Team 1 ~ Project 4

Movie Recommendation

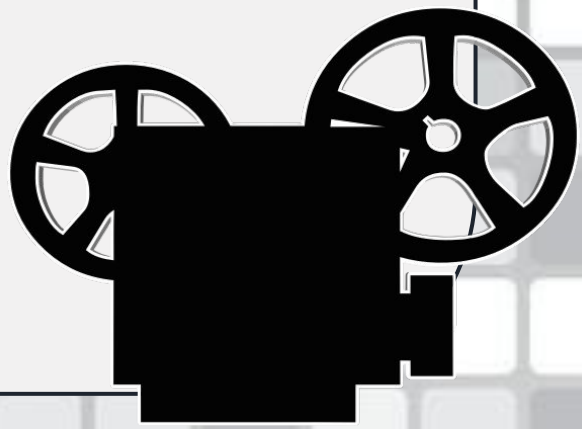
Created by: Brandon Failing, Edith Lotterman

Wei Kent Chen & Chloe Li

Project Description

Objective: This project aims to provide personalized movie recommendations to users. The user inputs a movie they enjoyed, and the system uses a content-based algorithm to suggest similar movies.

Data Source: Grouplens
https://grouplens.org/datasets/movielens/



Tools Used

Python

- Pandas
- Numpy
- Scikit
- Flask
- PySpark

Unsupervised Machine Learning

- Scikit CountVectorizer
- Scikit Cosine Similarity
- Scikit TF-IDF Vectorizer
- NLTK Snowball Stemmer

Web App

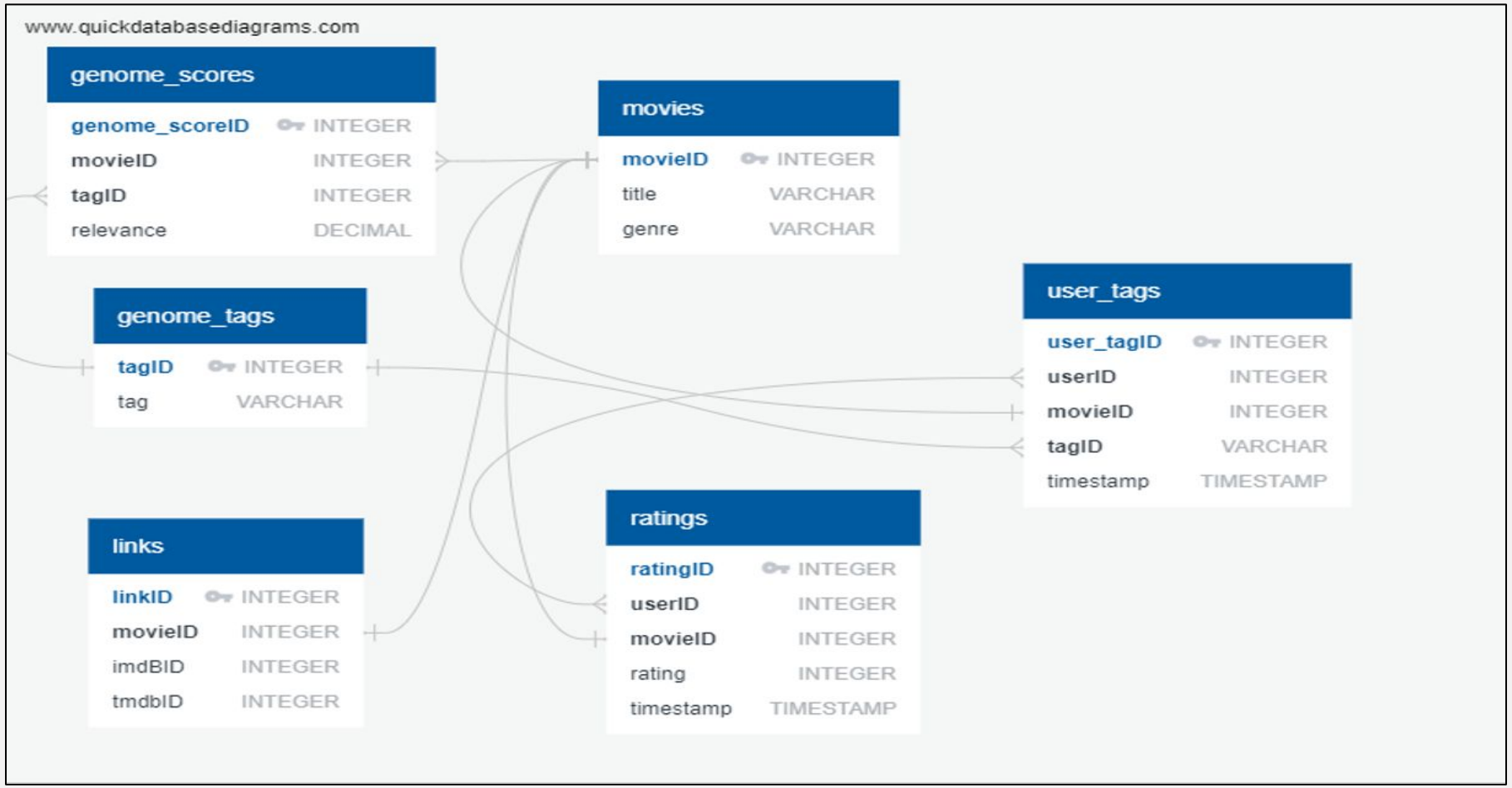
- FLASK/HTML/CSS

Data Cleanup

The process of cleaning the data encompassed recognizing any missing data, rectifying discrepancies, eliminating copies, insignificant data, and adjusting the data.

As a result of this meticulous cleaning process, the refined dataset is now appropriate for additional analysis and can provide a valuable understanding of the patterns, and projections regarding the suggestion of movie recommendations.

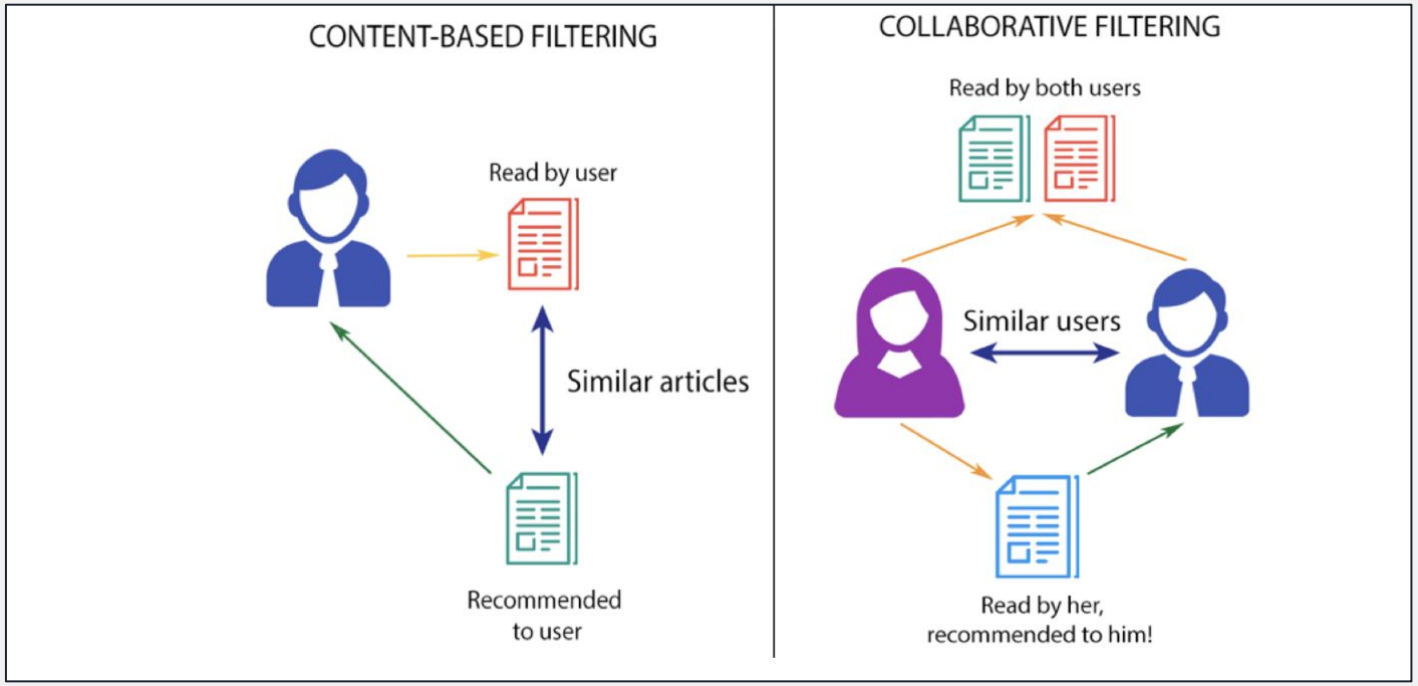
Database


www.quickdatabasediagrams.com
genome_scores
genome_scoreID INTEGER
movieID INTEGER
tagID INTEGER
relevance DECIMAL
movies
movieID INTEGER
title VARCHAR
genre VARCHAR
genome_tags
tagID INTEGER
tag VARCHAR
user_tags
user_tagID INTEGER
userID INTEGER
movieID INTEGER
tagID VARCHAR
timestamp TIMESTAMP
links
linkID INTEGER
movieID INTEGER
imdbID INTEGER
tmdbID INTEGER
ratings
ratingID INTEGER
userID INTEGER
movieID INTEGER
rating INTEGER
timestamp TIMESTAMP

Data

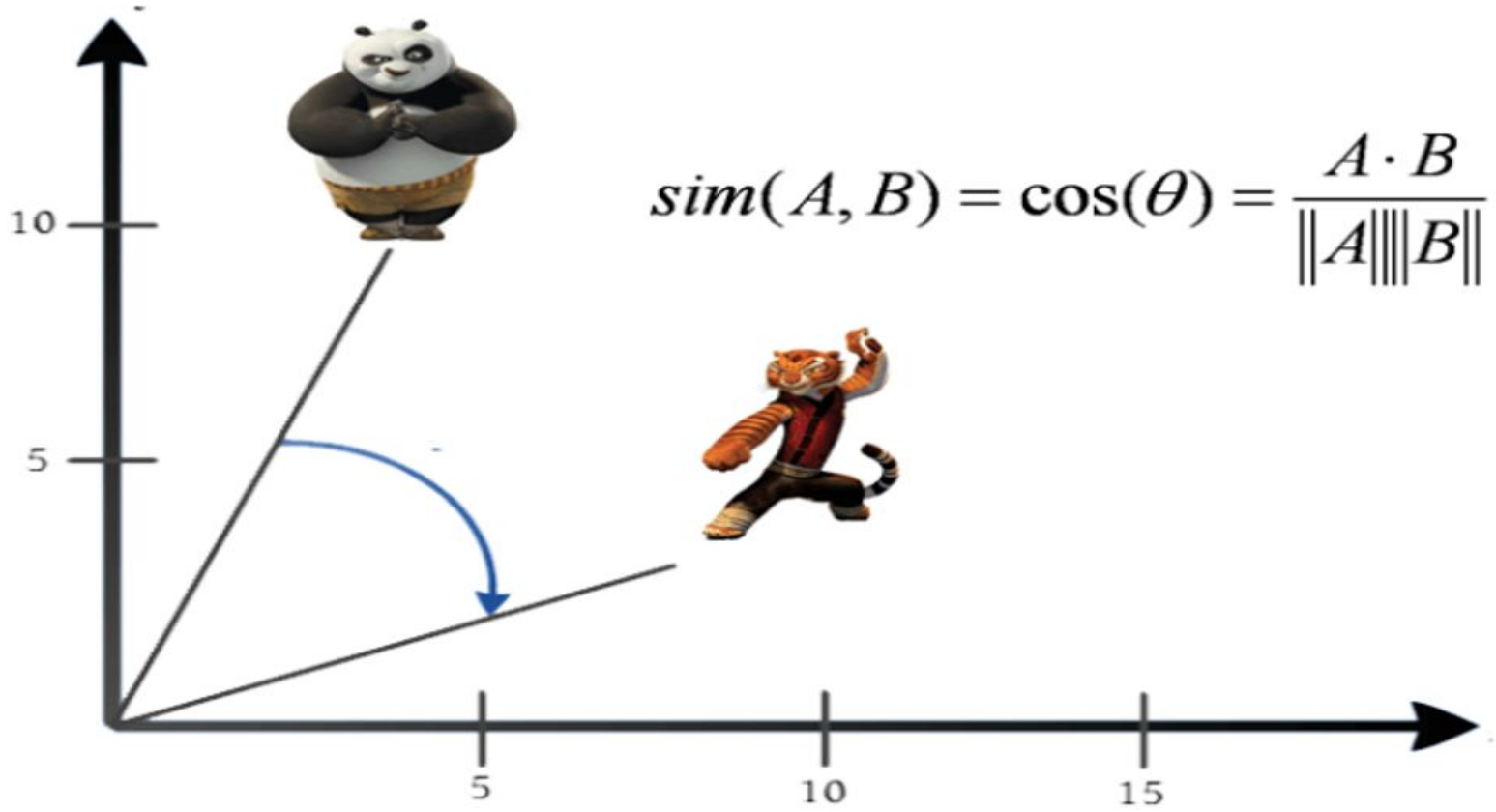
1) Dataset was loaded to AWS S3
2) Data was initially processed in PySpark because of the size of the genome_scores and ratings files. Some final processing was completed in pandas
3) NLP models require all relevant text to be in one column, as others were a lot of text concatenations
    a) Average rating was calculated and assigned to a text category.
    b) Analysis showed that keeping tags over 80% relevant ensured most movies had between 1 and 10 tags. All other tags were discarded.
    c) The year was pulled from the title and added to the analysis column.
    d) Ratings counts were binned and added to the text column.

Model Framework



- We used a content-based system since we don’t have user history
- All descriptive data gathered in one column
- NLP processing to extract features
- NLP produces vectors for analysis
- Cosine similarity used to find most similar movies

Cosine Similarity



$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Cosine Similarity Matrix from our First Model Attempt

```
[[1.         0.38490018 0.14433757 ... 0.07698004 0.14002801 0.11547005]
 [0.38490018 1.         0.20833333 ... 0.11111111 0.0404226  0.05555556]
 [0.14433757 0.20833333 1.         ... 0.16666667 0.42443734 0.25      ]
 ...
 [0.07698004 0.11111111 0.16666667 ... 1.         0.40422604 0.55555556]
 [0.14002801 0.0404226  0.42443734 ... 0.40422604 1.         0.72760688]
 [0.11547005 0.05555556 0.25       ... 0.55555556 0.72760688 1.        ]]
```

Model Optimization

Attempt 1: Scikit-Learn's Count Vectorization function

Creates a matrix of word frequencies, which looks like below:

```
Count Matrix: [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Attempt 2 Use NLTK's Snowball Stemmer function

- Stemming analyzes word to arrive at a common roots word "surfing", "surfs", "surfed" & "surf", all became "surf".

Attempt 3 Use Scikit Learn TL-IDF function

- TL-IDF uses the inverse frequency of a word to determine its importance in the document when creating the features vector.
- We used TL-IDF on the stemmed dataset.

Input Me = Despicable Me

Model 1 Results

```
[(12954, 0.7750576015460305),
 (1974, 0.7745966692414832),
 (8655, 0.7659416862050704),
 (12241, 0.7631672440718631),
 (12836, 0.7615773105863907),
 (2029, 0.7548711866766251),
 (11368, 0.7487767802667671),
 (12244, 0.7462025072446365),
 (2774, 0.741041737787324),
 (10445, 0.741041737787324)]
```

```
The Boss Baby (2017)
Antz (1998)
Ratatouille (2007)
Minions (2015)
Storks (2016)
Bug's Life, A (1998)
Cloudy with a Chance of Meatballs 2 (2013)
The Good Dinosaur (2015)
Stuart Little (1999)
Hop (2011)
```

Model 2 -After Stemming

```
[(8655, 0.8129103091557566),
 (10445, 0.800044325013193),
 (1974, 0.7938566201357352),
 (2774, 0.7858252779857413),
 (33, 0.7807200583588269),
 (12836, 0.7789043068258894),
 (4825, 0.777777777777778),
 (7907, 0.7761823345023017),
 (12913, 0.7761823345023017),
 (12221, 0.7759402897989857)]
```

```
Ratatouille (2007)
Hop (2011)
Antz (1998)
Stuart Little (1999)
Babe (1995)
Storks (2016)
Stuart Little 2 (2002)
Madagascar (2005)
Sing (2016)
The Secret Life of Pets (2016)
```

3rd Attempt Using TL-IDF

```
[(8655, 0.8129103091557566),
 (10445, 0.800044325013193),
 (1974, 0.7938566201357352),
 (2774, 0.7858252779857413),
 (33, 0.7807200583588269),
 (12836, 0.7789043068258894),
 (4825, 0.777777777777778),
 (7907, 0.7761823345023017),
 (12913, 0.7761823345023017),
 (12221, 0.7759402897989857)]
```

```
Ratatouille (2007)
Hop (2011)
Antz (1998)
Stuart Little (1999)
Babe (1995)
Storks (2016)
Stuart Little 2 (2002)
Madagascar (2005)
Sing (2016)
The Secret Life of Pets (2016)
```

THE FLASK APP

IMPORT DEPENDENCIES

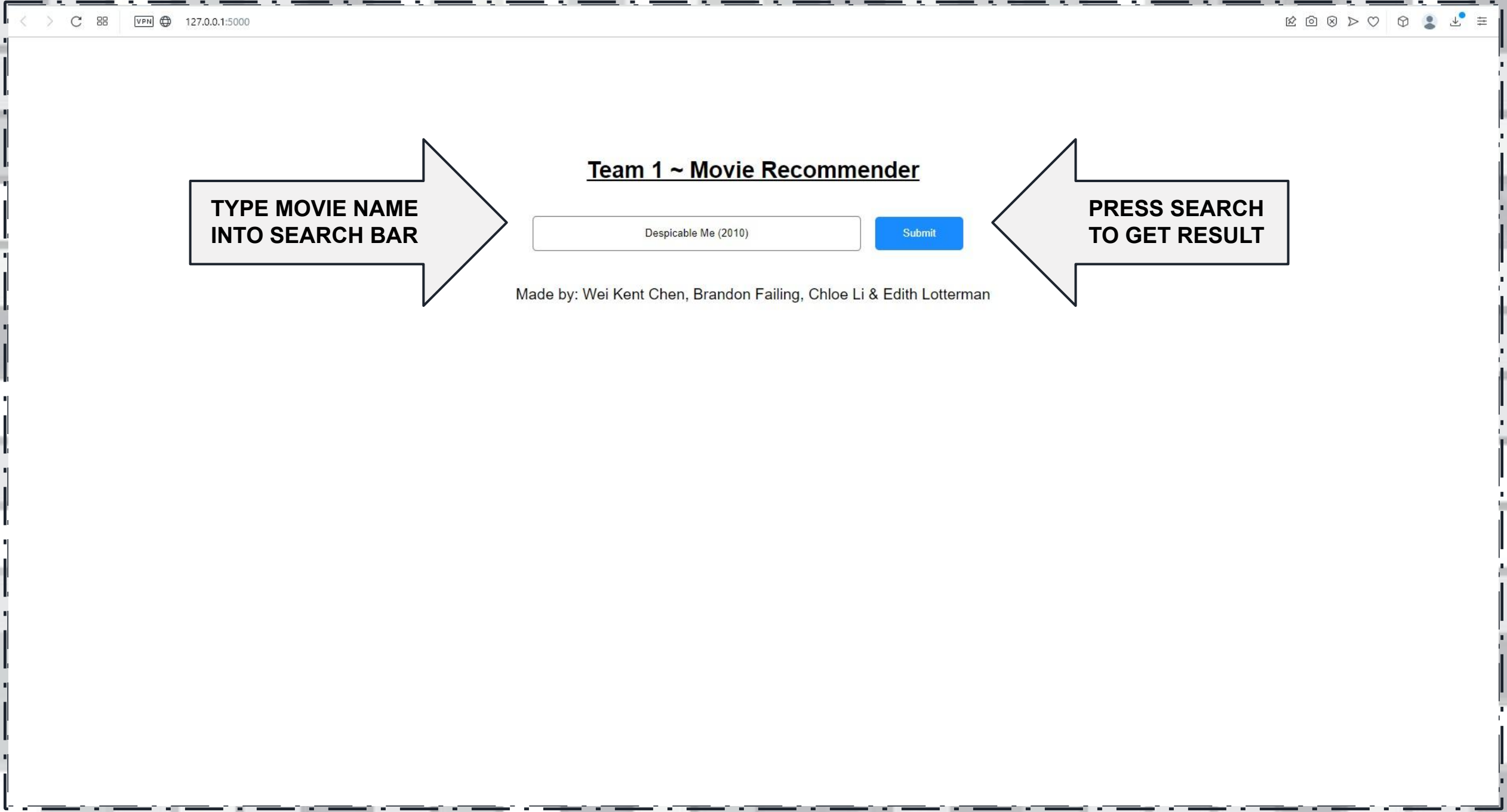
```
# Create a Flask app and import the necessary modu
import flask
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

app = flask.Flask(__name__, template_folder='templates')

data = pd.read_csv('./model/movielens-database.csv')
```

CREATE THE ML MODEL

```
# Create model
cv = CountVectorizer()

# Construct the count vectorizer matrix by fitting & transforming the
data
count_matrix = cv.fit_transform(data['combined_text'])
print("Count Matrix:", count_matrix.toarray())

# Construct cosine similarity matrix
cosine_sim = cosine_similarity(count_matrix)

# Create array with all movie titles
all_titles = [data['title'][i] for i in range(len(data['title']))]
```

```python
def get_recommendations(title):
    # Get the index of the movie that matches the title
    movie_index = data[data.title == title].index.values[0]
    # Get the pairwise similarity scores of all movies with that
    sim_scores = list(enumerate(cosine_sim[movie_index]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Create returns_df for use in app route
    name_list = []

    for movie in movie_indices:
        name = data[data.index == movie]["title"].values[0]
        name_list.append(name)

    returns_df = pd.DataFrame(name_list, columns=['Title'])
    return returns_df
```

```python
# Set up the main route
@app.route('/', methods=['GET', 'POST'])
def main():
    if flask.request.method == 'GET':
        return(flask.render_template('index.html'))

    if flask.request.method == 'POST':
        m_name = " ".join(flask.request.form['movie_name'].split())
        if m_name not in all_titles:
            return(flask.render_template('movie-not-found.html',
            name=m_name))
        else:
            result_final = get_recommendations(m_name)
            names = []
            for i in range(len(result_final)):
                names.append(result_final.iloc[i][0])

            return flask.render_template('movie-found.html',
            movie_names=names, search_name=m_name)

# Run Flask app w/ debugging.
if __name__ == '__main__':
    app.run(debug=False)
```


127.0.0.1:5000
TYPE MOVIE NAME INTO SEARCH BAR
Team 1 ~ Movie Recommender
Despicable Me (2010)
Submit
PRESS SEARCH TO GET RESULT
Made by: Wei Kent Chen, Brandon Failing, Chloe Li & Edith Lotterman

FLASK

```
<body>
    <h2><u>Team 1 ~ Movie Recommender</u></h2>
    <div class="movie">
      <form action="{{ url_for('main') }}" method="POST">
        <input type="text" id="movie_name" name="movie_name"
        placeholder="Enter a full movie name" autocomplete="off"
        autocorrect="off" required />
        <input type="submit" id="submission_button" value="Submit"/>
    </form>
  </div>
  <p class="creds" align="center">Made by: Wei Kent Chen, Brandon
  Failing, Chloe Li & Edith Lotterman</p>
</body>
</html>
```

INDEX.HTML

CSS

```
.movie{
    display: block;
    text-align: center;
}
#movie_name{
    width: 30%;
    padding: 1em;
    border-radius: 5px;
    text-align: center;
    border: 1px solid grey;
}
#submission_button{
    width: 8%;
    padding: 1em;
    margin: 1em;
    border-radius: 5px;
    color: white;
    background-color: rgb(25, 139, 253);
    border-style: none;
}
```

Back to Home Page

RETURN TO HOMEPAGE

Movie Recommendations for "Despicable Me (2010)"

Movie Title
The Boss Baby (2017)
Antz (1998)
Ratatouille (2007)
Minions (2015)
Storks (2016)
Bug's Life, A (1998)
Cloudy with a Chance of Meatballs 2 (2013)
The Good Dinosaur (2015)
Stuart Little (1999)
Hop (2011)

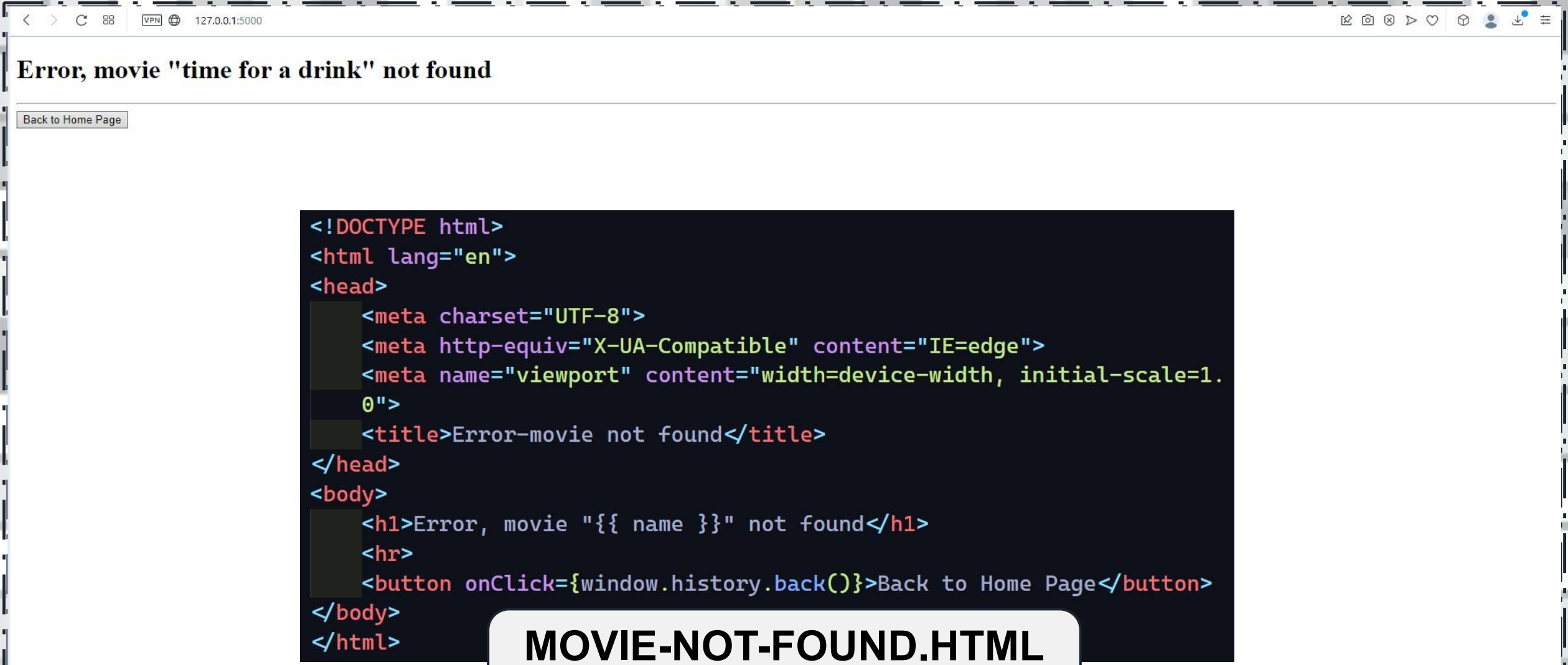
```
<body>
    <button onClick={window.history.back()}>Back to Home Page</button>
    <div class="movie">
        <h2><u>Movie Recommendations for "{{ search_name }}"</u></h2>
        <table class="movie_table" align="center">
            <tr>
                <th>Movie Title</th>
            </tr>
            <tbody class="table_body">
                <tr>
                    <td>{{ movie_names[0] }}</td>
                </tr>
                <tr>
                    <td>{{ movie_names[1] }}</td>
                </tr>
```

FLASK

MOVIE-FOUND.HTML

```
table,tr,td,th{
    border: 1px solid black;
    border-collapse: collapse;
}
td{
  padding: 10px;
}
```

CSS


127.0.0.1:5000
Error, movie "time for a drink" not found
Back to Home Page
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.
0">
<title>Error-movie not found</title>
</head>
<body>
<h1>Error, movie "{{ name }}" not found</h1>
<hr>
<button onClick={window.history.back()}>Back to Home Page</button>
</body>
</html>
MOVIE-NOT-FOUND.HTML
(NO CSS)


HOLLYWOOD
PRODUCTION
DIRECTOR
CAMERA
SCENE
TAKE


FINISH