701051\DataScienceProject_Copy\DataScienceProject>
```