Topic	Flask Mockup 2	
Class Description	**Students complete the Flask API for their mobile app on movie recommendation.**	
Class	**C142**	
Class time	**45 mins**	
Goal	● Student completes the Flask API for movie recommendation App	
Resources Required	● Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 mins** **15 min** **20 min** **5 min**

CONTEXT

- **Review the concepts learned in the earlier classes**

Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	Hi <Student Name>! We completed the Flask API for the first screen of our mobile app! Now, we want to complete the API by thinking through the second screen of our app so that we can start with the React Native part!	

	I have an exciting quiz question for you! Are you ready to answer this question? *Teacher clicks on the*  *button on the bottom right corner of their screen to start the In-Class Quiz.* *A quiz will be visible to both you and the student.* *Encourage the student to answer the quiz question.* *The student may choose the wrong option, help the student to think correctly about the question and then answer again.* *After the student selects the correct option, the*  *button will start appearing on your screen.* *Click the End quiz to close the quiz pop-up and continue the class.*	
	For the second page, we will be displaying recommendations based on the user's preference, so we will build 1 API for that. Now, we can also display a list of most popular movies! For this, we can build 1 API too and give popular	**ESR:** "Yes!"

	movie names using our demographic filtering method! In all, we want to build these two APIs. Sounds like a plan?	
	Let's start!	
Teacher Initiates Screen Share		
CHALLENGE • **Help the student get the data right**		
Step 2: Teacher-led Activity (15 min)	*<This class will be divided in 2 phases. In the first phase, the student will get their data right. In the second phase, the student will be building the rest of the two APIs>*	
	If we notice in our data, we don't have the movie's poster link. That seems to play an important role in a movie recommendation app! Worry not! We have a CSV for you that we will be merging with our existing CSV to create a final csv that we will use in our API! *https://raw.githubusercontent.com/whitehatjr/c-142/main/movie_links.csv* *<The CSV with movie's poster links can be downloaded from above!>*	**ESR:** varied

	Now, for this, let's first get this file in our Flask APIs code directory. Next, on studying the data, we can see that this CSV has 2 columns. First is the name of the movie and the second is the link to the movie's poster.	
movie_links.csv 1 name,imdb_link		
	Now, let's create a new file **merge_csv.py** and in this, let's make the imports, read our original **movies.csv** and separate out the headers and the data. ``` import csv with open('movies.csv') as f: reader = csv.reader(f) data = list(reader) all_movies = data[1:] headers = data[0] ``` Here, we are importing the csv library. We are then opening our movies.csv and reading the data out of it. We are storing all our movie's data in **all_movies** and we are storing the headers in **headers**.	
	Let's add a new variable to the header, create a new csv file and append our new headers into this new csv.	

	``
	headers.append("poster_link")
	
	with open("final.csv", "a+") as f:
	  csvwriter = csv.writer(f)
	  csvwriter.writerow(headers)
	``
	
	Here, we are appending the variable **poster_link** in our headers. We are then creating a new file **final.csv** and we are appending the headers into it.
	
	The **"a+"** attribute means that we want to append in the file. This means that the old data would not be deleted but the new data will be added. **"w"** on the other hand removes all the data from the file and then adds new data.	

	Let's now read the contents of the movie's poster links and store it in a variable.
	
	``
	with open("movie_links.csv") as f:
	  reader = csv.reader(f)
	  data = list(reader)
	  all_movie_links = data[1:]
	``	

	Finally, let's add our logic to match the movie's data with its poster link and then save this data in the new csv. Now, if we observe closely, the original data that we have contains 4,807 rows while the movie's poster	

link contains 4,748 rows. This means that we do not have poster links for a few movies. Let's be mindful of that while writing our logic.

```
for movie_item in all_movies:
   poster_found = any(movie_item[8]
in movie_link_items for
movie_link_items in all_movie_links)
   if poster_found:
       for movie_link_item in
all_movie_links:
           if movie_item[8] ==
movie_link_item[0]:

movie_item.append(movie_link_item[1])
               if len(movie_item) ==
28:
                   with
open("final.csv", "a+") as f:
                       csvwriter =
csv.writer(f)

csvwriter.writerow(movie_item)
```

Here, we are first iterating over all the movie data that we have.

We are then checking if there is any row in the movie's poster link data that contains the name of the movie with the **any()** function. This will simply return **True or False**.

If we found a poster, we are iterating over all the movies in the movie's poster links data and comparing the

names. If we find the corresponding poster of a movie, we are appending this link into our original movie's data and then writing this entire movie's data in the **final.csv**.

Before that, we are also checking if the total length is 28 or not. Ideally, we should have 28 columns. If the number of columns is not 28, we are not adding the movie's data. This will remove inconsistency from our data.

merge_csv.py

```python
import csv

with open('movies.csv') as f:
    reader = csv.reader(f)
    data = list(reader)
    all_movies = data[1:]
    headers = data[0]

headers.append("poster_link")

with open("final.csv", "a+") as f:
    csvwriter = csv.writer(f)
    csvwriter.writerow(headers)

with open("movie_links.csv") as f:
    reader = csv.reader(f)
    data = list(reader)
    all_movie_links = data[1:]

for movie_item in all_movies:
    poster_found = any(movie_item[8] in movie_link_items for movie_link_items in all_movie_links)
    if poster_found:
        for movie_link_item in all_movie_links:
            if movie_item[8] == movie_link_item[0]:
                movie_item.append(movie_link_item[1])
                if len(movie_item) == 28:
                    with open("final.csv", "a+") as f:
                        csvwriter = csv.writer(f)
                        csvwriter.writerow(movie_item)
```

	Okay, now all we have to do is to use this **final.csv** instead of the old **movies.csv** in our Flask API. For this, we will create 2 more files. The first file would contain our code of demographic filtering. The second file will contain the code for content based filtering and the function that returns the list of movies that it recommends. We will then use these 2 files inside our API, just like how we did in the Digit Recognition App. Remember? Can you tell how it will help our code?	**ESR:** It will do all the processing when the server starts and gives good performance later on!
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share** • **Teacher gets into Fullscreen**		
ACTIVITY • **Student codes to complete the remaining two APIs.**		

Step 3: Student-Led Activity (20 min)	*<Let student code most of the part and help them wherever they are stuck>*	
	Help the student create the first file for demographic filtering. Import numpy and pandas here. Read **final.csv** into a DataFrame. Calculate the values of C, m and find the movies that have more votes than 0.9 quantile of the movie, just like how we did in Google Colab. Define a function to calculate the weighted rating and create a new column of score in the dataframe for all the movies. Create a variable with top 20 movies as a list. *<Sample Code>*	*The student codes the demographic filtering method in the new file.*

```
import pandas as pd
import numpy as np

df = pd.read_csv('final.csv')

C = df['vote_average'].mean()
m = df['vote_count'].quantile(0.9)
q_movies = df.copy().loc[df['vote_count'] >= m]

def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
```

	``` R = x['vote_average'] return (v/(v+m) * R) + (m/(m+v) * C) q_movies['score'] = q_movies.apply(weighted_rating, axis=1) q_movies = q_movies.sort_values('score', ascending=False) output = q_movies[['title', 'vote_count', 'vote_average', 'poster_link']].head(20).values.tolist() ``` *Help the student create API to return the 20 most popular movies based on this output. This will be a GET API.*	*Student codes the API.*
	Help the student create the second file with the function to give recommendations based on the content based filtering's cosine similarity classifier, similar to what was done in Google Colab. *This will be in a new file* ***content_filtering.py*** Remember, we do not have to do any sort of processing. We have already exported this CSV from google colab, which means that it already contains the metadata soup string that we created. One thing that we need to ensure is to drop the rows that do not have a valid metadata soup string. We can do that with the following code:	*The student codes the content based filtering method in a new file.*

```
df = pd.read_csv('final.csv')
df = df[df['soup'].notna()]
```

<Sample Code>

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
import numpy as np

df = pd.read_csv('final.csv')
df = df[df['soup'].notna()]

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df['soup'])

cosine_sim = cosine_similarity(count_matrix, count_matrix)

df = df.reset_index()
indices = pd.Series(df.index, index=df['title'])

def get_recommendations(title):
   idx = indices[title]
   sim_scores = list(enumerate(cosine_sim[idx]))
   sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
   sim_scores = sim_scores[1:11]
   movie_indices = [i[0] for i in sim_scores]
   return df[['title', 'vote_count', 'vote_average',
'poster_link']].iloc[movie_indices].values.tolist()
```

	Help the student create the second API to send a list of recommended movies. *Here, the student would be required to remove duplicate entries of a given movie. (if in case same movie was*	*Student codes the API.*

	recommended for two different liked movies. For example, Godfather 3 can be recommended both for Godfather 1 and Godfather 2. Hence, it is important for us to ensure we are not sending the same movie recommendation twice.)	
	<Optional> Test your APIs using postman if there's time.	
Teacher Guides Student to Stop Screen Share		
FEEDBACK • **Appreciate the student for their efforts** • **Identify 2 strengths and 1 area of progress for the student**		
Step 4: Wrap-Up (5 min)	Great! We have successfully completed our Flask API. Now in the next class, we will be starting to work on our mobile app for Movie Recommendation System!	
Teacher Clicks ✖ End Class		

Activity	Activity Name	Links
Teacher Activity 1	Solution for Flask API	https://github.com/whitehatjr/c-142