**PHASE 4 DEVELOPMENT PART 2:**

**Model Training:**

Data collection :-

I coded all of my work, including the data collection, in python in visual studio code. It probably would’ve been simpler to scrape data from IMDb’s website, but becuase I wasn’t sure if that was allowed by IMDb, I collected the data from 4 sources:

Datasets:https://www.kaggle.com/datasets/luiscroter/netflix-original-films-imdb-scores

**IMDb’s Datasets**

IMDb provides subsets of IMDb data that are available for personal and non-commercial use, so I downloaded 7 of the TSV files from its website and I read the TSV files as Dataframe using pandas’ read\_csv() function.



After exploring the data, I ultimately used the data from 4 of these TSV files:

1. Regions shown in (called region) came from title.akas.tsv
2. IMDb ratings (called averageRating) and IMDb # of votes (called numVotes) came from title.ratings.tsv
3. Directors and writers came from title.crew.tsv
4. Runtimes (called runtimeMinutes) and media types (called titleType) came from title.basics.tsv

**RapidAPI’s Movie Database IMDb Alternative**

RapidAPI’s Movie Database IMDb Alternative is an API that I used to extract actors, titles, genres, plots, release years, and countries filmed in. It queries the data for each movie individually, which was very slow for my computer because there were over 500,000 movies to query before I decided on what data to filter out.

Therefore, I coded multiple scripts to perform different queries in parallel, and this sped up the process a lot. I created 4 copies of my Jupyter Notebook and slightly adjusted them to simultaneously call and store data for different movies. For example, in my original Jupyter Notebook, I sliced the 1st 1/5 of the title IDs, and saved it as a movies1.csv, and in the next notebook, I sliced the 2nd 1/5 of title IDs, and saved it as movies2.csv.

https://miro.medium.com/v2/resize:fit:640/1*5AzbErLbDsDYoPXphOTvnA.png

**PROGRAM OF TRAINING:**

import requests  
import json  
import traceback  
import unicodedatadef remove\_control\_characters(s):  
 return "".join(ch for ch in s if unicodedata.category(ch)[0]!="C" and ch!='\\')url = "<https://movie-database-imdb-alternative.p.rapidapi.com/>"headers = {  
 'x-rapidapi-key': key,  
 'x-rapidapi-host': "movie-database-imdb-alternative.p.rapidapi.com"  
 }json\_list = []  
error\_title\_IDs = []  
for title\_ID in title\_IDs:  
 querystring = {"i":title\_ID,"r":"json"}  
   
 try:  
 response = requests.request("GET", url, headers=headers, params=querystring)  
 json\_list.append(response.json())  
 except:  
 try:  
 json\_list.append(json.loads(remove\_control\_characters(response.text)))  
 except:  
 error\_title\_IDs.append(title\_ID)  
 print(title\_ID, traceback.format\_exc())  
   
df = pd.DataFrame(json\_list)  
df.to\_csv('movies1.csv',index=False)

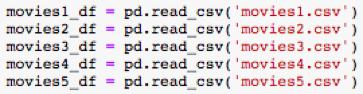
There were cases where the title ID from IMDb’s dataset displayed a different imdbID when I called the API. For example, the title ID for this in IMDb’s dataset was ‘tt0090111’.

https://miro.medium.com/v2/resize:fit:720/1*_AsRL6D-q7vM7SbjyQ8SQw.png

When I manually looked up the title IDs up on IMDb’s website, the URL redirected to the same imdbID value from the API calls. Some of them had information, while others didn’t. I added code to remove their JSON from json\_list, and I saved their IDs in a list to check afterward. Using numpy’s **.setdiff1d()** function, there were no title IDs from RapidAPI that weren’t in the IMDb datasets because the code below returned an empty set, so I didn’t have to do anything with this removed data.

https://miro.medium.com/v2/resize:fit:640/1*BCZhix5yhF6sfv6mrkt6NA.png

In my original Jupyter Notebook, I read all the movies CSV files (that I created using the above code) as DataFrames and combined them, which provided with me with the data for all the movies.



https://miro.medium.com/v2/resize:fit:720/1*MxyhlsjQ7ZMO28xr6u7lxw.png

There were features, between IMDb’s datasets and RapidAPI’s Movie Database IMDb Alternative, that referred to the same feature but had different values, so I had to use my best judgement to decide which to use. For example, the genres from the IMDb datasets had only the first 3 genres that show up in IMDb, so I used the genres from RapidAPI that had up to 8 genres for a given movie; the IMDb ratings from RapidAPI were often missing or far from the current IMDb ratings, so I used the IMDb ratings from the IMDb datasets.

**Evaluation:**

A machine learning research project and paper analyzing the efficiency of different ML algorithms using evaluation metrics and drawing a comparison between them. The data is split into training data and testing data in an 80:20 ratio in accordance with the Pareto Principle. The algorithms analyzed in this project are: SVM, Random Forest, Decision Trees and Naive Bayes.