Topic	FLASK MOCKUP 1	
Class Description	**The student will brainstorm the app they are aiming to build on Movie Recommendation and then build the Flask API for the first screen of it.**	
Class	**PRO C141**	
Class time	**45 mins**	
Goal	● Brainstorm on the app between the student and the teacher. ● Merging two csv files to create the final version of csv. ● Build Flask API for the first screen of the App.	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-Led Activity 1** **Student-Led Activity 1** **Wrap-Up**	**5 mins** **15 mins** **20 mins** **5 mins**
WARM-UP SESSION - 5 mins		
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA>		

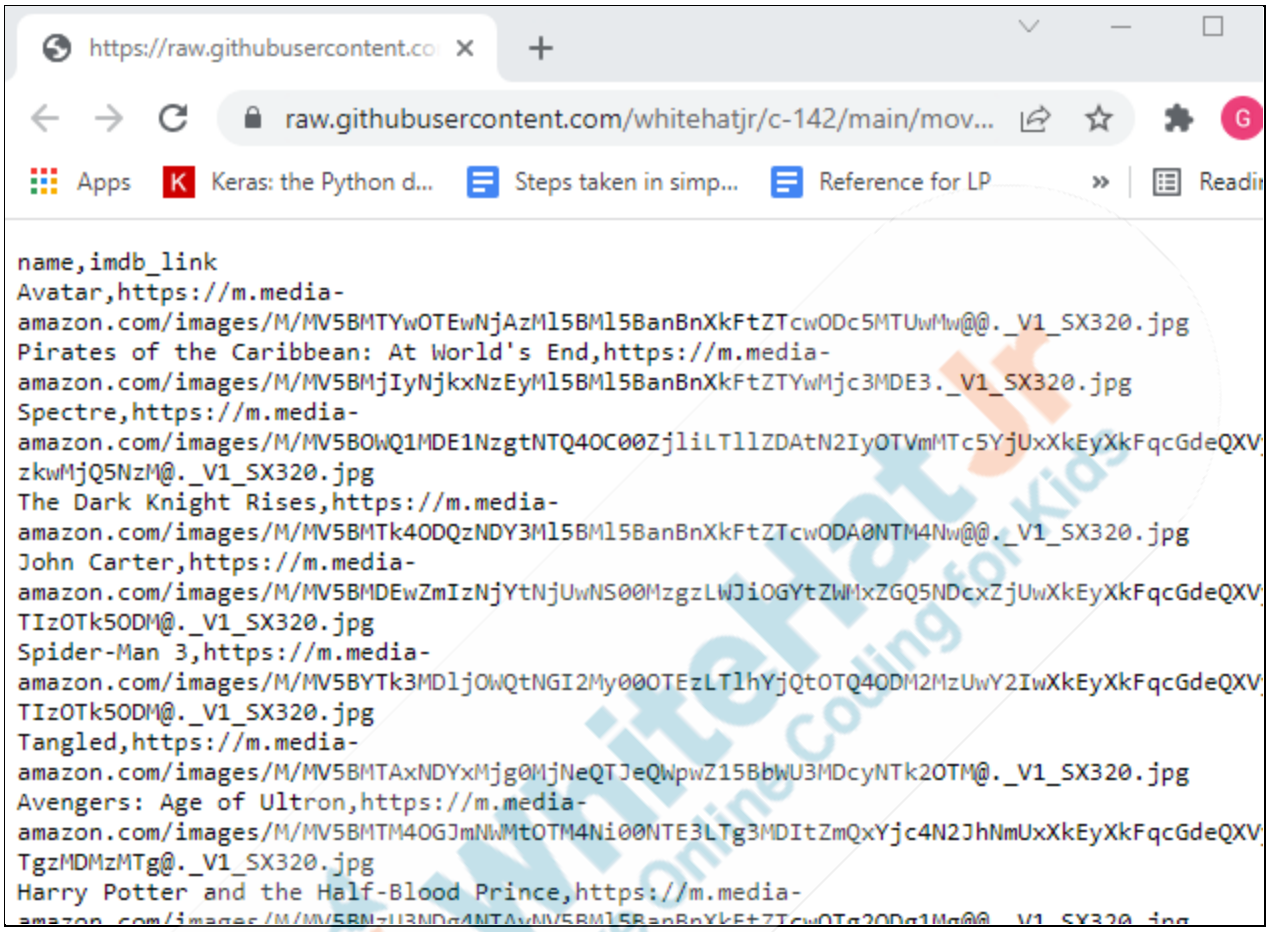
Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes, I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide # to #** <**Note**: Only Applicable for Classes with VA>	
Activity Details **Following are the session deliverables:** • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Action	**Student Action**
As of now, you must have built a lot of great apps and games using different technologies! You must have experienced getting recommendations on various platforms on the internet, but what if we could build a dedicated app that can provide users with the	**ESR**: Varied.

recommendations on which movie to watch next, based on what they have already watched. We are going to build exactly that! What do you think, how should this app be like in terms of User Experience and User Interface? *<Get the student to think about the app>* Great! Now let's brainstorm and start building it! *<This is a guided project. The student is free to add their own creativity to the app, but some basic templates need to be followed>* *<The app will consist of 2 screens. The first one would be to take user's input on the type of movie that the user likes and the second one would be to provide recommendations to the user>* *<We encourage the student to think and come up with a UI/UX design of their own, however the functionality should be more or less the same>* *<For reference purposes, we will be providing a sample app that we are going to create and the code explanations would be mentioned in the document>* *<Student can either build a similar app or add their own unique touch to the application>* *<Let the student lead the coding part and help them wherever required.>*	**ESR**: Varied.

<Teacher can refer to the sample code and explanation to help student on how a problem can be solved> Alright! Now let's think about the App first! *Brainstorm about the app with students.* The app, to begin with, will have 2 screens. The first screen would be to get the **user input** and the second screen will show the **recommendations** and movies **liked** by the user**.** Let's take this screen by screen and only focus on the backend APIs for the first screen in this class! The first screen would be where we will be taking input from the user. *Student thinks about the features* What do you think should be the features of the first page? *Note - The backend's functionality would be the same, however the student can come up with their own logic for it* Great! Now, we can simply build 4 APIs for these 4 functionalities. All the APIs (to get the details of a movie, to mark a movie as **liked**, to make a movie as **disliked,** to make a movie as	*Student brainstorms.* **ESR:** The first page should include - ~ User's ability to see a movie. ~ Users can mark the movie as liked. ~ Users can mark the movie as not liked. ~ Users can say that they did not watch a movie.

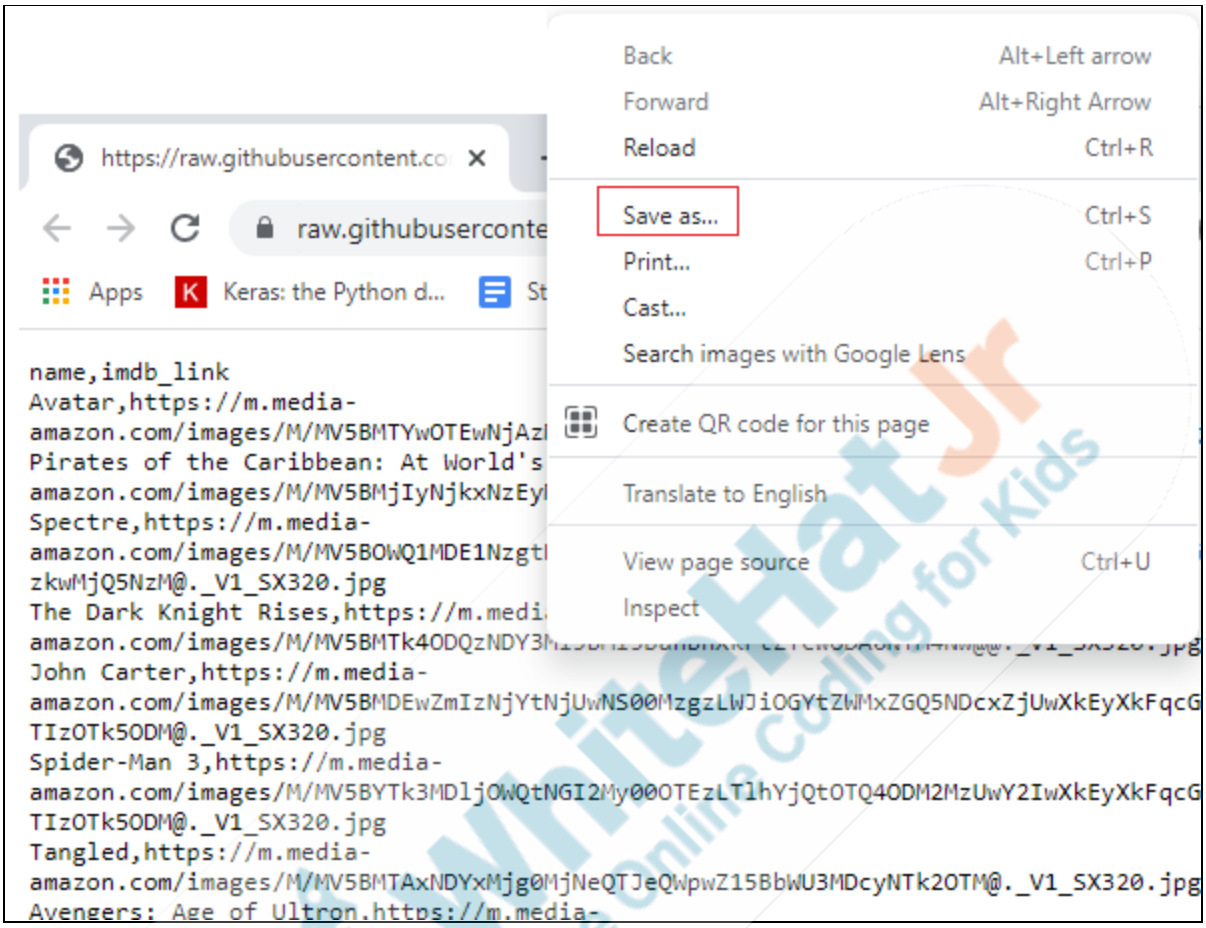
not watched yet) should listen to incoming **GET** requests. Also, how do we store all this data? We can create 3 different lists. ● List of movies **liked** by the user. ● List of movies **disliked** by the user. ● List of movies that the user **hasn't watched** yet. As the student marks the movie as **liked**, **disliked,** or **not watched** yet, we can remove that movie from the data of all movies, and add it to the designated list. Can you tell me why we need to do that? Yes! We will keep only one entry of the movie in any of the 4 storages to avoid having any duplicates. This means that if a user likes a movie, we will first have to remove that movie from our data of all movies and then add that movie in our list of movies that the user likes. Okay, now we are ready to start coding.	*The student thinks.* **ESR:** So that we do not show the same movie again and again to the user and have duplicates in our data.
Teacher Ends Slideshow	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	

ACTIVITY • **Brainstorm the app together and arrive with a set of features that the app might require**	
Teacher Action	**Student Action**
Before creating the APIs, let's have a look at our **movies.csv** file which we downloaded in the last class. If we notice our **movies.csv** file carefully, we don't have the **link** for the **movie's poster image**. That seems to play an important role in a movie recommendation app. Don't you think the same way? Don't worry, we have a **csv file** that we will be merging with our existing **movies.csv** file to create a **final csv** file that we will use, while building our APIs. Use this link, to get the poster links for our movies. *https://raw.githubusercontent.com/whitehatjr/c-142/main/movie_links.csv* When you click on this link, it will direct you to a raw GitHub page, which contains the required poster links.	**ESR**: Yes

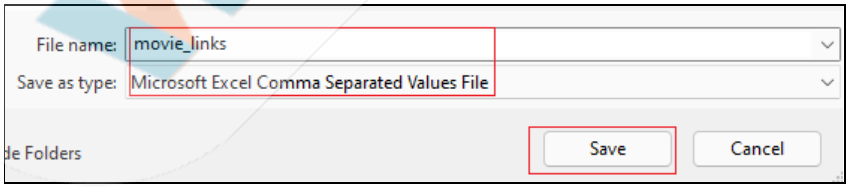



```
name,imdb_link
Avatar,https://m.media-
amazon.com/images/M/MV5BMTYwOTEwNjAzMl5BMl5BanBnXkFtZTcwODc5MTUwMw@@._V1_SX320.jpg
Pirates of the Caribbean: At World's End,https://m.media-
amazon.com/images/M/MV5BMjIyNjkxNzEyMl5BMl5BanBnXkFtZTYwMjc3MDE3._V1_SX320.jpg
Spectre,https://m.media-
amazon.com/images/M/MV5BOWQ1MDE1NzgtNTQ4OC00ZjliLTllZDAtN2IyOTVmMTc5YjUxXkEyXkFqcGdeQXV
zkwMjQ5NzM@._V1_SX320.jpg
The Dark Knight Rises,https://m.media-
amazon.com/images/M/MV5BMTk4ODQzNDY3Ml5BMl5BanBnXkFtZTcwODA0NTM4Nw@@._V1_SX320.jpg
John Carter,https://m.media-
amazon.com/images/M/MV5BMDEwZmIzNjYtNjUwNS00MzgzLWJiOGYtZWMxZGQ5NDcxZjUwXkEyXkFqcGdeQXV
TIzOTk5ODM@._V1_SX320.jpg
Spider-Man 3,https://m.media-
amazon.com/images/M/MV5BYTk3MDljOWQtNGI2My00OTEzLTlhYjQtOTQ4ODM2MzUwY2IwXkEyXkFqcGdeQXV
TIzOTk5ODM@._V1_SX320.jpg
Tangled,https://m.media-
amazon.com/images/M/MV5BMTAxNDYxMjg0MjNeQTJeQWpwZ15BbWU3MDcyNTk2OTM@._V1_SX320.jpg
Avengers: Age of Ultron,https://m.media-
amazon.com/images/M/MV5BMTM4OGJmNWMtOTM4Ni00NTE3LTg3MDItZmQxYjc4N2JhNmUxXkEyXkFqcGdeQXV
TgzMDMzMTg@._V1_SX320.jpg
Harry Potter and the Half-Blood Prince,https://m.media-
```

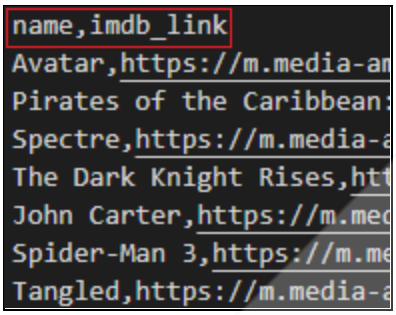
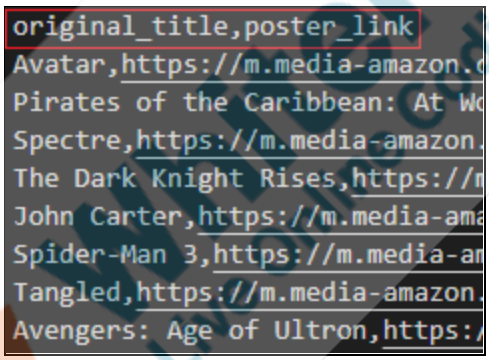
To download the data, right click on the window and click on **Save as** option.



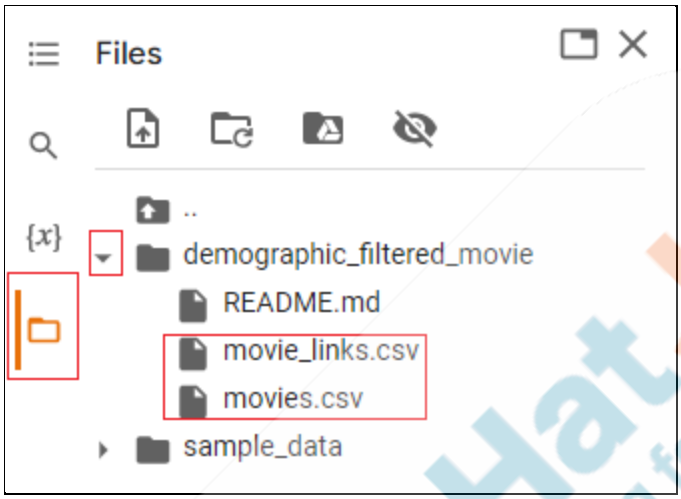
A window will appear which will ask you to save the file at any location you want. Make sure that you have selected the right directory, and then click on the **Save** button.



The file will be downloaded.

Open the **movie_links.csv** file with the **Visual Studio code** editor.	
	
Change the column names from **name, imdb_link** to **original_title** and **poster_link.** This will come handy later in the class.	
	
Now it's time to **merge** this **movie_links.csv** and our **movies.csv** file to create a final csv file. For your ease, both the files are uploaded on a GitHub link. Let's start with merging of both the files. For that, open the *Teacher Activity 1* and run all the cells. To verify, let's go to the **files** option, **expand** the **demographic_filtered_movie** directory and you will see,	

both **movie_links.csv** and **movies.csv** files are downloaded.



Create a DataFrame out of these files.

```
import pandas as pd
movies_df = pd.read_csv('/content/demographic_filtered_movie/movies.csv')
movie_links_df = pd.read_csv('/content/demographic_filtered_movie/movie_links.csv')
```

Let's get the dimensions for both the dataframe using the **shape** property.

```
# printing shape for both the df
print('movie_df shape' , movies_df.shape)
print('movie_links_df shape' , movie_links_df.shape)
```

```
movie_df shape (4803, 28)
movie_links_df shape (4747, 2)
```

We can clearly see that our **movies_df** has **4803 rows** or **4803 movies** whereas our **movie_links_df** has **4747 rows or 4747 movies,** which means we don't have poster links for all the movies.

Now, let's get the information for **movies_df** and **movie_links_df**, using the **info()** method, so that we can find a common column, required for merging both the dataframe.

```
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 28 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4803 non-null   int64
 1   index                 4803 non-null   int64
 2   budget                4803 non-null   int64
 3   genres                4803 non-null   object
 4   homepage              1712 non-null   object
 5   id                    4803 non-null   int64
 6   keywords              4803 non-null   object
 7   original_language     4803 non-null   object
 8   original_title        4803 non-null   object
 9   overview              4799 non-null   object
 10  popularity            4803 non-null   float64
```

Let's use the **info()** method for **movie_links_df** as well.

```
movie_links_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4747 entries, 0 to 4746
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   original_title  4747 non-null   object
 1   poster_link     4747 non-null   object
dtypes: object(2)
memory usage: 74.3+ KB
```

We can clearly see that, both the dataframes have a column named **'original_title'.** We will use this column to **merge** both the dataframes.

```
final_df = pd.merge(movies_df, movie_links_df, on="original_title")
```

Let's print the information, using the **info()** method.

```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4748 entries, 0 to 4747
Data columns (total 29 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            4748 non-null   int64
 1   index                 4748 non-null   int64
 2   budget                4748 non-null   int64
 3   genres                4748 non-null   object
 4   homepage              1701 non-null   object
 5   id                    4748 non-null   int64
 6   keywords              4748 non-null   object
 7   original_language     4748 non-null   object
 8   original_title        4748 non-null   object
 9   overview              4744 non-null   object
 10  popularity            4748 non-null   float64
 11  production_companies  4748 non-null   object
 12  production_countries  4748 non-null   object
 13  release_date          4747 non-null   object
 14  revenue               4748 non-null   int64
 15  runtime               4746 non-null   float64
 16  spoken_languages      4748 non-null   object
 17  status                4748 non-null   object
 18  tagline               3929 non-null   object
 19  title_x               4748 non-null   object
 20  vote_average          4748 non-null   float64
 21  vote_count            4748 non-null   int64
 22  title_y               4748 non-null   object
 23  cast                  4748 non-null   object
 24  crew                  4747 non-null   object
 25  weighted_rating       4748 non-null   float64
 26  director              4719 non-null   object
 27  soup                  4736 non-null   object
 28  poster_link           4748 non-null   object
dtypes: float64(4), int64(6), object(19)
```

Great, now we have **poster links** for our movies as well.

Let's convert our **final_df** to csv.

```
final_df.to_csv('final.csv')
```

Finally, let's download the **final.csv** file.

```
from google.colab import files
files.download('final.csv')
```

Teacher Starts Slideshow

Slide # to #

<**Note**: Only Applicable for Classes with VA>

Refer to speaker notes and follow the instructions on each slide.

So now it's your turn. We have one more class challenge for you.

Can you solve it?

Let's try. I will guide you through it.

Teacher Ends Slideshow

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

ACTIVITY • **Student codes to build the APIs in Flask and uses the data that we just downloaded**	
Teacher Action	**Student Action**
It's time to create APIs. For your ease we have already uploaded the **final.csv** file into the Student Activity 1. Click on this link and download all the files. Traverse to this folder using the **command prompt** and create a Python virtual environment in it using the command **python -m venv env**. Activate the environment and install the flask and pandas module using the command **pip install flask** and **pip install pandas.** *Note : Help the student set up a basic Flask Project inside a virtual environment.* Open project folder with the help Visual Studio code editor and click on **main.py** file. We have already create a dataframe using the command, **movies_data = pd.read_csv('final.csv'),** Let's extract **original_title, poster_link, release_data , runtime and weighted_rating** from this dataframe, for all the movies, and create a new dataframe out of it as,	*The student sets up a Flask Project.*

all_movies = movies_data[['original_title' , 'poster_link' , 'release_date' , 'runtime' , 'weighted_rating']]	

```
all_movies = movies_data[["original_title","poster_link","release_date","runtime","weighted_rating"]]
```

Let's create 3 lists, so that we can segregate our data into different categories.	

```
liked_movies = []
not_liked_movies = []
did_not_watch = []
```

Let's create a method which will extract the **original_title** , **poster_link, release_data, runtime and weighted_rating** for the **first movie** in the dataframe and store that data in a **dictionary** format. Finally, it will return that data.	

```
def assign_val():
    m_data = {
        "original_title": all_movies.iloc[0,0],
        "poster_link": all_movies.iloc[0,1],
        "release_date": all_movies.iloc[0,2] or "N/A",
        "duration": all_movies.iloc[0,3],
        "rating":all_movies.iloc[0,4]/2
    }
    return m_data
```

Help the student write the first API, where they will be sending the data of a movie as a JSON response from our main all_movies dataframe.	*Student codes.*

Now, it's time to create our first API. Define a new route and specify the URL as **'/movies'**, which will listen for incoming **GET** requests only. Define a **decorator** method named **get_movies(),** which will extract the **original_title, poster_link, release_data, runtime and weighted_rating** for the **first movie** in the dataframe by calling the **assign_val() method** and will return this **data** along with the **success status** in JSON format, whenever a GET request is received on this API.	

```
@app.route("/movies")
def get_movie():
    movie_data = assign_val()

    return jsonify({
        "data": movie_data,
        "status": "success"
    })
```

Make the student write the second API, when the user has liked a movie. Here, we have to remove the movie's entry from our all_movies dataframe and then add this entry into the list of liked movies. Define a new route and specify the URL as **'/like'**, which will listen for incoming **GET** requests only. Define a decorator method named **liked_movie(),** which will, • Extract the details for the first movie from our **all_movies** dataframe by calling the **assign_val() method**. • Append it to the **liked_movies** list.	*Student codes.*

- **Drop** the first movie from the **all_movies** dataframe.
- Reset the index of our dataframe.
- Finally, return as **success status** in **JSON** format.

Note: Here we are changing the all_movies dataframe from inside of the function, so we have to use the global keyword before all_movies dataframe.

```
@app.route("/like", methods=["POST"])
def liked_movie():
    global all_movies
    movie_data=assign_val()
    liked_movies.append(movie_data)
    all_movies.drop([0], inplace=True)
    all_movies=all_movies.reset_index(drop=True)
    return jsonify({
        "status": "success"
    })
```

Make the student write the third API, when the user has disliked a movie. Here, we have to remove the movie's entry from our all_movies dataframe and then add this entry into the list of not_liked_movies list.

Same as earlier, but this time the user is marking the movie as disliked so we are moving the movie to the **not_liked_movies** list.

```python
@app.route("/dislike", methods=["POST"])
def unliked_movie():
    global all_movies

    movie_data=assign_val()
    not_liked_movies.append(movie_data)
    all_movies.drop([0], inplace=True)
    all_movies=all_movies.reset_index(drop=True)

    return jsonify({
        "status": "success"
    })
```

Finally, make the 4th API where the user has not watched the movie. Again, we have to remove the movie from our all_movies dataframe and then add this entry into the list of movies that the user has not watched.

The code would look something like below -

Same as earlier, but this time the user is marking the movie as not watched, so we are moving the movie to the **did_not_watch** list.

```python
@app.route("/did_not_watch", methods=["POST"])
def did_not_watch_view():
    global all_movies

    movie_data=assign_val()
    did_not_watch.append(movie_data)
    all_movies.drop([0], inplace=True)
    all_movies=all_movies.reset_index(drop=True)

    return jsonify({
        "status": "success"
    })
```

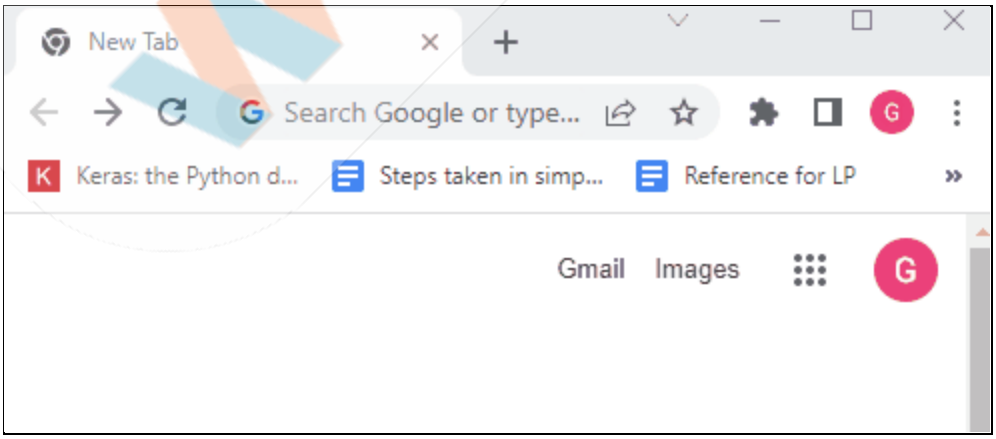
Go to your command prompt and run this file using the command, **python main.py**

If there are no errors, you will a see message which says that your server is **Running on** http://127.0.0.1:5000/ **(Press CTRL + C to quit)**

```
(env) C:\Users\ITRS-1795\Desktop\movie recommendation>python main.py
<_io.TextIOWrapper name='movies.csv' mode='r' encoding='utf-8'>
 * Serving Flask app 'main' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
<_io.TextIOWrapper name='movies.csv' mode='r' encoding='utf-8'>
 * Debugger is active!
 * Debugger PIN: 242-698-832
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Once your server is up and running, let's test all the APIs as:

1. Open localhost link http://127.0.0.1:5000/ in browser.
2. Click on the URL tab to add **/movies** or **/not_liked_movies** or **/did_not_watch** right after the localhost link to check the API response.



Refer link:
https://s3-whjr-curriculum-uploads.whjr.online/087987e5-243f-4c6c-884f-059f2527e632.gif

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Teacher Starts Slideshow
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Continue WRAP-UP Session
Slide # to #
<**Note**: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Continue WRAP-UP Session

Slide # to #

<**Note**: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get "hats-off" for your excellent work!	*Make sure you have given at least 2 hats-off during the class for:*  

In the next class we'll complete the Flask API for their mobile app on movie recommendation.	Strong Concentration +10
PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks ✖ End Class	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1HOhcb82cjASRfmDSyK2tSm8SL4CWsXqG?usp=sharing
Teacher Activity 2	Reference Code 1: Merge Data	https://colab.research.google.com/drive/1QniLPZYId5eJEf2qs18JZkahpILP-P25?usp=sharing
Teacher Activity 3	Reference Code 2: API	https://github.com/procodingclass/PRO-C142-Reference-Code.git
Teacher Activity 4	Output	https://s3-whjr-curriculum-uploads.whjr.online/087987e5-243f-4c6c-884f-059f2527e632.gif
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/25ef3302-2df9-45a8-91f2-0fd3d54c8e76.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C141-Project-Solution.git
Teacher Reference 3	Visual-Aid	Will be added after VA creation
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/9685bb19-293c-45ec-b6e5-c6bbc2cea1a7.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C141-Student-Activity.git