Application: Culture, Media and AI Research Assistanship

Rishabh Bijani

July 29, 2025

Contents

Chapter 1

Introduction

1.1 Brief Description

This report is an accompaniment to the GitHub repository (available here submitted to apply for the Culture, Media and AI research assistantship. The repository contains the code, datasets and plots to complete the coding task. This report explains the data pipeline and has the plots.

1.2 Justification of Proposed Theme

Apart from analysis the prescribed themes of Hindu-Muslim Relations, Gender Relations and Nationalism, this project also explores LGBTQIA+ themes the justification for which is as follows

1. Historical Marginalisation
   LGBTQIA+ characters and narratives have long existed on the fringes of Bollywood cinema. However, movies like Aligarh (2015), Ek Ladki Ko Dekha Toh Aisa Laga (2019), and Shubh Mangal Zyada Saavdhan (2020) have opened space for more authentic representation. This makes LGBTQIA+ themes a timely and meaningful domain for computational tracking and analysis.

2. Legal and Cultural Inflection Point
   The 2018 Supreme Court verdict decriminalizing homosexuality marks a watershed moment in India's socio-political landscape. Tracing how Bollywood responded to (or ignored) this moment offers a quasi-historical perspective on cultural change.

Chapter 2

Data Pipeline

2.1 Preliminaries

1. For reproducing the code, please create a folder titled `ra_app` on your desktop.
2. The Kaggle dataset (available here) when downloaded as a zip file is saved as a folder titled `archive`. Transfer the file `archive/1950-2019/bollywood_full_1950-2019.csv` to the folder `Desktop/ra_app` just created. Alternatively, this file has been uploaded onto the repository under `/cleaned_datasets` and can be downloaded directly from there.
3. Please note, TMDb API does not work properly in India without a VPN connection.
4. The data pipeline consists of 10 scripts which are explained in the following section.

2.2 Code Scripts

2.2.1 random_sampling.R

1. **Description:**
   Used to randomly select a sample of 100 movies from the Kaggle dataset under the set.seed 05042004 (Note: No particular reason for this, it is just my date of birth) for reproducibility.
2. **Input File:**
   archive/1950-2019/bollywood_full_1950-2019.csv from the Kaggle dataset.
3. **Output File:**
   movies_random_sample.csv

The code script is as follows -

```r
#Kaggle Dataset - Random Sampling
#24th July 2025

#WARNING: The following command will clear RStudio environment
#Clearing RStudio
rm(list=ls())

#Importing and calling packages and libraries
library(readr)
library(tidyverse)
library(ggplot2)
library(dplyr)
library(lubridate)

#Loading full Bollywood dataset

#PLEASE CHANGE FILE PATH ACCORDINGLY
#INPUT FILE: bollywood_full_1950-2019.csv

getwd()
setwd("/Users/rishabhbijani/Desktop/ra_app")
full_bollywood_list <- read_csv("bollywood_full_1950-2019.csv")

#Inspecting this dataset
str(full_bollywood_list)

#Converting year_of_release into integer
#Filtering the dataset for movies released post 2010
full_bollywood_list <- full_bollywood_list %>%
  mutate(year_of_release_int = as.integer(year_of_release)) %>%
  filter(year_of_release_int >2010)

#Checking for duplicates
length(unique(full_bollywood_list$imdb_id))

#No of unique imdb_id = 816
#No of rows = 819
#Therefore, checking for and removing duplicates
table(duplicated(full_bollywood_list$imdb_id))
full_bollywood_unique <- full_bollywood_list[!duplicated(full_bollywood_list$
    imdb_id), ]

#Random Sampling
set.seed(05042004)
movies_randomly_sampled <- full_bollywood_unique[sample(nrow(full_bollywood_
    unique), 100), ]

#Cleaning random sample
colnames(movies_randomly_sampled)
```

```
movies_randomly_sampled <- movies_randomly_sampled %>%
  select(-c(title_y,
            year_of_release)) %>%
  rename("title_kaggle" = "title_x",
         "story_kaggle" = "story",
         "summary_kaggle" = "summary",
         "tagline_kaggle" = "tagline")

#Checking structure of the dataset
str(movies_randomly_sampled)
colnames(movies_randomly_sampled)
summary(movies_randomly_sampled)

#Saving the random sample as a csv
write_csv(movies_randomly_sampled, "movies_random_sample.csv")
```

2.2.2 api_test.ipynb

1. **Description:**
   Used to test proper functioning of the TMDb and OpenSubtitles APIs. Please note, TMDb API does not work properly in India without a VPN connection. All keys and tokens are hardcoded therefore, no input/ output files.

The code script is as follows -

```
#25th July 2025
#API Test Script

import requests

# Keys and IDs
# NOTE: Contains API keys and passwords
tmdb_api_key = "ef74866feee9084817794614ffbae21d"
opensub_api_key = "V8cxCT3494611SyHgQRVHbetkQEZVTHW"
opensub_username = "rishabh_bijani"
opensub_password = "Rishabhb2004!"
sample_imdb_id = "tt8207768"

#API Test Key
print("\  n     Testing TMDb API...")
tmdb_url = f"https://api.themoviedb.org/3/find/{sample_imdb_id}?api_key={tmdb_
    api_key}&external_source=imdb_id"
try:
    resp = requests.get(tmdb_url)
    print("TMDb Status Code:", resp.status_code)
    if resp.status_code == 200:
        overview = resp.json()["movie_results"][0].get("overview", "")
        print("    TMDb description retrieved.")
        print("Overview:", overview[:200], "...")
    elif resp.status_code == 401:
        print("    TMDb: Invalid API key.")
    elif resp.status_code == 403:
        print("    TMDb: Blocked (VPN may be required).")
    else:
        print("       TMDb Error:", resp.text)
except Exception as e:
    print(f"    TMDb Request failed: {e}")

#OpenSubtitles API Test
print("\  n     Testing OpenSubtitles login with token...")
login_url = "https://api.opensubtitles.com/api/v1/login"
headers_login = {
    "Api-Key": opensub_api_key,
    "Content-Type": "application/json",
    "User-Agent": "RAApp/1.0.0"
}
login_payload = {
    "username": opensub_username,
    "password": opensub_password
}

try:
    login_resp = requests.post(login_url, headers=headers_login, json=login_
       payload)
    print("Login Status Code:", login_resp.status_code)
    if login_resp.status_code == 200:
        token = login_resp.json()["token"]
        print("    OpenSubtitles token acquired.")
    else:
        print("    Login failed:", login_resp.text)
        token = None
except Exception as e:
    print(f"    OpenSubtitles Login Error: {e}")
    token = None

#OpenSubtitles Search and Download Test
if token:
    print("\  n     Searching subtitles for", sample_imdb_id)
    headers_auth = {
        "Authorization": f"Bearer {token}",
        "Api-Key": opensub_api_key,
        "Content-Type": "application/json",
        "Accept": "application/json",
        "User-Agent": "RAApp/1.0.0"
    }
    search_url = f"https://api.opensubtitles.com/api/v1/subtitles?imdb_id={
       sample_imdb_id[2:]}&languages=en"

    try:
        sub_resp = requests.get(search_url, headers=headers_auth)
        print("Subtitle Search Status:", sub_resp.status_code)
        if sub_resp.status_code == 200:
            results = sub_resp.json().get("data", [])
            print(f"    Found {len(results)} subtitles.")
            if results:
                file_id = results[0]["attributes"]["files"][0]["file_id"]
                print("Top subtitle file ID:", file_id)

                # Download .srt link
                download_url = "https://api.opensubtitles.com/api/v1/download"
                dl_resp = requests.post(download_url, headers=headers_auth,
                    json={"file_id": file_id})
                if dl_resp.status_code == 200:
                    dl_link = dl_resp.json()["link"]
                    print("    Subtitle download link:", dl_link)

                    # Optional actual download
                    srt = requests.get(dl_link)
                    if srt.status_code == 200:
                        with open(f"{sample_imdb_id}.srt", "wb") as f:
                            f.write(srt.content)
                        print(f"    Subtitle file saved as {sample_imdb_id}.srt
                            ")
                    else:
                        print("    Failed to retrieve subtitle file.")
                else:
                    print("    Failed to generate download link.")
        else:
            print("    Subtitle search failed:", sub_resp.text)
    except Exception as e:
        print(f"    Subtitle search error: {e}")
```

2.2.3 primary_data_download.ipynb

1. **Description:**
   Downloads posters, subtitles and descriptions of the movies in the random sample. Creates the folders data/descriptions, data/posters, data/subtitles. Also, logs the downloads. Descriptions, posters and subtitles are named with the IMDb ID of the movie. The data source for subtitles is OpenSubtitles and the data source for posters and descriptions is TMDb API.

2. **Input File:**
   movies_random_sample.csv

3. **Output Files:**
   (a) data/description
   (b) data/posters
   (c) data/subtitles

(d) download_report.csv

(e) log_downloads.txt

The code script is as follows -

```
#Bollywood Data Downloader
# 25th July 2025

#Importing and Calling Packages
import os
import time
import requests
import pandas as pd
from tqdm import tqdm

#Keys, File Paths and Passwords
#NOTE: Contains API Keys and Passwords
tmdb_api_key = "ef74866feee9084817794614ffbae21d"
tmdb_user_agent = "RAApp/1.0.0"

opensub_api_key = "V8cxCT3494611SyHgQRVHbetkQEZVTHW"
opensub_username = "rishabh_bijani"
opensub_password = "Rishabhb2004!"

#PLEASE CHANGE FILE PATH ACCORDINGLY
# PLEASE ENSURE - movies_random_sample.csv is in Desktop/ra_app
csv_path = os.path.expanduser("~/Desktop/ra_app/movies_random_sample.csv")

base_folder = os.path.expanduser("~/Desktop/ra_app/data")
log_file = os.path.join(base_folder, "log_downloads.txt")

#Creating Folders
os.makedirs(os.path.join(base_folder, "description"), exist_ok=True)
os.makedirs(os.path.join(base_folder, "posters"), exist_ok=True)
os.makedirs(os.path.join(base_folder, "subtitles"), exist_ok=True)

#Logging Function
def log(msg):
    print(msg)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(msg + "\n")

#Login into OpenSubtitles
log("       Logging in to OpenSubtitles...")
login_url = "https://api.opensubtitles.com/api/v1/login"
headers_login = {
    "Api-Key": opensub_api_key,
    "Content-Type": "application/json",
    "User-Agent": tmdb_user_agent
}
payload = {
    "username": opensub_username,
    "password": opensub_password
}

login_resp = requests.post(login_url, headers=headers_login, json=payload)
if login_resp.status_code == 200:
    opensub_token = login_resp.json()["token"]
    log("    OpenSubtitles login successful.")
else:
    log("    OpenSubtitles login failed: " + login_resp.text)
    raise Exception("Cannot proceed without OpenSubtitles token.")

#Download Loop
df = pd.read_csv(csv_path)
imdb_ids = df["imdb_id"].dropna().unique()
report_rows = []

headers_auth = {
    "Authorization": f"Bearer {opensub_token}",
    "Api-Key": opensub_api_key,
    "Content-Type": "application/json",
    "Accept": "application/json",
    "User-Agent": tmdb_user_agent
}

for imdb_id in tqdm(imdb_ids):
    log(f"\n=== Processing {imdb_id} ===")
    desc_status = poster_status = sub_status = "   "

    # --- TMDb DESCRIPTION ---
    try:
        tmdb_url = f"https://api.themoviedb.org/3/find/{imdb_id}?api_key={tmdb_
            api_key}&external_source=imdb_id"
        tmdb_resp = requests.get(tmdb_url)
        if tmdb_resp.status_code == 200:
            movie = tmdb_resp.json().get("movie_results", [{}])[0]
            overview = movie.get("overview", "")
            desc_path = os.path.join(base_folder, "description", f"{imdb_id}.
                txt")
            with open(desc_path, "w", encoding="utf-8") as f:
                f.write(overview)
            desc_status = "   "
            log("    Description saved.")
        else:
            log(f"    TMDb description failed (status {tmdb_resp.status_code})"
                )
    except Exception as e:
        log(f"    TMDb description error: {e}")

    # --- TMDb POSTER ---
    try:
        poster_path = movie.get("poster_path", "")
        if poster_path:
            poster_url = f"https://image.tmdb.org/t/p/original{poster_path}"
            poster_resp = requests.get(poster_url)
            if poster_resp.status_code == 200:
                poster_file = os.path.join(base_folder, "posters", f"{imdb_id}.
                    png")
                with open(poster_file, "wb") as f:
                    f.write(poster_resp.content)
                poster_status = "   "
                log("    Poster saved.")
            else:
                log("    Poster download failed.")
        else:
            poster_status = "      "
            log("      No poster path available.")
    except Exception as e:
        log(f"    Poster error: {e}")

    # --- OpenSubtitles SUBTITLES ---
    try:
        search_url = f"https://api.opensubtitles.com/api/v1/subtitles?imdb_id={
            imdb_id[2:]}&languages=en"
        sub_resp = requests.get(search_url, headers=headers_auth)
        if sub_resp.status_code == 200:
            results = sub_resp.json().get("data", [])
            if results:
                file_id = results[0]["attributes"]["files"][0]["file_id"]
                dl_resp = requests.post(
                    "https://api.opensubtitles.com/api/v1/download",
                    headers=headers_auth,
                    json={"file_id": file_id}
                )
                if dl_resp.status_code == 200:
                    dl_link = dl_resp.json()["link"]
                    srt_resp = requests.get(dl_link)
                    if srt_resp.status_code == 200:
                        sub_path = os.path.join(base_folder, "subtitles", f"{
                            imdb_id}.srt")
                        with open(sub_path, "wb") as f:
                            f.write(srt_resp.content)
                        sub_status = "   "
                        log("    Subtitle saved.")
                    else:
                        log("    Subtitle download failed.")
                else:
                    log("    Subtitle download link error.")
            else:
                sub_status = "      "
                log("       No subtitles found.")
        else:
            log("    Subtitle search failed.")
    except Exception as e:
        log(f"    Subtitle error: {e}")

    # --- Append to Report ---
    report_rows.append({
        "imdb_id": imdb_id,
        "description_status": desc_status,
        "poster_status": poster_status,
        "subtitle_status": sub_status
    })

#CSV Download Report
report_df = pd.DataFrame(report_rows)
report_df.to_csv(os.path.join(base_folder, "download_report.csv"), index=False)
log("       Saved report to download_report.csv")
log("    Script finished.")
```

2.2.4 secondary_data_download.ipynb

1. **Description:**
   primary_data_download.ipynb leaves certain subtitles, posters and descriptions undownloaded as they are not available at TMDb. This script downloads the remaining posters and descriptions. Data Sources: For posters: poster_path in movies_random_sample.csv (which is in turn taken from Kaggle), for descriptions: wiki_link i.e. Wikipedia in movies_random_sample.csv.

2. **Input File:**
   movies_random_sample.csv

3. Output File:

   (a) data/description

   (b) data/posters

   (c) data/subtitles

The code script is as follows -

```
# Bollywood Retry Script: Descriptions & Posters
# 25th July 2025

import os
import requests
import pandas as pd
from bs4 import BeautifulSoup
from tqdm import tqdm

# === File and folder setup ===
#NOTE: PLEASE CHANGE FILE PATHS ACCORDINGLY
# PLEASE ENSURE movies_random_sample.csv is in Desktop/ra_app
# REST AUTOMATIC CARRY FORWARD FROM primary_data_download.ipynb

csv_path = os.path.expanduser("~/Desktop/ra_app/movies_random_sample.csv")
base_folder = os.path.expanduser("~/Desktop/ra_app/data")
desc_folder = os.path.join(base_folder, "description")
poster_folder = os.path.join(base_folder, "posters")
log_file = os.path.join(base_folder, "log_retry_from_dryrun.txt")
summary_file = os.path.join(base_folder, "missing_files_report.csv")

# Ensure folders exist
os.makedirs(desc_folder, exist_ok=True)
os.makedirs(poster_folder, exist_ok=True)

# Logging function
def log(msg):
    print(msg)
    with open(log_file, "a", encoding="utf-8") as f:
        f.write(msg + "\n")

# Load data
df = pd.read_csv(csv_path)
imdb_ids = df["imdb_id"].tolist()

# Get existing files
existing_desc = set(f.replace(".txt", "") for f in os.listdir(desc_folder) if f
    .endswith(".txt"))
existing_poster = set(f.replace(".jpg", "").replace(".png", "") for f in os.
    listdir(poster_folder) if f.endswith((".jpg", ".png")))

# Identify and retry missing files
missing_data = []
for _, row in tqdm(df.iterrows(), total=len(df), desc="Checking & Downloading
    Missing"):
    imdb_id = row["imdb_id"]
    wiki_link = row["wiki_link"]
    poster_url = row["poster_path"]

    # === Check whats missing ===
    missing_desc = imdb_id not in existing_desc
    missing_post = imdb_id not in existing_poster

    # === Log & track missing ===
    if missing_desc or missing_post:
        missing_data.append({
            "imdb_id": imdb_id,
            "missing_description": missing_desc,
            "missing_poster": missing_post
        })
        log(f"{imdb_id}     description: {'   ' if missing_desc else '   '},
            poster: {'   ' if missing_post else '   '}")

    # === Download Description ===
    if missing_desc and pd.notna(wiki_link):
        try:
            resp = requests.get(wiki_link, timeout=10)
            if resp.status_code == 200:
                soup = BeautifulSoup(resp.text, "html.parser")
                content = soup.select("div.mw-parser-output > p")
                paragraphs = [p.get_text().strip() for p in content if p.get_
                    text().strip()]
                text = "\n".join(paragraphs[:2])
                if text:
                    desc_path = os.path.join(desc_folder, f"{imdb_id}.txt")
                    with open(desc_path, "w", encoding="utf-8") as f:
                        f.write(text)
                    log("    Description downloaded from Wikipedia.")
                else:
                    log("       Wikipedia has no valid paragraph content.")
            else:
                log(f"    Failed to fetch Wikipedia page (status {resp.status_
                    code}).")
        except Exception as e:
            log(f"    Error fetching Wikipedia: {e}")

    # === Download Poster ===
    if missing_post and pd.notna(poster_url):
        try:
            headers = {
                "User-Agent": "Mozilla/5.0",
                "Referer": "https://en.wikipedia.org/"
            }
            resp = requests.get(poster_url, headers=headers, timeout=10)
            if resp.status_code == 200:
                poster_path = os.path.join(poster_folder, f"{imdb_id}.jpg")
                with open(poster_path, "wb") as f:
                    f.write(resp.content)
                log("    Poster downloaded from Wikimedia.")
            else:
                log(f"    Failed to download poster (status {resp.status_code})
                    .")
        except Exception as e:
            log(f"    Error downloading poster: {e}")

# === Save report of what was missing ===
if missing_data:
    pd.DataFrame(missing_data).to_csv(summary_file, index=False)
    log(f"\  n     Missing files report saved to {summary_file}")
else:
    log("\ n   No missing files found. All data is complete.")

log("\  n     Script finished.")
```

2.2.5 ai_gender_inf.ipynb

1. **Description:**
   Uses TMDb API to extract the name and biographical sketch of the director of the movie. Uses Open AI API to prompt GPT-4 to infer the gender of the director and assign a confidence score. Also logs failed cases. Please enter the API key provided over email in the script for it to run. The prompt used in this included in the appendix.
2. **Input File:**
   movies_random_sample.csv
3. **Output File:**

   (a) director_gender_full_output.csv
   (b) director_gender_failed_cases.csv

The code script is as follows -

```
#AI-Based Gender Inference
#Director Name and Bio using TMDb
#25th July 2025

import os
import pandas as pd
import requests
import json
import time
from tqdm import tqdm
from openai import OpenAI

# ---        API Keys ---
TMDB_API_KEY = "ef74866feee9084817794614ffbae21d"

#NOTE: PLEASE REPLACE KEY WITH THE ONE PROVIDED OVER EMAIL
OPENAI_API_KEY = ""

#PLEASE ENDURE movies_random_sample.csv is in Desktop/ra_app
# ---        Load Movie Dataset ---
csv_path = os.path.expanduser("˜/Desktop/ra_app/movies_random_sample.csv")
df = pd.read_csv(csv_path)

# ---        Output Containers ---
output_rows = []
failed_imdb_ids = []
```

```
# ---        TMDb Helper Functions ---
def get_tmdb_id(imdb_id):
    try:
        url = f"https://api.themoviedb.org/3/find/{imdb_id}"
        params = {"api_key": TMDB_API_KEY, "external_source": "imdb_id"}
        r = requests.get(url, params=params)
        data = r.json()
        return data['movie_results'][0]['id'] if data['movie_results'] else
            None
    except:
        return None

def get_movie_title(tmdb_id):
    try:
        url = f"https://api.themoviedb.org/3/movie/{tmdb_id}"
        params = {"api_key": TMDB_API_KEY}
        r = requests.get(url, params=params)
        return r.json().get("title", "Unknown Title")
    except:
        return "Unknown Title"

def get_directors(tmdb_id):
    try:
        url = f"https://api.themoviedb.org/3/movie/{tmdb_id}/credits"
        params = {"api_key": TMDB_API_KEY}
        r = requests.get(url, params=params)
        data = r.json()
        return [(p['id'], p['name']) for p in data.get("crew", []) if p.get("
            job") == "Director"]
    except:
        return []

def get_bio(person_id):
    try:
        url = f"https://api.themoviedb.org/3/person/{person_id}"
        params = {"api_key": TMDB_API_KEY}
        r = requests.get(url, params=params)
        return r.json().get("biography", "")
    except:
        return ""

# ---        GPT Inference Function ---
def infer_gender(name, bio, max_retries=3):
    client = OpenAI(api_key=OPENAI_API_KEY)  #     Correctly define client
        inside the function

    prompt = f"""
Based on the following name and biography of a film director associated with
    Indian cinema (especially Bollywood), what is the most likely gender of
    this person?

Name: {name}
Biography: {bio}

Respond in JSON format with two fields:
{{
  "gender": "male" or "female",
  "confidence": a number between 0.0 and 1.0 indicating how confident you are
      in this classification
}}
"""
    for attempt in range(max_retries):
        try:
            response = client.chat.completions.create(
                model="gpt-4",
                messages=[{"role": "user", "content": prompt}],
                temperature=0,
                timeout=60
            )
            raw = response.choices[0].message.content.strip()
            result = json.loads(raw)
            return result["gender"].lower(), float(result["confidence"])
        except Exception as e:
            print(f"       GPT error (attempt {attempt+1}) for {name}: {e}")
            time.sleep(2 + attempt * 2)

    return "unknown", 0.0

# ---        Main Loop Over Movies ---
for idx, row in tqdm(df.iterrows(), total=df.shape[0]):
    imdb_id = row['imdb_id']

    tmdb_id = get_tmdb_id(imdb_id)
    if not tmdb_id:
        print(f"    TMDb ID not found for IMDb ID {imdb_id}")
        failed_imdb_ids.append({"imdb_id": imdb_id, "reason": "tmdb_id_not_
            found"})
        continue

    movie_title = get_movie_title(tmdb_id)
    directors = get_directors(tmdb_id)

    if not directors:
        print(f"       No directors found for: {movie_title} ({imdb_id})")
        failed_imdb_ids.append({"imdb_id": imdb_id, "reason": "no_director_
            found"})
        continue

    for person_id, director_name in directors:
        bio = get_bio(person_id)
        gender, confidence = infer_gender(director_name, bio)

        output_rows.append({
            "imdb_id": imdb_id,
            "movie_title": movie_title,
            "director_name": director_name,
            "director_bio": bio,
            "inferred_gender": gender,
            "confidence_score": confidence
        })

        print(f"    {movie_title}     {director_name}     {gender} ({confidence
            :.2f})")

        time.sleep(1)  # polite rate limit delay

# ---        Save Output Files ---
output_df = pd.DataFrame(output_rows)
output_path = os.path.expanduser("~/Desktop/ra_app/director_gender_full_output.
    csv")
output_df.to_csv(output_path, index=False)

if failed_imdb_ids:
    fail_df = pd.DataFrame(failed_imdb_ids)
    fail_path = os.path.expanduser("~/Desktop/ra_app/director_gender_failed_
        cases.csv")
    fail_df.to_csv(fail_path, index=False)
    print(f"\  n     Logged {len(failed_imdb_ids)} failed cases to: {fail_path}
        ")

print(f"\ n   Saved final output to: {output_path}")
```

2.2.6 web_scraping.ipynb

1. **Description:**
   Scarpes the Bollywood Hungama website (available here for the box office collection (in Cr) of all bollywood movies released from 2010 to 2024.

2. **Input File: None**

3. **Ouput File:**
   bollywood_box_office_2010_2024.csv

The code script is as follows -

```
# 26th July 2025
# Web Scraping Script

import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry

# Retry strategy
session = requests.Session()
retry = Retry(
    total=5,
    backoff_factor=1,
    status_forcelist=[429, 500, 502, 503, 504],
    raise_on_status=False,
)
adapter = HTTPAdapter(max_retries=retry)
session.mount("https://", adapter)
session.mount("http://", adapter)

# Target years
years = list(range(2010, 2025))

# Output data
data = []

# Loop through years
for year in years:
    url = f"https://www.bollywoodhungama.com/box-office-collections/
        filterbycountry/IND/{year}"
    print(f"       Scraping year {year}     {url}")

    try:
        response = session.get(url, headers={'User-Agent': 'Mozilla/5.0'},
            timeout=10)
        soup = BeautifulSoup(response.text, "html.parser")
        time.sleep(2)  # wait for table to load

        rows = soup.find_all("tr", class_="table-row")
        if not rows:
            print(f"       No data table found for year {year}")
            continue

        for row in rows:
            cells = row.find_all("td", class_="table-cell")
            if len(cells) >= 6:
                data.append({
                    "year": year,
                    "movie_name": cells[0].get_text(strip=True),
                    "release_date": cells[1].get_text(strip=True),
                    "opening_day": cells[2].get_text(strip=True),
                    "opening_weekend": cells[3].get_text(strip=True),
                    "week_1": cells[4].get_text(strip=True),
                    "lifetime_collection": cells[5].get_text(strip=True)
                })

        print(f"    {len(rows)} movies scraped for year {year}")

    except Exception as e:
        print(f"    Error scraping year {year}: {e}")

# Save to CSV
df = pd.DataFrame(data)
df.to_csv("bollywood_box_office_2010_2024.csv", index=False)
print("       Data saved to 'bollywood_box_office_2010_2024.csv'")
```

2.2.7 fuzzy_matching_new.ipynb

1. **Description:**
   Applies fuzzy string matching to both, title_kaggle and original_title in movies_random_sample.csv with movie_name in bollywood_box_office_2010_2024.csv i.e. the output file of web_scarping.ipynb. Finds the best match (in terms of confidence score) of the two and retains that. If confidence score is less than 70 for both, return NA. Adds lifetime_collection_matched column to movies_random_sample.csv.

2. **Input File:**
   (a) movies_random_sample.csv
   (b) bollywood_box_office_2010_2024.csv

3. **Output File:**
   movies_random_sample.csv (appended)

The code script is as follows -

```
#Fuzzy Matching
#26th July 2025

import pandas as pd
import os
from rapidfuzz import process, fuzz
import re

# === Load datasets ===
#NOTE: PLEASE ENSURE both, movies_random_sample.csv and bollywood_box_office_
    2010_2024.csv
```

```
# are in Desktop/ra_app
box_office_path = os.path.expanduser("~/Desktop/ra_app/bollywood_box_office_
    2010_2024.csv")
movies_sample_path = os.path.expanduser("~/Desktop/ra_app/movies_random_sample.
    csv")

box_office_df = pd.read_csv(box_office_path)
movies_df = pd.read_csv(movies_sample_path)

# === Clean and normalize movie names ===
def clean_title(title):
    if pd.isna(title):
        return ""
    title = str(title).lower()
    title = re.sub(r'\([^)]*\)', '', title)  # Remove text in parentheses
    title = re.sub(r'[^a-z0-9\s]', '', title)  # Remove punctuation
    title = re.sub(r'\s+', ' ', title).strip()  # Normalize whitespace
    return title

# Add cleaned columns
movies_df["clean_original_title"] = movies_df["original_title"].apply(clean_
    title)
movies_df["clean_title_kaggle"] = movies_df["title_kaggle"].apply(clean_title)
box_office_df["clean_movie_name"] = box_office_df["movie_name"].apply(clean_
    title)

# === Match function ===
def match_movie(clean_title1, clean_title2, choices, threshold=70):
    match1 = process.extractOne(clean_title1, choices, scorer=fuzz.ratio) if
        clean_title1 else None
    match2 = process.extractOne(clean_title2, choices, scorer=fuzz.ratio) if
        clean_title2 else None

    best_match = None
    if match1 and match2:
        best_match = match1 if match1[1] >= match2[1] else match2
    elif match1:
        best_match = match1
    elif match2:
        best_match = match2

    if best_match and best_match[1] >= threshold:
        return best_match
    else:
        return None

# === Full Run: Match and print all 100 results ===
print("\n=== Full Fuzzy Matching Results (All 100 Movies) ===\n")

for idx, row in movies_df.iterrows():
    # Special case override: force NA for "Humsafar"
    if row['title_kaggle'].strip().lower() == "humsafar":
        print(f"       {row['title_kaggle']}")
        print("       Matched With: NA")
        print("       Lifetime Collection: NA")
        print("-" * 60)
        continue

    match_result = match_movie(row["clean_original_title"], row["clean_title_
        kaggle"], box_office_df["clean_movie_name"])

    print(f"       {row['title_kaggle']}")

    if match_result:
        matched_title_clean, score, match_idx = match_result
        matched_row = box_office_df.iloc[match_idx]
        print(f"       Matched With: {matched_row['movie_name']} (Score: {score
            })")
        print(f"       Lifetime Collection: {matched_row['lifetime_collection
            ']}")
    else:
        print("       Matched With: NA")
        print("       Lifetime Collection: NA")

    print("-" * 60)

# === Store lifetime collection results ===
lifetime_collections = []

for idx, row in movies_df.iterrows():
    # Special case: force NA for "Humsafar"
    if row['title_kaggle'].strip().lower() == "humsafar":
        lifetime_collections.append(None)
        continue

    match_result = match_movie(row["clean_original_title"], row["clean_title_
        kaggle"], box_office_df["clean_movie_name"])

    if match_result:
        _, score, match_idx = match_result
        matched_row = box_office_df.iloc[match_idx]
        lifetime_collections.append(matched_row['lifetime_collection'])
    else:
        lifetime_collections.append(None)

# Add matched results to original DataFrame and save
movies_df["lifetime_collection_matched"] = lifetime_collections
output_path = os.path.expanduser("~/Desktop/ra_app/movies_random_sample.csv")
movies_df.to_csv(output_path, index=False)
print(f"\ n   Lifetime collections appended and saved to: {output_path}")
```

2.2.8 sentiment_analysis_new.ipynb

1. **Description:**
   Carries out sentiment analysis of bollywood movies using GPT-4 as the LLM. If either description or subtitles of a movie are missing, skipped for analysis. If subtitles have less than 100 characters, also skilled for analysis.
   If a certain theme out of the four (i.e Hindu-Muslim Relations, Gender Relations, LGBTQIA+ Themes, Nationalism) is present, it is assigned a score on each of the following axes: positive-negative, progressive-conservative and inclusionary-exclusionary. Scores are assigned on a continuum of -1 and +1 where -1 represents most negative, conservative and exclusionary sentiments and +1 represents most positive, inclusionary and progressive sentiments.
   NOTE: Open AI API key provided over email is required to reproduce the code.

2. **Input Files:**
   (a) movies_random _sample.csv
   (b) data/description
   (c) data/subtitles

3. **Output File:** sentiment_raw_output.txt

The code script is as follows -

```
# Bollywood Sentiment Analysis     Raw Output to Console and .txt
# 26 July 2025

import os
import openai
import pandas as pd
import re
import time

# === API KEY ===
#Please enter key provided over email.
openai.api_key = ""

# === SETTINGS ===
DATA_FOLDER = os.path.expanduser("~/Desktop/ra_app/data")
CSV_PATH = os.path.expanduser("~/Desktop/ra_app/movies_random_sample.csv")
OUTPUT_TXT_PATH = os.path.expanduser("~/Desktop/ra_app/sentiment_raw_output.txt
    ")
MODEL = "gpt-4o"
CHUNK_SIZE = 6000

# === CLEANING FUNCTION ===
def clean_text(text):
    text = re.sub(r"<[^>]+>", "", text)
    text = re.sub(r"{[^}]+}", "", text)
    text = re.sub(r"\d+\n", "", text)
    text = re.sub(r"\d{2}:\d{2}:\d{2},\d{3} --> .*", "", text)
    text = re.sub(r"\n{2,}", "\n", text)
    return text.strip()

# === PROMPT TEMPLATE (Verbatim) ===
PROMPT_TEMPLATE = """
You are a social scientist and analyst who is tasked with analysing cultural
    and political themes in the subtitles and descriptions of Bollywood (i.e.
    Indian film industry) films since 2010.

Your task is to
1. Detect whether each of the following themes appears
   1. Hindu-Muslim Relations
   2. Gender Relations
   3. Nationalism
   4. LGBTQIA+ Themes

2. For each theme that is present, assess it on the following axes
   1. Exclusionary v/s Inclusionary
   2. Negative v/s Positive
   3. Conservative v/s Progressive

3. For each axis, assign a score on a continuous scale from **-1 to +1**:
   - A score of **-1** represents the most Exclusionary, Negative, or
       Conservative representation possible.
   - A score of **+1** represents the most Inclusionary, Positive, or
       Progressive representation possible.
   - A score of **0** indicates a neutral or balanced portrayal.

Thematic definitions of the themes are as follows

1. Hindu-Muslim Relations
In India, Hindus form about 79% of the population whereas Muslims form about
    14%. Hindu-Muslim relations are characterised by periods of both, amity,
    cooperation and syncretism and also with strife and violence.

2. Gender Relations
UNICEF defines gender relations as follows: A specific sub-set of social
    relations uniting men and women as social groups in a particular community.
     Gender relations intersect with all other influences on social relations
        age, ethnicity, race, religion     to determine the position and
    identity of people in a social group. Since gender relations are a social
    construct, they can be changed.

3. Nationalism
Oxford Reference defines nationalism as follows: A political ideology and
    associated movement intended to realize or further the aims of a nation,
    most notably for independent self-government in a defined territory. In a
    broader sense, nationalism also refers to sentiments of attachment to or
    solidarity with a national identity or purpose.

4. LGBTQIA+ Themes
Cambridge Dictionary defines LQBTBQIA+ themes as follows: abbreviation for
    lesbian, gay, bisexual, transgender, queer (or questioning), intersex, and
    asexual (or ally): relating to or characteristic of people whose sexual
    orientation is not heterosexual (= sexually or romantically attracted to
    women if you are a man, and men if you are a woman) or whose gender
    identity is not cisgender (= having a gender that matches the physical body
     you were born with).

Definitions of the axes are as follows:

1. Exclusionary v/s Inclusionary
Cambridge Dictionary defines exclusionary to be: limited to only one group or
    particular groups of people, in a way that is unfair; resulting in a person
     or thing not being included in something.
Cambridge Dictionary defines inclusion to be: the act of including someone or
    something as part of a group, list, etc., or a person or thing that is
    included; the idea that everyone should be able to use the same facilities,
     take part in the same activities, and enjoy the same experiences,
    including people who have a disability or other disadvantage.

2. Negative v/s Positive
```

```
Cambridge Dictionary defines negative to be: not expecting good things, or
    likely to consider only the bad side of a situation; bad or harmful.
Cambridge Dictionary defines positive to be: full of hope and confidence, or
    giving cause for hope and confidence.

3. Conservative v/s Progressive
Cambridge Dictionary defines conservative to be: not usually liking or trusting
     change, especially sudden change.
Cambridge Dictionary defines progressive to be: Progressive ideas or systems
    are new and modern, encouraging change in society or in the way that things
     are done.

---

Please analyze the following **Movie Description and Subtitle Transcript**
    carefully and respond using this structure:

For each of the 4 themes, do the following:
- Presence: Present / Not Present / Ambiguous
- If Present, assign a score between -1 and +1 on each of the following axes:
  - ExclusionaryInclusionary
  - NegativePositive
  - ConservativeProgressive

---

Movie Plot Description:
{description}

---

Subtitle Transcript:
{subtitles}
"""

# === OPENAI CALLS ===
def call_openai(prompt):
    response = openai.chat.completions.create(
        model=MODEL,
        messages=[{"role": "user", "content": prompt}],
        temperature=0
    )
    return response.choices[0].message.content.strip()

def analyze_with_fallback(description, subtitles):
    try:
        return call_openai(PROMPT_TEMPLATE.format(description=description,
            subtitles=subtitles))
    except Exception:
        print("       Full input failed. Retrying with chunked input...")
        return analyze_in_chunks(description, subtitles)

def analyze_in_chunks(description, subtitles):
    chunks = [subtitles[i:i+CHUNK_SIZE] for i in range(0, len(subtitles), CHUNK
        _SIZE)]
    combined = ""
    for idx, chunk in enumerate(chunks):
        print(f"      Analyzing chunk {idx+1}/{len(chunks)}...")
        try:
            resp = call_openai(PROMPT_TEMPLATE.format(description=description,
                subtitles=chunk))
            combined += f"\n\n[Chunk {idx+1}]\n{resp}"
        except Exception as e:
            print(f"      Failed on chunk {idx+1}: {e}")
            continue
    return combined.strip()

# === MAIN LOOP ===
df = pd.read_csv(CSV_PATH)
output_lines = []

for _, row in df.iterrows():
    imdb_id = row["imdb_id"]
    title = row["original_title"]
    desc_path = os.path.join(DATA_FOLDER, "description", f"{imdb_id}.txt")
    sub_path = os.path.join(DATA_FOLDER, "subtitles", f"{imdb_id}.srt")

    if not (os.path.exists(desc_path) and os.path.exists(sub_path)):
        print(f"       Missing files for {imdb_id}     {title}. Skipping.")
        continue

    with open(desc_path, "r", encoding="utf-8") as f:
        description = clean_text(f.read())
    with open(sub_path, "r", encoding="utf-8", errors="ignore") as f:
        subtitles = clean_text(f.read())

    if len(subtitles) < 100:
        print(f"       Subtitles too short for {imdb_id}     {title}. Skipping.
            ")
        continue

    print(f"\  n     Analyzing: {title} ({imdb_id})")
    try:
        response = analyze_with_fallback(description, subtitles)
    except Exception as e:
        print(f"    Error for {imdb_id}     {title}: {e}")
        continue

    full_entry = f"=== {title} ({imdb_id}) ===\n{response}\n{'='*80}\n"
    print(full_entry)
    output_lines.append(full_entry)

    time.sleep(1.5)

# === SAVE TO .TXT ===
with open(OUTPUT_TXT_PATH, "w", encoding="utf-8") as f:
    f.writelines(output_lines)

print(f"\  n     All results saved to {OUTPUT_TXT_PATH}")
```

2.2.9 text_parsing.ipynb

1. **Description:**
   Parses sentiment_raw_ouput.txt and saves the measures of sentiment in a .csv file.
2. **Input File:** sentiment_raw_output.txt
3. **Output File:** sentiment_parsed_output.csv

The code script is as follows -

```
# ---
# jupyter:
#   jupytext:
#     formats: py:percent
#     text_representation:
#       extension: .py
#       format_name: percent
#       format_version: '1.3'
#       jupytext_version: 1.17.2
#   kernelspec:
#     display_name: Python [conda env:base] *
#     language: python
#     name: conda-base-py
# ---

# %%
#Parsing Test File
#26th July 2025

import re
import pandas as pd
import os

# Define theme names in order and corresponding short prefixes for columns
themes = [
    ("Hindu-Muslim Relations", "hindu_muslim"),
    ("Gender Relations", "gender_relations"),
    ("Nationalism", "nationalism"),
    ("LGBTQIA+ Themes", "lgbtq")
]

# Axis labels as they appear in the text (note the en dash character in each)
axis_labels = [
    "ExclusionaryInclusionary",
    "NegativePositive",
    "ConservativeProgressive"
]

#NOTE: Please change file paths accordingly
#Input File: sentiment_raw_output.txt as the result of sentiment_analysis_new.
    ipynb

# Input and output paths
input_path = os.path.expanduser("~/Desktop/ra_app/sentiment_raw_output.txt")
output_path = os.path.expanduser("~/Desktop/ra_app/sentiment_parsed_output.csv"
    )

# Ensure the output directory exists
os.makedirs(os.path.dirname(output_path), exist_ok=True)

# Read the entire text content
with open(input_path, "r", encoding="utf-8") as f:
    content = f.read()

# Split content into lines for easier parsing
lines = content.splitlines()

# Regular expression to identify movie title lines (=== Title (ttXXXXXXX) ===)
title_pattern = re.compile(r'^={3,}\s*(?P<title>.*?)\s*\((?P<imdb_id>tt\d+)\)\s
    *={3,}$')

movies_data = []  # list to collect each movie's data as a dictionary

# Find all movie title lines and their indices
title_indices = []
for i, line in enumerate(lines):
    match = title_pattern.match(line)
    if match:
        title_indices.append((i, match.group("title"), match.group("imdb_id")))
title_indices.sort(key=lambda x: x[0])  # sort by line number just in case

# Loop through each identified movie block
for idx, title, imdb_id in title_indices:
    next_title_idx = None
    for j, _, _ in title_indices:
        if j > idx:
            next_title_idx = j
            break
    end_idx = next_title_idx if next_title_idx is not None else len(lines)
    block_lines = lines[idx:end_idx]
    while block_lines and re.match(r'^[=\s]+$', block_lines[-1]) and not title_
        pattern.match(block_lines[-1]):
        block_lines.pop()

    movie_record = {
        "title": title.strip(),
        "imdb_id": imdb_id.strip()
    }

    presence_line_indices = []
    for k, line in enumerate(block_lines):
        if re.search(r'Presence\**\s*:', line):
            presence_line_indices.append(k)
    presence_line_indices.sort()

    if len(presence_line_indices) != 4:
        presence_line_indices = []
        for k, line in enumerate(block_lines):
            if "Presence:" in line.replace("*", ""):
                presence_line_indices.append(k)
        presence_line_indices.sort()

    for t_index, (theme_name, prefix) in enumerate(themes):
        presence_status = None
        excl_incl = neg_pos = cons_prog = None

        if t_index < len(presence_line_indices):
            pres_idx = presence_line_indices[t_index]
            pres_line = block_lines[pres_idx]
            if ":" in pres_line:
                _, status_part = pres_line.split(":", 1)
            else:
                status_part = ""
            status_part = status_part.strip().strip("*").rstrip(".")
            if status_part.lower().startswith("present"):
                presence_status = "Present"
            elif status_part.lower().startswith("not"):
                presence_status = "Not Present"
            elif status_part.lower().startswith("ambiguous"):
                presence_status = "Ambiguous"
            else:
                presence_status = status_part or None
        else:
            presence_status = None

        if presence_status == "Present":
            start_line = pres_idx + 1
            if t_index < len(presence_line_indices) - 1:
                end_line = presence_line_indices[t_index + 1]
            else:
                end_line = len(block_lines)

            for axis_label in axis_labels:
                value = None
                for line in block_lines[start_line:end_line]:
                    if axis_label in line:
                        clean_line = line.replace("*", "")
                        if ":" in clean_line:
                            _, val_part = clean_line.split(":", 1)
                        else:
                            val_part = ""
                        match = re.search(r'[-+]?[\d]+(?:\.[\d]+)?', val_part)
                        if match:
                            num_str = match.group(0)
                            num_str = num_str.replace("\u2212", "-").replace("\
                               u2013", "-")
                            try:
                                value = float(num_str)
                            except ValueError:
                                value = None
                        else:
                            if "N/A" in val_part or "n/a" in val_part:
                                value = None
                        break
                if axis_label.startswith("Exclusionary"):
                    excl_incl = value
                elif axis_label.startswith("Negative"):
                    neg_pos = value
                elif axis_label.startswith("Conservative"):
                    cons_prog = value

        movie_record[f"{prefix}_presence"] = presence_status
        movie_record[f"{prefix}_exclusionary_inclusionary"] = excl_incl
        movie_record[f"{prefix}_negative_positive"] = neg_pos
        movie_record[f"{prefix}_conservative_progressive"] = cons_prog

    movies_data.append(movie_record)

df = pd.DataFrame(movies_data)
print(df.to_string(index=False))
df.to_csv(output_path, index=False)
```

2.2.10 plotting_visualisation.R

1. **Description:**
   Generates plots from sentiment_parsed_output.csv
2. **Input File:**

   (a) movies_random_sample.csv
   (b) sentiment_parsed_output.csv

3. **Output Files:**
   (a) bollywood_average_score.png
   (b) bollywood_stacked_area_plot.png
   (c) bollywood_theme_frequency.png
   (d) bollywood_average_score_exclusionary_inclusive.png
   (e) bollywood_average_score_positive_negative.png

The code script is as follows -

```
#Plotting and Visualisation
#26th July 2025

#WARNING: The following command will clear RStudio environment
#Clearing RStudio
rm(list = ls())

#Loading and calling libraries and packages
library(ggplot2)
library(dplyr)
library(tidyverse)
library(readr)
library(ggthemes)
library(scales)

#Loading datasets
#NOTE: Pls set working directory/file path accordingly
#NOTE: sentiment_parsed_output.csv is the ouput file of text_parsing.ipynb
getwd()
analysed_dataset <- read_csv("/Users/rishabhbijani/Desktop/ra_app/sentiment_
    parsed_output.csv")
random_sample <- read_csv("/Users/rishabhbijani/Desktop/ra_app/movies_random_
    sample.csv")

#Checking the structure of the dataset
str(analysed_dataset)

#Checking unique values in presence columns
print(unique(analysed_dataset$hindu_muslim_presence))
print(unique(analysed_dataset$gender_relations_presence))
print(unique(analysed_dataset$nationalism_presence))
print(unique(analysed_dataset$lgbtq_presence))

#Recoding "Ambigous" as "Not Present"
analysed_dataset <- analysed_dataset %>%
  mutate(across(ends_with("_presence"), ~ ifelse(. == "Ambiguous", "Not Present
      ", .)))
```

```
#PLOT 1: Theme Frequency
# Merge to get release year
merged <- analysed_dataset %>%
  left_join(random_sample %>% select(imdb_id, year_of_release_int), by = "imdb_
      id") %>%
  rename(release_year = year_of_release_int)

# Pivot to long format: one row per movie-theme
theme_presence_long <- merged %>%
  select(imdb_id, title, release_year,
         hindu_muslim_presence,
         gender_relations_presence,
         nationalism_presence,
         lgbtq_presence) %>%
  pivot_longer(
    cols = ends_with("_presence"),
    names_to = "theme",
    values_to = "presence"
  ) %>%
  filter(presence == "Present" & !is.na(release_year))

# Clean theme labels
theme_presence_long$theme <- recode(theme_presence_long$theme,
                                    hindu_muslim_presence = " H i n d u Muslim
                                        Relations",
                                    gender_relations_presence = "Gender
                                        Relations",
                                    nationalism_presence = "Nationalism",
                                    lgbtq_presence = "LGBTQIA+ Themes"
)

# Count how many movies had each theme by year
theme_freq <- theme_presence_long %>%
  group_by(release_year, theme) %>%
  summarise(count = n(), .groups = "drop")

# Plot
plot_frequency <- ggplot(theme_freq, aes(x = release_year, y = count, color =
    theme)) +
  geom_line(linewidth = 1.2) +
  geom_point(size = 2) +
  geom_text(aes(label = count), vjust = -0.7, size = 3.5, show.legend = FALSE)
      +  # numbers on points
  labs(
    title = "Theme Frequency in Bollywood Films (2010  Present  )",
    x = "Release Year",
    y = "Number of Movies",
    color = "Theme"
  ) +
  scale_x_continuous(breaks = seq(2010, 2024, 1)) +
  scale_y_continuous(breaks = pretty_breaks(n = 10)) +  # whole number breaks
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 9),
    legend.box = "horizontal"
  ) +
  guides(color = guide_legend(nrow = 2, byrow = TRUE))

# Display the plot
print(plot_frequency)

#Saving the frequency plot
#NOTE: Pls change the path to save the plot accordingly
ggsave("bollywood_theme_frequency.png", plot = plot_frequency,
       path = "/Users/rishabhbijani/Desktop/ra_app/plots")

#PLOT 2: Average score across conservative-progressive axes by theme
# Add year to main dataset
merged_sentiment <- analysed_dataset %>%
  left_join(random_sample %>% select(imdb_id, year_of_release_int), by = "imdb_
      id") %>%
  rename(release_year = year_of_release_int) %>%
  filter(!is.na(release_year))

# Pivot longer for conservativeprogressive axis
sentiment_long <- merged_sentiment %>%
  select(release_year, imdb_id,
         ends_with("_presence"),
         ends_with("conservative_progressive")) %>%
  pivot_longer(
    cols = ends_with("conservative_progressive"),
    names_to = "theme_axis",
    values_to = "score"
  ) %>%
  filter(!is.na(score))

# Join with presence info
presence_long <- merged_sentiment %>%
  select(imdb_id,
         ends_with("_presence")) %>%
  pivot_longer(
    cols = ends_with("_presence"),
    names_to = "theme_presence",
    values_to = "presence"
  )
```

```
# Merge and clean
sentiment_facet <- sentiment_long %>%
  mutate(theme = str_replace(theme_axis, "_conservative_progressive", ""),
         presence_col = paste0(theme, "_presence")) %>%
  left_join(presence_long, by = c("imdb_id", "presence_col" = "theme_presence")
      ) %>%
  filter(presence == "Present")

# Clean theme names
sentiment_facet$theme <- recode(sentiment_facet$theme,
                                hindu_muslim = " H i n d u Muslim  Relations",
                                gender_relations = "Gender Relations",
                                nationalism = "Nationalism",
                                lgbtq = "LGBTQIA+ Themes"
)
avg_sentiment <- sentiment_facet %>%
  group_by(release_year, theme) %>%
  summarise(avg_score = mean(score), .groups = "drop")

plot_facet <- ggplot(avg_sentiment, aes(x = release_year, y = avg_score)) +
  geom_line(color = "steelblue", size = 1) +
  geom_point(color = "steelblue", size = 2) +
  geom_text(aes(label = round(avg_score, 2)), vjust = -0.8, size = 3.3, color =
       "black") +  #        Add labels
  facet_wrap(~ theme, ncol = 2) +
  scale_x_continuous(breaks = seq(2010, 2024, 2)) +
  scale_y_continuous(limits = c(-1, 1), breaks = seq(-1, 1, 0.5)) +
  labs(
    title = "Average  C o n s e r v a t i v e Progressive  Score by Theme (2010  Present  )
       ",
    x = "Release Year",
    y = "Average Sentiment Score"
  ) +
  theme_economist_white() +
  theme(
    strip.text = element_text(face = "bold", size = 11),
    plot.title = element_text(size = 14, face = "bold", hjust = 0.5)
  )

print(plot_facet)

#Saving the plot
#NOTE: Pls change the path accordingly
ggsave("bollywood_average_score.png", plot = plot_facet,
       path = "/Users/rishabhbijani/Desktop/ra_app/plots")



#PLOT 3: Positive-Negative Axis
```

```
# Pivot longer using correct suffix
sentiment_posneg <- merged_sentiment %>%
  select(release_year, imdb_id,
         ends_with("_presence"),
         ends_with("negative_positive")) %>%
  pivot_longer(
    cols = ends_with("negative_positive"),
    names_to = "theme_axis",
    values_to = "score"
  ) %>%
  filter(!is.na(score))

# Join with presence info
presence_long <- merged_sentiment %>%
  select(imdb_id,
         ends_with("_presence")) %>%
  pivot_longer(
    cols = ends_with("_presence"),
    names_to = "theme_presence",
    values_to = "presence"
  )

# Merge and clean
sentiment_posneg_facet <- sentiment_posneg %>%
  mutate(theme = str_replace(theme_axis, "_negative_positive", ""),
         presence_col = paste0(theme, "_presence")) %>%
  left_join(presence_long, by = c("imdb_id", "presence_col" = "theme_presence")
    ) %>%
  filter(presence == "Present")

# Clean theme names
sentiment_posneg_facet$theme <- recode(sentiment_posneg_facet$theme,
                                       hindu_muslim = "HinduMuslim Relations
                                           ",
                                       gender_relations = "Gender Relations",
                                       nationalism = "Nationalism",
                                       lgbtq = "LGBTQIA+ Themes"
)

# Compute average score by year and theme
avg_posneg <- sentiment_posneg_facet %>%
  group_by(release_year, theme) %>%
  summarise(avg_score = mean(score), .groups = "drop")

# Plot
plot_posneg <- ggplot(avg_posneg, aes(x = release_year, y = avg_score)) +
  geom_line(color = "darkgreen", size = 1) +
  geom_point(color = "darkgreen", size = 2) +
  geom_text(aes(label = round(avg_score, 2)), vjust = -0.8, size = 3.3, color =
       "black") +
  facet_wrap(~ theme, ncol = 2) +
  scale_x_continuous(breaks = seq(2010, 2024, 2)) +
  scale_y_continuous(limits = c(-1, 1), breaks = seq(-1, 1, 0.5)) +
  labs(
    title = "Average P o s i t i v eNegative  Score by Theme (2010  Present  )",
    x = "Release Year",
    y = "Average Sentiment Score"
  ) +
  theme_economist_white() +
  theme(
    strip.text = element_text(face = "bold", size = 11),
    plot.title = element_text(size = 14, face = "bold", hjust = 0.5)
  )

# Show and save
print(plot_posneg)

ggsave("bollywood_average_score_positive_negative.png", plot = plot_posneg,
       path = "/Users/rishabhbijani/Desktop/ra_app/plots")


#PLOT4: Inclusionary-Exclusionary



# Pivot longer using correct suffix
sentiment_exclincl <- merged_sentiment %>%
  select(release_year, imdb_id,
         ends_with("_presence"),
         ends_with("_exclusionary_inclusionary")) %>%
  pivot_longer(
    cols = ends_with("_exclusionary_inclusionary"),
    names_to = "theme_axis",
    values_to = "score"
  ) %>%
  filter(!is.na(score))

# Merge with presence data
sentiment_exclincl_facet <- sentiment_exclincl %>%
  mutate(theme = str_replace(theme_axis, "_exclusionary_inclusionary", ""),
         presence_col = paste0(theme, "_presence")) %>%
  left_join(presence_long, by = c("imdb_id", "presence_col" = "theme_presence")
     ) %>%
  filter(presence == "Present")

# Clean theme names
sentiment_exclincl_facet$theme <- recode(sentiment_exclincl_facet$theme,
                                          hindu_muslim = " H i n d u Muslim
                                              Relations",
                                          gender_relations = "Gender Relations",
                                         nationalism = "Nationalism",
                                         lgbtq = "LGBTQIA+ Themes"
)

# Compute average score by year and theme
avg_exclincl <- sentiment_exclincl_facet %>%
  group_by(release_year, theme) %>%
  summarise(avg_score = mean(score), .groups = "drop")

# Plot
plot_exclincl <- ggplot(avg_exclincl, aes(x = release_year, y = avg_score)) +
  geom_line(color = "purple", size = 1) +
  geom_point(color = "purple", size = 2) +
  geom_text(aes(label = round(avg_score, 2)), vjust = -0.8, size = 3.3, color =
      "black") +
  facet_wrap(˜ theme, ncol = 2) +
  scale_x_continuous(breaks = seq(2010, 2024, 2)) +
  scale_y_continuous(limits = c(-1, 1), breaks = seq(-1, 1, 0.5)) +
  labs(
    title = "Average ExclusionaryInclusive Score by Theme (2010 Present )",
    x = "Release Year",
    y = "Average Sentiment Score"
  ) +
  theme_economist_white() +
  theme(
    strip.text = element_text(face = "bold", size = 11),
    plot.title = element_text(size = 14, face = "bold", hjust = 0.5)
  )

# Show and save
print(plot_exclincl)

ggsave("bollywood_average_score_exclusionary_inclusive.png", plot = plot_
    exclincl,
       path = "/Users/rishabhbijani/Desktop/ra_app/plots")




#PLOT 5 : Stacked Area Chart

#Counting number of movies per year
year_totals <- theme_freq %>%
  group_by(release_year) %>%
  summarise(total_movies = sum(count), .groups = "drop")

#Plotting
plot_area <- ggplot(theme_freq, aes(x = release_year, y = count, fill = theme))
     +
  geom_area(alpha = 0.8, position = "stack") +
  geom_text(
    data = year_totals,
    aes(x = release_year, y = total_movies + 0.3, label = total_movies),
    inherit.aes = FALSE,
    size = 3.5,
    vjust = 0
  ) +
  labs(
    title = "Stacked Area Chart of Theme Frequency (2010  Present  )",
    x = "Release Year", y = "Number of Movies", fill = "Theme"
  ) +
  theme_economist() +
  scale_y_continuous(breaks = pretty_breaks(n = 8)) +
  scale_x_continuous(breaks = seq(2010, 2024, 1)) +
  theme(
    legend.position = "bottom",
    legend.title = element_text(size = 10),
    legend.text = element_text(size = 9)
  ) +
  guides(fill = guide_legend(nrow = 2, byrow = TRUE))

# Display plot
print(plot_area)

#Saving Area Plot
#NOTE: Pls change the path accordingly
ggsave("bollywood_stacked_area_plot.png", plot = plot_area,
       path = "/Users/rishabhbijani/Desktop/ra_app/plots")
```

Chapter 3

Plots

3.1 Main Plot

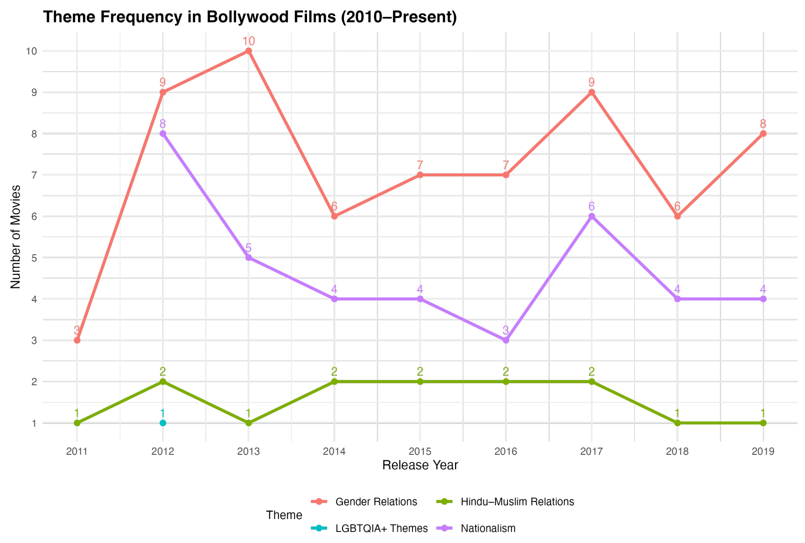
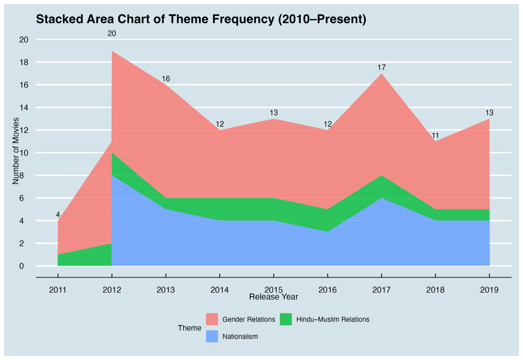
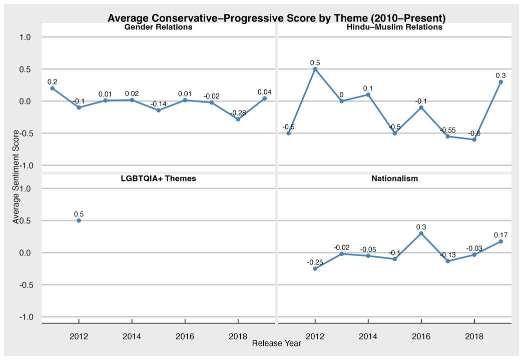


Figure 3.1: Theme Frequency in Bollywood Films (2010–Present)

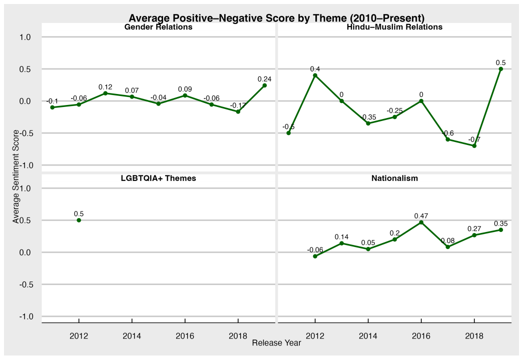
3.2 Additional Plots



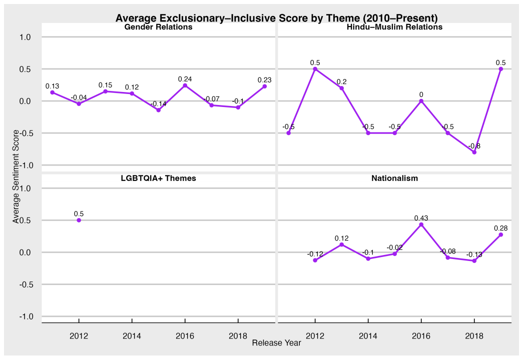
(a) Theme Frequency in Bollywood Films (2010–Present)



(b) Avg Sentiment (Conservative–Progressive)



(c) Avg Sentiment (Positive–Negative)



(d) Avg Sentiment (Exclusionary–Inclusive)

Figure 3.2: Thematic Sentiment Trends in Bollywood Films (2010–2024)

Chapter 4

Supplemtary Ideas

4.1 Thematic Keywords

1. Description
   TMDb contains thematic keywords for every movie which describe the major themes in the movie.

2. Data Collection Approach
   The TMDb API contains the following endpoint which can be used to scrape keywords for a move using its IMDb/TMDb ID -https://api.themoviedb.org/3/movie/tmdb_id/keywords

3. Reasoning
   (a) Keywords can help in more pointed, focussed prompted of LLMs for thematic analysis.
   (b) Keywords can also be used for confirming the theme inferred by the LLM by analysing the subtitles and description as a form of robustness check.

4.2 Actors

1. Description
   The Kaggle dataset contains the information on the actors in a particular film.

2. Data Collection Approach
   The file archive/2010-2019/Bollywood_text_2010-2019.csv and also TMDb API can be used to extract the names of the actors in a particular film. The file archive/2010-2019/Bollywood_text_2010-2019.csv has the names of actors in a string vector which can be matched with the file archive/1950-2019/Bollywood_crew_data_1950-2019.csv which identifies each actor with a unique crew_id.

4.3 Writers

1. Description
   The Kaggle dataset contains information on writers as well.

2. Data Collection Approach
   The file archive/1950-2019/Bollywood_crew_1950-2019.csv lists unique crew_id of writers for each film. The file archive/1950-2019/bollywood_writers_data_1950-2019.csv lists out all writers with their unique crew_id.

3. Reasoning for Points 2 and 3
   Identifying the actors and writers with each film and then, correlating it with the themes and sentiment-scores can help us in profiling of actors and writers.

Chapter 5

Appendix

5.1 Prompts

5.1.1 Gender Inference Prompt

```
prompt = f"""
Based on the following name and biography of a film director associated with
    Indian cinema (especially Bollywood), what is the most likely gender of
    this person?

Name: {name}

Biography: {bio}

Respond in JSON format with two fields:

{
  "gender": "male" or "female",
  "confidence": a number between 0.0 and 1.0 indicating how confident you are
      in this classification
}
"""
```

5.1.2 Sentiment Analysis Prompt

```
You are a social scientist and analyst who is tasked with analysing cultural
    and political themes in the subtitles and descriptions of Bollywood (i.e.
    Indian film industry) films since 2010.

Your task is to --

1. Detect whether each of the following themes appears --

```

```
1. Hindu-Muslim Relations

2. Gender Relations

3. Nationalism

4. LGBTQIA+ Themes

2. For each theme that is present, assess it on the following axes --

1. Exclusionary v/s Inclusionary

2. Negative v/s Positive

3. Conservative v/s Progressive

3. For each axis, assign a score on a continuous scale from **-1 to +1**:

- A score of **-1** represents the most Exclusionary, Negative, or Conservative
     representation possible.

- A score of **+1** represents the most Inclusionary, Positive, or Progressive
    representation possible.

- A score of **0** indicates a neutral or balanced portrayal.

Thematic definitions of the themes are as follows --

1. Hindu-Muslim Relations

In India, Hindus form about 79% of the population whereas Muslims form about
    14%. Hindu-Muslim relations are characterised by periods of both, amity,
    cooperation and syncretism and also with strife and violence.

2. Gender Relations

UNICEF defines gender relations as follows: A specific sub-set of social
    relations uniting men and women as social groups in a particular community.
     Gender relations intersect with all other influences on social relations
    -- age, ethnicity, race, religion -- to determine the position and identity
     of people in a social group. Since gender relations are a social construct
    , they can be changed.

3. Nationalism

Oxford Reference defines nationalism as follows: A political ideology and
    associated movement intended to realize or further the aims of a nation,
    most notably for independent self-government in a defined territory. In a
    broader sense, nationalism also refers to sentiments of attachment to or
    solidarity with a national identity or purpose.
```

```
4. LGBTQIA+ Themes

Cambridge Dictionary defines LQBTBQIA+ themes as follows: abbreviation for
    lesbian, gay, bisexual, transgender, queer (or questioning), intersex, and
    asexual (or ally): relating to or characteristic of people whose sexual
    orientation is not heterosexual (= sexually or romantically attracted to
    women if you are a man, and men if you are a woman) or whose gender
    identity is not cisgender (= having a gender that matches the physical body
     you were born with).

Definitions of the axes are as follows:

1. Exclusionary v/s Inclusionary

Cambridge Dictionary defines exclusionary to be: limited to only one group or
    particular groups of people, in a way that is unfair; resulting in a person
     or thing not being included in something.

Cambridge Dictionary defines inclusion to be: the act of including someone or
    something as part of a group, list, etc., or a person or thing that is
    included; the idea that everyone should be able to use the same facilities,
     take part in the same activities, and enjoy the same experiences,
    including people who have a disability or other disadvantage.

2. Negative v/s Positive

Cambridge Dictionary defines negative to be: not expecting good things, or
    likely to consider only the bad side of a situation; bad or harmful.

Cambridge Dictionary defines positive to be: full of hope and confidence, or
    giving cause for hope and confidence.

3. Conservative v/s Progressive

Cambridge Dictionary defines conservative to be: not usually liking or trusting
     change, especially sudden change.

Cambridge Dictionary defines progressive to be: Progressive ideas or systems
    are new and modern, encouraging change in society or in the way that things
     are done.

---

Please analyze the following **Movie Description and Subtitle Transcript**
    carefully and respond using this structure:

For each of the 4 themes, do the following:

- Presence: Present / Not Present / Ambiguous
- If Present, assign a score between -1 and +1 on each of the following axes:

- Exclusionary--Inclusionary

- Negative--Positive

- Conservative--Progressive

---

Movie Plot Description:

{description}

---

Subtitle Transcript:

{subtitles}
```