**NAAN MUDHALVAN :IBM**

**PHASE : 4**

**TECHNOLOGY : DATA SCIENCE**

**PROJECT : IMDb SCORES PREDICTION**

**DEVELOPMENT PART 2**

**Problem Statement :**

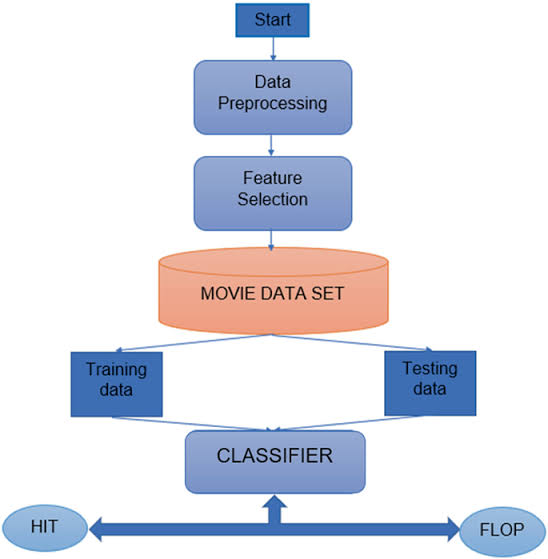
**Problem statement succinctly defines the issue or challenge that a project, research study, or endever aims to address. To craft an effective problem statement, it's important to be clear, specific, and concise.**

**PROBLEM DEFINITION AND DESIGN THINKING :**

**Problem definition is the process of clearly and precisely articulating a specific issue or challenge that needs to be addressed. It involves identifying the key aspects of the problem, understanding its scope and impact, and specifying the desired outcome or solution.**

**Design thinking is widely used in various fields, including product design, user experience design, and problem-solving in general. It's a human-centered approach that aims to create solutions that are not only functional but also resonate with the needs and emotions of the end-users.**

**Flowchart:**



**Feature Selection:**

**Feature selection is a process in machine learning and statistics where you choose a subset of relevant features (variables, attributes) to use in building a model. This is done to improve model performance, reduce computational complexity, and avoid overfitting.**

## **Rapid API’s Movie Database IMDb Alternative:**

Rapid API’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.

**Program:**

**def predict\_rating(title, k):  
#Convert title into IMDb title ID if needed, and index title ID  
if title in title\_ID\_dict:   
title = title\_ID\_dict[title]  
  
idx = ID\_indices\_dict[title]  
  
#Sort total similarity scores  
sims = np.array(budget\_sims[idx])\*2 + np.array(year\_sims[idx]) + np.array(co\_sim\_dict['One-hot Actors'][idx]) + np.array(co\_sim\_dict['One-hot Directors'][idx])\*2 + np.array(co\_sim\_dict['One-hot Genres'][idx]) + np.array(co\_sim\_dict['Plot'][idx] + np.array(co\_sim\_dict['One-hot Writers'][idx]) + np.array(co\_sim\_dict['One-hot Countries'][idx]))  
  
sims = list(enumerate(sims))  
sims = sorted(sims, key=lambda x: x[1], reverse=True)  
  
#Create list of k most similar movies that have > 5000 IMDb votes  
k\_sims = sims[0] #Index 0 is the selected movie, only included to print its data  
for sim in sims[1:]:  
if df['numVotes'].iloc[sim[0]] > 5000:  
k\_sims.append(sim)  
if len(k\_sims)>k:  
break  
  
#Make predictions using mean of IMDb ratings of k most similar movies  
movie\_indices = [i[0] for i in k\_sims]   
print(df[['Title','imdbRating','budget','Year','Actors','directors','writers','Genre','Country']].iloc[movie\_indices])  
if (df['imdbRating'].iloc[movie\_indices[1:11]].mean() > 7) and (df['imdbRating'].iloc[movie\_indices[1:6]].mean() > 7.5):  
return df['imdbRating'].iloc[movie\_indices[1:6]].mean()  
elif df['imdbRating'].iloc[movie\_indices[1:4]].mean() < 4:  
return df['imdbRating'].iloc[movie\_indices[1:4]].mean()  
else:  
return df['imdbRating'].iloc[movie\_indices[1:]].mean()**

**Implemention algorithm :**

**Of course! I'd be happy to help you implement an algorithm. Could you please specify which algorithm you're interested in? There are many different types of algorithms for various purposes, such as sorting, searching, machine learning, and more. Let me know which specific algorithm you'd like assistance with, and I'll do my best to guide you through it!**

**import warnings**

**import pandas as pd**

**import numpy as np**

**import scipy as sc**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**plt.style.use('fivethirtyeight')**

**pd.set\_option('display.max\_rows', 50)**

**pd.set\_option('display.max\_columns', 50)**

**warnings.filterwarnings('ignore')**