BECHDEL TEST

OPERATIONAL GUIDE

Alice Brunazzi, Alessandro della Beffa, Daniele Lepre

Keywords: Bechdel Test, Gender representation, Scripts, Names Dataset, Automated Analysis, SQLite Project.

Index

Introduction

This work presents a comprehensive study on gender representation in films, through the application of the Bechdel test criteria, leveraging film scripts and cast information. This paper provides information about the creation and the usage of an SQL database ideated to answer the research questions.

The final objective of this project is to develop a database containing scripts and cast details, in order to automate the evaluation of the Bechdel test criteria. In order to pass, a film must contain at least two women in the cats, who speak to each other in a scene (without a man present). The conversation must not be about a man.

CODE GUIDE

Requirements

The goal of the code is to download movie scripts, divide the scripts into scenes, analyze the cast of the movies, and populate a SQLite database with this information.

The Python packages required are:

- Requests: used to download web pages content, i.e. movie scripts and cast information
- BeautifulSoup4: allowed easy extraction and navigation of data from the downloaded web pages, facilitating the analysis of the movies' ccontent.
- Sqlalchemy. Used to manage and interact with the SQLite database, simplifying the handling of movie and cast data.
- Spacy: used to identify and analyze character names in scripts, essential for gender classification
- Names-dataset: used to classify movie characters as female or males, just by using their birth names.
- Re: used to search and/or manipulate strings in movie scripts, useful for the extraction of names in dialogs.
- String: Used to manipulate and analyze texts in scripts, such as cleaning dialogues

It is important to install the packages with the !pip install command. It's important to install Name-Dataset library whenever you want to run the code.

Database Creation

First, the code creates a SQLite database named `Bechdel_test.db` and three tables: `MOVIE`, `SCRIPT`, and `CAST`.

- `MOVIE` formed by the columns *film_id, titolo, anno*
- `SCRIPT` formed by the columns *film_id, scene_id, place, scena, personaggi, count_female, count_male, nom, count_male_nom*
- `CAST` formed by the columns *film_id, nome, ruolo, gender*

```
# Creazione del database
engine = create_engine('sqlite:///Bechdel_test.db')

# Funzione per creare le tabelle
def create_tables():
    with engine.connect() as conn:
        conn.execute(text('''
        CREATE TABLE IF NOT EXISTS MOVIE (
            film_id INTEGER PRIMARY KEY,
            titolo TEXT,
            anno INTEGER
        )
        '''))

        conn.execute(text('''
        CREATE TABLE IF NOT EXISTS SCRIPT (
            film_id INTEGER,
            scene_id INTEGER,
            place TEXT,
            scena TEXT,
            personaggi TEXT,
```

```
            count_female INTEGER,
            count_male INTEGER,
            nom TEXT,
            count_male_nom INTEGER,
            PRIMARY KEY (film_id, scene_id)
        )
        '''))

        conn.execute(text('''
        CREATE TABLE IF NOT EXISTS CAST (
            film_id INTEGER,
            nome TEXT,
            ruolo TEXT,
            gender TEXT,
            PRIMARY KEY (film_id, nome)
        )
        '''))
create_tables()
```

Functions used in the code

Download and split movie script from IMSDb

```
def scarica_e_dividi_script_da_imsdb(titolo_film):
    url = f"https://www.imsdb.com/scripts/{titolo_film.replace(' ', '-')}.html"
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        script = soup.find('td', {'class': 'scrtext'}).text
        scene_divise = dividere_script_con_int_ext(script)
        return scene_divise
    else:
        print(f"Errore nel scaricare lo script da IMSDb per il film
{titolo_film}")
        return None
```

This code use webscraping techniques to import the required scripts, and then splits it using the scene_divise function, if you want to import the entire script (without division), the line `scene_divise = dividere_script_con_int_ext(script)` must to be deleted.

If instead you want to change the way the script is divided, go to section: *script division*

Cast download from IMDb

```
def scarica_cast_da_imdb(imdb_id):
    url = f"https://www.imdb.com/title/{imdb_id}/fullcredits"
    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        cast_list = []
        roles_seen = set()

        for row in soup.select('.cast_list tr')[1:]:  # Ignore the header row
            columns = row.find_all('td')
            if len(columns) >= 4:
                actor_name = columns[1].get_text(strip=True)
                character_name = columns[3].get_text(strip=True).split('
')[0]  # Take only the first word of the role

                # Remove all punctuation from the character name
```

```
                character_name = character_name.translate(str.maketrans('', '',
string.punctuation))

                # Remove trailing 's if present
                character_name = re.sub(r"'s$", "", character_name)

                # Check if the character name has already been seen
                if character_name not in roles_seen:
                    roles_seen.add(character_name)

                    # Exclude roles that are "A", "The", contain numbers, or
contain "uncredited"
                    if (character_name.lower() in {"a", "the"} or
any(char.isdigit() for char in character_name)
                            or "uncredited" in character_name.lower()):
                        continue  # Skip this row

                    cast_list.append({'name': actor_name, 'role':
character_name})

        return cast_list
    else:
        print(f"Error: Unable to fetch data from IMDB for {imdb_id}")
        return []
```

During the cast import phase, it was decided to:

- Import only the first name of the characters portrayed
- Remove punctuation from the names
- Remove ‘s from the names
- Not import characters that have already been imported, so if multiple actors played the same character, only the first one is considered
- Remove characters that start with “THE” or “A” or that contain “uncredited”

If it is desired to change these settings, simply remove or modify the desired sections from the above code.

Script division

```
def dividere_script_con_int_ext(script):
    nlp = spacy.load('en_core_web_sm')

    int_ext_positions = [match.start() for match in
re.finditer(r'\bINT\b|\bEXT\b', script)]
    # Dividi il testo dello script in base alle posizioni trovate
    scene_divise = []
    for start, end in zip([0] + int_ext_positions, int_ext_positions +
[len(script)]):
        scena = script[start:end].strip()
        if scena:
            place = scena.split('\n', 1)[0].strip() if '\n' in scena else
scena.strip()

            # Estrai solo INT o EXT dalla stringa di place
            match = re.match(r'\b(INT|EXT)\b', place)
            if match:
                place = match.group(0)

            # Controlla se la scena inizia con INT./, EXT./, INT/, o EXT/
            if re.match(r'\b(INT\./|EXT\./|INT/|EXT/)', scena):
```

```
                continue  # Salta questa scena
            scena_pulita = re.sub(r'[<>*\d]', '', scena)
            # Estrai i personaggi dalla scena utilizzando spaCy
            doc = nlp(scena)
            personaggi = [ent.text for ent in doc.ents if ent.label_ ==
"PERSON"]
            scene_divise.append({'place': place, 'scena': scena_pulita,
'personaggi': ' '.join(personaggi)})
    return scene_divise
```

The function divides the script into scenes based on the occurrences of INT (interior) and EXT (exterior). To achieve this, regular expressions are used to find the positions where the keywords "INT" and "EXT" appear. Each segment of text between the found positions of INT and EXT is considered a scene. The first line, containing INT and EXT, is used to populate the 'place' column, which is then cleaned to obtain only INT or EXT.

If a scene contains only INT./, EXT./, INT/, or EXT/, it is ignored. Additionally, the scene is cleaned by removing undesired characters (<, >, *, \d). The function also extracts characters from the scene using spaCy.

Insert data

```
def insert_data(data, table_name):
    with engine.begin() as conn:
        if table_name == "SCRIPT":
            conn.execute(text('''
            INSERT INTO SCRIPT (film_id, scene_id, place, scena, personaggi,
count_female, count_male, nom, count_male_nom)
            VALUES (:film_id, :scene_id, :place, :scena, :personaggi,
:count_female, :count_male, :nom, :count_male_nom)
            '''), data)
        elif table_name == "CAST":
            # Check if the entry already exists before inserting
            existing_entry = conn.execute(text('''
                SELECT 1
                FROM CAST
                WHERE film_id = :film_id AND nome = :nome
            '''), data).fetchone()
            if not existing_entry:  # Insert only if it doesn't exist
                conn.execute(text('''
                INSERT INTO CAST (film_id, nome, ruolo, gender)
                VALUES (:film_id, :nome, :ruolo, :gender)
                '''), data)
        elif table_name == "MOVIE":
            conn.execute(text('''
            INSERT INTO MOVIE (film_id, titolo, anno)
            VALUES (:film_id, :titolo, :anno)
            '''), data)
```

Gender determination

```
def gender_name(name):
    # Inizializza il dataset dei nomi
    nd = NameDataset()
    # Cerca il nome nel dataset
    name_info = nd.search(name)
    if name_info and 'first_name' in name_info and name_info['first_name']:
        # Ottieni le informazioni sul genere
        gender_info = name_info['first_name'].get('gender')
        if gender_info:
```

```
            # Estrai il genere più probabile
            most_probable_gender = max(gender_info, key=gender_info.get)
            return most_probable_gender
    # Se non ci sono informazioni sul genere, restituisci None
    return None
```

In this way, the most probable gender was determined. This process allows to achieve an high level of accuracy in gender determination, eliminating the problems that other method had (e.g. Web scraping the Wikipedia page didn't always provide an answer)
This process also overcame the limitations of the Gender API, that limited the number of queries to 100.

The unknown values (3% of the whole `CAST` table) will be eliminated in a following step, precisely in the main function

Extraction of characters in the scene

```
def estrai_personaggi_comuni(scena, ruoli, parole_chiave, ):
    parole_originali = {}

    personaggi_scena = []
    for word in scena.split():
        parola_pulita = word.strip(string.punctuation)
        if parola_pulita.isupper():  # Controlla se la parola è interamente maiuscola
            personaggi_scena.append(parola_pulita.lower())
            parole_originali[parola_pulita.lower()] = parola_pulita


    ruoli_set = set(ruolo.lower() for ruolo in ruoli)
    parole_chiave_set = set(parola.lower() for parola in parole_chiave)


    personaggi_comuni_set = set()
    for personaggio in personaggi_scena:
        if personaggio in ruoli_set.union(parole_chiave_set):
            personaggi_comuni_set.add(parole_originali[personaggio])

    # Restituisce le parole nel loro formato originale, mantenendo l'ordine originale
    personaggi_comuni = [personaggio for personaggio in
parole_originali.values() if personaggio in personaggi_comuni_set]

    return ' '.join(personaggi_comuni)
```

This function is designed to extract and return the "speaking" characters from a scene based on specific roles and provided keywords. Characters that appear in the scene will be extracted only if they are written entirely in uppercase letters. To achieve this, the function removes punctuation, converts words to lowercase, and identifies characters entirely in uppercase. It uses sets to compare roles and keywords, returning the found characters in their original format, maintaining the original order of discovery in the scene. The output is a string with the names of the characters (without repetition) separated by spaces.

Nominated character extraction

```
def estrai_ruoli_comuni(scena, ruoli_cast, parole_chiave_maschi):
    scena_pulita = re.sub(r'[{}]'.format(re.escape(string.punctuation)), '',
scena)
    parole_iniziali_maiuscole = re.findall(r'\b[A-Z][a-z]*\.?\b', scena_pulita)

    ruoli_cast_set = set(ruolo.lower() for ruolo in ruoli_cast)
```

```
    parole_chiave_set = set(parola.lower() for parola in parole_chiave_maschi)

    # Trova le parole comuni tra le parole iniziali maiuscole e i ruoli_cast
    ruoli_comuni_set = set()
    for parola in parole_iniziali_maiuscole:
        parola_minuscola = parola.lower()
        if parola_minuscola in ruoli_cast_set:
            ruoli_comuni_set.add(parola)
    # Trova le parole chiave direttamente nella scena
    parole_chiave_trovate = set()
    parole_nella_scena = scena_pulita.lower().split()
    for parola in parole_nella_scena:
        if parola in parole_chiave_set:
            parole_chiave_trovate.add(parola)

   # Restituisce i ruoli comuni e le parole chiave trovate come stringa separata da spazi
    return ' '.join(ruoli_comuni_set) + ' ' + ' '.join(parole_chiave_trovate)
```

This function analyzes a scene to extract common male roles and keywords. It removes punctuation from the scene, identifies words with capitalized initials or abbreviations, and uses sets for efficient searching. It finds matches between cast roles and initial words identified. It also finds keywords directly in the scene, returning a string containing the found roles and keywords, separated by spaces.

Character count

```
def update_script_counts(parole_chiave, parole_chiave_maschi, film_id):

    with engine.begin() as conn:
        # Seleziona i ruoli di tutti i personaggi dalla tabella CAST per il film
specificato
        cast_roles = conn.execute(text('''
            SELECT ruolo, LOWER(gender)
            FROM CAST
            WHERE film_id = :film_id
        '''), {'film_id': film_id}).fetchall()

        cast_roles_map = {'male': set(), 'female': set()}
        for role, gender in cast_roles:
            if gender == 'male':
                cast_roles_map['male'].add(role.lower())
            elif gender == 'female':
                cast_roles_map['female'].add(role.lower())

        # Converti le parole chiave in minuscolo
        parole_chiave_set = set(parola.lower() for parola in parole_chiave)
        parole_chiave_maschi_set = set(parola.lower() for parola in
parole_chiave_maschi)

        # Rimuovi le parole chiave dalle liste dei ruoli del cast
        cast_roles_map['male'] -= parole_chiave_set
        cast_roles_map['male'] -= parole_chiave_maschi_set
        cast_roles_map['female'] -= parole_chiave_set
        cast_roles_map['female'] -= parole_chiave_maschi_set

        scripts = conn.execute(text('''
            SELECT film_id, scene_id, personaggi
            FROM SCRIPT
            WHERE film_id = :film_id
        '''), {'film_id': film_id}).fetchall()

        # Calcola il conteggio degli elementi comuni e aggiorna la tabella
SCRIPT
```

```
        for script in scripts:
            personaggi = script[2].lower().split()
            common_count = sum(1 for personaggio in personaggi if personaggio in
cast_roles_map['female'])
            keyword_count = sum(1 for personaggio in personaggi if personaggio
in parole_chiave_set)
            male_count = sum(1 for personaggio in personaggi if personaggio in
cast_roles_map['male'])
            keyword_male_count = sum(1 for personaggio in personaggi if
personaggio in parole_chiave_maschi_set)
            total_count = common_count + keyword_count
            total_male_count = male_count + keyword_male_count
            conn.execute(text('''
                UPDATE SCRIPT
                SET count_female = :count_female,
                    count_male = :count_male
                WHERE film_id = :film_id AND scene_id = :scene_id
            '''), {'count_female': total_count, 'count_male': total_male_count,
'film_id': script[0], 'scene_id': script[1]})
```

This function updates the counts in the database for female and male characters, as well as for keywords, based on the specified movie.

1. *Cast roles selection*: Retrieve the character roles from the `CAST` database for the specified movie and map them into a `cast_roles_map` dictionary divided by gender (male and female), converting the roles to lowercase.
2. *Keywords preparation*: Convert keywords and male keywords into sets of lowercase strings for efficient matching.
3. *Update counts*: For each scene in the `SCRIPT` database related to the specified movie, the function calculates and updates the counts of female and male characters. This includes the count of common characters (female), count of present keywords, count of male characters, and count of male keywords.

The resulting counts are then updated in the `SCRIPT` table.

Named character count

```
def update_script_male_nom_counts(parole_chiave_maschi):

    parole_chiave_maschi_set = set(parola.lower() for parola in
parole_chiave_maschi)

    with engine.begin() as conn:
        scripts = conn.execute(text('''
            SELECT film_id, scene_id, nom
            FROM SCRIPT
        ''')).fetchall()

        for script in scripts:
            count_male_nom = 0
            if script[2]:
                nom_names = script[2].split()
                for name in nom_names:
                # Verifica se il nome è presente tra le parole chiave maschili
                    if name.lower() in parole_chiave_maschi_set:
                        count_male_nom += 1
                    else:
```

```
                        result = conn.execute(text('''
                            SELECT 1
                            FROM CAST
                            WHERE film_id = :film_id
                                AND LOWER(gender) = 'male'
                                AND LOWER(ruolo) = :ruolo
                        '''), {'film_id': script[0], 'ruolo':
name.lower()}).fetchone()
                        if result:
                            count_male_nom += 1

            # Aggiorna la colonna count_male_nom nella riga corrente
            conn.execute(text('''
                UPDATE SCRIPT
                SET count_male_nom = :count_male_nom
                WHERE film_id = :film_id AND scene_id = :scene_id
            '''), {'count_male_nom': count_male_nom, 'film_id': script[0],
'scene_id': script[1]})
```

This function updates the counts of male names in the scenes of scripts in the database.

1. *Male keywords preparation*: Convert the list of male keywords into a set of lowercase strings for efficient matching.
2. *Selection of Rows from the SCRIPT Table*: Retrieve all rows from the `SCRIPT` table containing film_id, scene_id, and nom.
3. *Counting Male Names*: For each retrieved row:
   - Count the occurrences of male names present in nom (if not empty), comparing them with the male keywords.
   - If a name is not among the keywords, search in the `CAST` table to verify if it is a male role.
4. *Updating the Counts in the Database*: Update the `count_male_nom` column for each row in the `SCRIPT` table with the count of male names found.

Main Function

```
def main():

    films = [
{"titolo": "Rocky", "imdb_id": "tt0075148", "anno": 1976},
{"titolo": "Blade Runner","imdb_id": "tt0083658", "anno":1982 },
{"titolo": "Dead Poets Society", "imdb_id": "tt0097165", "anno": 1989},
{"titolo": "Fargo", "imdb_id": "tt0116282", "anno": 1996},
{"titolo": "Good Will Hunting", "imdb_id": "tt0119217", "anno": 1997},
{"titolo": "Truman Show, The", "imdb_id": "tt0120382", "anno": 1998},
{"titolo": "American Beauty", "imdb_id": "tt0169547", "anno": 1999},
{"titolo": "Gladiator", "imdb_id": "tt0172495", "anno": 2000},
{"titolo": "Memento", "imdb_id": "tt0209144", "anno": 2000},
{"titolo": "American Psycho", "imdb_id": "tt0144084", "anno": 2000},
{"titolo": "Big Fish", "imdb_id": "tt0319061", "anno": 2003},
{"titolo": "Eternal Sunshine of the Spotless Mind", "imdb_id": "tt0338013", "anno":
2004},
{"titolo": "Devil Wears Prada, The", "imdb_id": "tt0458352", "anno": 2006},
{"titolo": "No Country for Old Men", "imdb_id": "tt0477348", "anno": 2007},
{"titolo": "Anna Karenina", "imdb_id": "tt1781769", "anno": 2012},
{"titolo": "12 Years a Slave", "imdb_id": "tt2024544", "anno": 2013},
{"titolo": "Great Gatsby, The", "imdb_id": "tt1343092", "anno": 2013},
{"titolo": "Interstellar", "imdb_id": "tt0816692", "anno": 2014},
{"titolo": "American Sniper", "imdb_id": "tt2179136", "anno": 2014},
{"titolo": "La La Land", "imdb_id": "tt3783958", "anno": 2016},

    ]
```

```
    parole_chiave = ["LADY", "WOMAN", "MOM", "Mother","AUNT", "WIFE", "GIRLFRIEND",
"MOTHER","GRANDMOTHER", "GRANDMA", "FEMALE", "GIRL", "ACTRESS", "MISS", "MS.",
"MRS","ASSISTANT", "NURSE","princess"
    ]
    parole_chiave_maschi = ["Sir", "Man", "Dad", "Uncle", "Husband", "Boy",
"Father","Grandad", "Grandfather", "Male","Prince", "Mr"
    ]

    for film_id, film in enumerate(films, start=1):
        titolo_film = film["titolo"]
        imdb_id = film["imdb_id"]
        anno = film["anno"]

        cast_list = scarica_cast_da_imdb(imdb_id)

        with engine.begin() as conn:
            for actor in cast_list:
                first_name = actor['name'].split(' ', 1)[0]
                most_probable_gender = gender_name(first_name)
                if most_probable_gender is None:
                    continue
                print(f"Determinato gender per {first_name} ({actor['name']}):{most_probable_gender}")
                cast_data = {
                    'film_id': film_id,
                    'nome': actor['name'],
                    'ruolo': pulisci_ruolo(actor['role']),
                    'gender': most_probable_gender,
                }
                insert_data(cast_data, 'CAST')

        # Scarica e divide lo script da IMSDb
        scene_divise = scarica_e_dividi_script_da_imsdb(titolo_film)

        # Estrai i ruoli da CAST per i personaggi delle scene
        ruoli_cast = [pulisci_ruolo(actor['role']) for actor in cast_list]

        if scene_divise:
            ruoli_cast.clear()
ruoli_cast.extend(pulisci_ruolo(actor['role']) for actor in cast_list)
            print(ruoli_cast)

            for scene_id, scene in enumerate(scene_divise, start=1):
                place = scene['place']
                cleaned_scene = pulisci_scena(scene['scena'])
                ruoli_comuni = estrai_ruoli_comuni(scene['scena'], ruoli_cast,
parole_chiave_maschi)
                personaggi_comuni = estrai_personaggi_comuni(scene['scena'],
ruoli_cast, parole_chiave)

                count_female = sum(1 for personaggio in
personaggi_comuni.split() if personaggio in parole_chiave)

                count_male_nom = sum(1 for nome in ruoli_comuni.split() if nome
in parole_chiave_maschi)

                scene_data = {
                    'film_id': film_id,
                    'scene_id': scene_id,
                    'place': place,
                    'scena': cleaned_scene,
                    'personaggi': personaggi_comuni,
                    'count_female': count_female,
```

```
                'count_male': 0,  # Calcolato in update_script_counts
                'nom': ruoli_comuni,
                'count_male_nom': count_male_nom,
            }
            insert_data(scene_data, 'SCRIPT')

        print(f"Scene del film {titolo_film} inserite nel database.")

    # Inserimento dati nella tabella MOVIE
    movie_data = {'film_id': film_id, 'titolo': titolo_film, 'anno': anno}
    insert_data(movie_data, 'MOVIE')

    # Aggiorna la colonna count_female nella tabella SCRIPT
    update_script_counts(parole_chiave, parole_chiave_maschi, film_id)
    update_script_male_nom_counts(parole_chiave_maschi)
```

The main function is the core of an automated process for downloading, analyzing, and storing data related to movies, including the cast and script.

1. *Movie List*: There is a list of movies with titles, IMDb IDs, and years (currently commented out).
2. *Keywords*: Lists of female and male keywords are defined. If it is desired to modify the keywords, simply add them by inserting a comma and the desired word into the appropriate dictionary between the quotation marks.
3. *Processing each movie*:
   - *Cast download*: Use the `scarica_cast_da_imdb` function to obtain the movie's cast from IMDb.
   - *Database insertion*: Insert the cast data into the database, determining the gender of each actor.
   - *Script Download*: Use `scarica_e_dividi_script_da_imsdb` to obtain and divide the script into scenes.
   - *Scene Processing*:
     - Clean and analyze the scenes, extracting roles and common characters.
     - Calculate the counts of female characters and male names in the scenes.
     - Insert the scene data into the database.
   - *Movie data insertion*: Insert the general movie data into the `MOVIE` table.
   - *Counts update*: Update the counts of female characters and male names in the scenes using the `update_script_counts` and `update_script_male_nom_counts` functions.

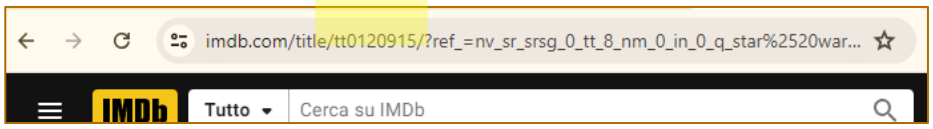
Adding films:

To add a new movie follow the steps below:

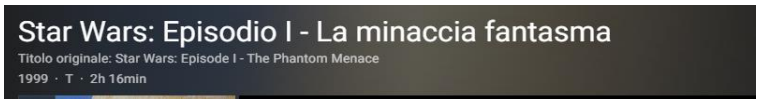
1. use the IMSDB website to check if the script of the film is available. To check it go write the title in the apposite bar. Then click on the link of the movie and verify if the script is clicking on Read “title of the film” Script.

2. if the script is available, search the film title in the IMDB website. This step is essential to provide the unique code, that can be found as follows:

   Let's say that we want to integrate "Star Wars: Episode I: The Phantom Menace", after writing the name in the search bar in the website, it is crucial to look up the following code:



   After that, the year of release and the original title should be written down as well:



3. Write it into the listo of films, available right under the definition of the main function, in the following way:

```
"titolo": "Star Wars: Episode I - The Phantom Menace", "imdb_id": "tt0120915", "anno": 1999}
```

4. Run the code, only if the lists of keywords are satisfying for the analysis.
5. Once the database is available, use the queries provided in the annected *report* to perform the three levels of the Bechdel Test.

DATABASE GUIDE

Requirements

- SQLite: used to store movie data, scripts and cast information, enabling efficient data management for automated analysis

How it looks

Once the database is downloaded, it is possible to open the three tables, which will appear as shown in the figures:

CAST

film_id	nome	ruolo	gender
Filtro	Filtro	Filtro	Filtro
1	Sylvester Stallone	Rocky	Male
1	Talia Shire	Adrian	Female
1	Burt Young	Paulie	Male
1	Carl Weathers	Apollo	Male
1	Burgess Meredith	Mickey	Male
1	Thayer David	Jergens	Male
1	Joe Spinell	Gazzo	Male
1	Jimmy Gambina	Mike	Male
1	Bill Baldwin	Fight	Male
1	Al Silvani	Cut	Male

MOVIE

film_id	titolo	anno
Filtro	Filtro	Filtro
1	Rocky	1976
2	Blade Runner	1982
3	Dead Poets Society	1989
4	Good Will Hunting	1997
5	Truman Show, The	1998
6	American Beauty	1999
7	Gladiator	2000
8	Memento	2000
9	American Psycho	2000
10	Big Fish	2003

SCRIPT

film_id	scene_id	place	scena	personaggi	count_female	count_male	nom	count_male_nom
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1	1	Rocky	"ROCKY"...	ROCKY	0	1	Rocky	1
1	2	INT	SUPERIMPOSE OVER ACTION... ...	FIGHT CLUB ROCKY FIGHTER	1	3	Fighter Rocky man	2
1	3	INT	Rocky has nearly completed dressing ...	FIGHTER RADIO	1	1	Rocky man	2
1	4	INT	Rocky is on the trolley heading to ...	WOMAN ROCKY	1	1	Rocky	1
1	5	EXT	Rocky exits the trolley and walks do...		0	0	Rocky man	2
1	6	EXT	A short while later, Rocky approache...		0	0	Rocky	1
1	7	INT	The narrow hallway is painted olive ...		0	0		0
1	8	INT	Rocky enters. The one-room ...	ROCKY	0	1	Rocky	1
1	9	EXT	We SEE the jagged skyline, highlight...		0	0		0
1	10	EXT	Rocky is walking along the ...	ROCKY	0	1	Rocky man	2
1	11	INT	The man enters the ship's hole and ...	FATS ROCKY	0	2	Gazzo Rocky man mr	4
1	12	EXT	Later that morning Rocky passes ...	ADRIAN	1	0	Adrian Rocky	1
1	13	INT	ADRIAN...	ADRIAN ROCKY OWNER	2	1	Owner Adrian Rocky	1
1	14	EXT	Gazzo picks up Rocky		0	0	Gazzo Rocky	2

To write queries on these, follow these instructions to address the research questions.

First Level:

The first level of the Bechdel test requires the presence of two named women in the cast. After the creation of the database, this level is easily satisfied by querying the CAST table

The query is

```
SELECT m.titolo, 'PASS' AS at_least_2_woman
FROM MOVIE m
JOIN (
   SELECT film_id
   FROM CAST
   WHERE LOWER(gender) = 'female'
   GROUP BY film_id
   HAVING COUNT(*) >= 2
) c ON m.film_id = c.film_id;
```

	titolo	at_least_2_woman
1	Rocky	PASS
2	Blade Runner	PASS
3	Dead Poets ...	PASS
4	Good Will ...	PASS
5	Truman Show...	PASS
6	American ...	PASS
7	Gladiator	PASS
8	Memento	PASS
9	American ...	PASS
10	Big Fish	PASS

Second Level:

The second level requires that two women in the cast entertain a conversation without a man present, in a whole scene.

```
SELECT m.film_id, m.titolo, m.anno

FROM MOVIE m

JOIN SCRIPT s ON m.film_id = s.film_id

WHERE s.count_female >= 2 and s.count_male=0;
```

film_id	titolo	anno
6	American Beauty	1999
11	Eternal Sunshine of the Spotless Mind	2004
12	Devil Wears Prada, The	2006
13	No Country for Old Men	2007
15	Anna Karenina	2012
18	Interstellar	2014
19	American Sniper	2014
20	La La Land	2016

To obtain also the scene_id we have to modify the query as follow

```
SELECT m.film_id, m.titolo, m.anno, s.scene_id
FROM MOVIE m
JOIN SCRIPT s ON m.film_id = s.film_id
WHERE s.count_female >= 2 and s.count_male=0;
```

film_id	titolo	anno	scene_id
6	American Beauty	1999	25
11	Eternal Sunshine of the Spotless Mind	2004	160
12	Devil Wears Prada, The	2006	37
12	Devil Wears Prada, The	2006	161
12	Devil Wears Prada, The	2006	163
12	Devil Wears Prada, The	2006	192
12	Devil Wears Prada, The	2006	199

Third Level:

To assess whether or not the topic of a conversation (between two women) was a man, the following query is defined:

```
SELECT m.film_id, m.titolo, m.anno
FROM MOVIE m
WHERE EXISTS (
  SELECT 1
  FROM SCRIPT s
  WHERE s.film_id = m.film_id
   AND s.count_male = 0    AND s.count_female >= 2
   AND s.count_male_nom=0)
```

film_id	titolo	anno
6	American Beauty	1999
11	Eternal Sunshine of the Spotless Mind	2004
12	Devil Wears Prada, The	2006
13	No Country for Old Men	2007
15	Anna Karenina	2012
18	Interstellar	2014
19	American Sniper	2014
20	La La Land	2016

If we want obtain the *scene_id* we have to modify the query

```
SELECT m.film_id, m.titolo, m.anno, s.scene_id
FROM MOVIE m
JOIN SCRIPT s ON s.film_id = m.film_id
WHERE s.count_male = 0
 AND s.count_female >= 2
 AND s.count_male_nom = 0;
```

film_id	titolo	anno	scene_id
6	American Beauty	1999	19
6	American Beauty	1999	20
6	American Beauty	1999	21
6	American Beauty	1999	23
6	American Beauty	1999	24
6	American Beauty	1999	25
6	American Beauty	1999	26
6	American Beauty	1999	27
6	American Beauty	1999	38
6	American Beauty	1999	55

LIMITATION OF THE ANALYSIS

Even if this project satisfied the research questions that were set, it can use improvements:

- The text mining process might need more precision and accuracy, using more sensitive tools to:
    - Match the gender, because the code provided might fail in larger databases due to possible name redundancies or differences in genders. To provide an example, this code will correctly recognize Jared Leto in Fight Club as a male, that interpret a male character, but will fail and give back a wrong result in Dallas Buyers Club, in which Leto interpret a female character. The reason why is because the statistic method applied in this code is not able to discriminate between the gender of the actor and the gender of the role interpreted.
    - Improve the extraction of characters that begin with "A" or "THE". In general, it is recommended to enhance character extraction by using more detailed and exhaustive matching techniques.
    - Implement the code to understand which characters interact with each other. In this project, it was assumed that all extracted characters interacted with each other, but this made the result more stringent. It was enough for a man to be present in the scene (even without speaking) to fail the test.

    - Improve script analysis, as uppercase words are not used solely to indicate characters in the scene but are used randomly, making it more difficult to identify the characters in the scene.
    - Implement the third level to also identify when female characters are indirectly talking about a man.
- The running time is quite high, taking about 4/5 hours of processing. This is mainly due to gender determination using NameDataset.
- To add a single film, the whole database must be re-written from zero, so the complexity of this operation should be taken into account