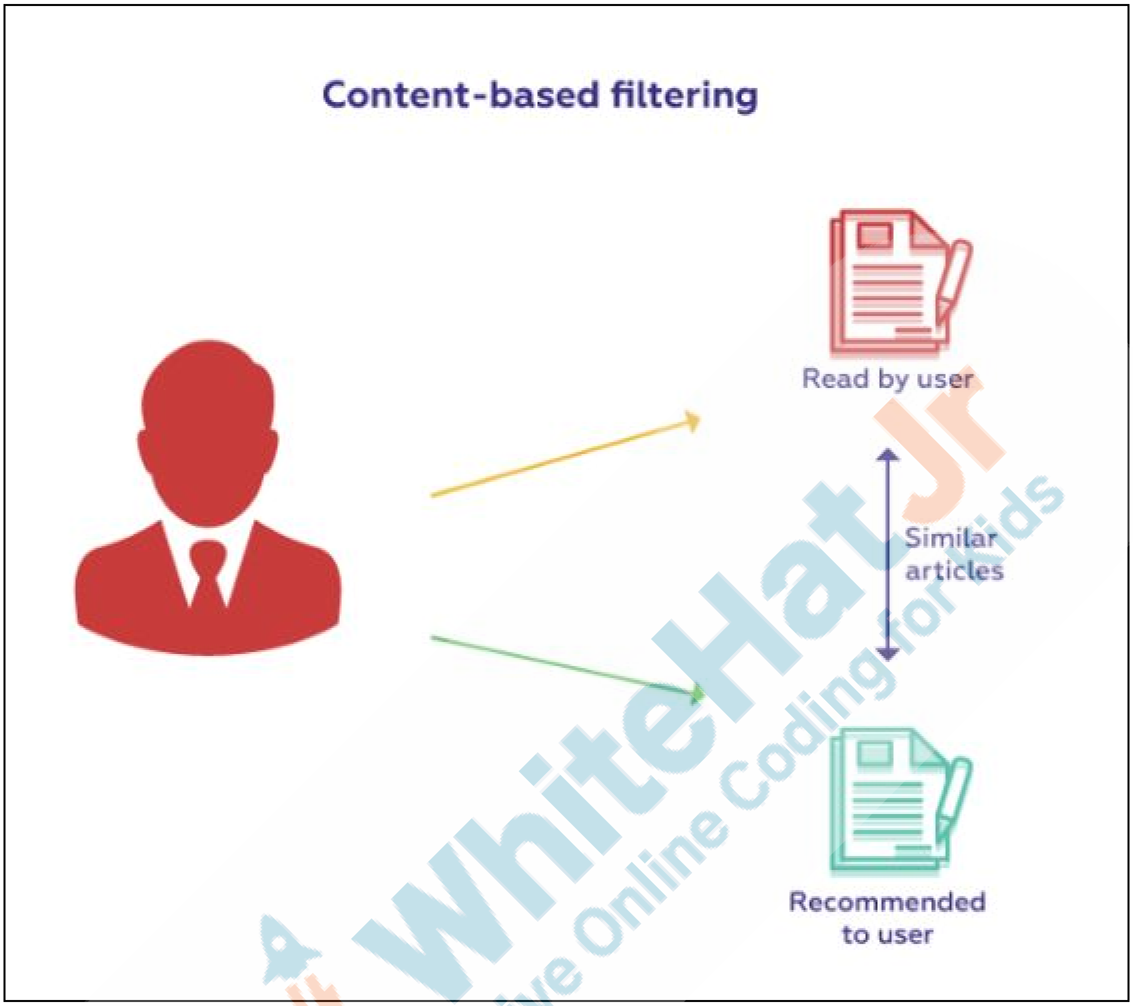
Topic	**Content Based Filtering**	
Class Description	**Students will be doing Content Based Filtering and understand the concept of cosine similarity.**	
Class	**C140**	
Class time	**45 mins**	
Goal	● Understand the concept of Cosine Similarity ● Perform content based filtering on the data	
Resources Required	● Teacher Resources ○ Google Colab ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ Google Colab ○ 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>! In the last class, we performed demographic filtering! Can you tell me how we did it and what kind of recommendation can we give with it?	**ESR:** We used IMDb's formula of weighted rating and calculated the score for all the movies.

		It can be used to give general recommendations!
	I have an exciting quiz question for you! Are you ready to answer this question? Teacher click on the  button on the bottom right corner of your 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.	
	Great! Now in today's class, we will be working on Content Based Filtering. Are you excited?	**ESR:** "Yes!"

	Let's get started!	
Teacher Initiates Screen Share		
<u>CHALLENGE</u> ● **Decompose "The World's Hardest Game"** ● **Ask the student to recall concepts which can be used to build the game**		
Step 2: Teacher-led Activity (15 min)	Do you remember what content-based filtering means?	**ESR:** The general idea behind content based filtering is that if a person likes a particular item, he or she will also like an item that is similar to it.
	Great! In our case, can you tell what all could act as the content of a movie?	**ESR:** ~ Overview ~ Cast ~ Crew ~ Keyword ~ Tagline ~ Genre

 		
	Now, let us proceed with features like **cast, crew, keywords and genres**. Here, we mean that if the cast, the crew, the keywords or the genre of Movie A and Movie B are similar (with least differences), and a user likes Movie A then we can recommend Movie B to the user!	
	Let us first see how these columns look like in our DataFrame.	

```
df2[['title', 'cast', 'crew',
'keywords', 'genres']].head(3)
```

Content Based Filtering

```
[22] df2[['title', 'cast', 'crew', 'keywords', 'genres']].head(3)
```

	title	cast	crew	keywords	genres
0	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...	[{"id": 1463, "name": "culture clash"}, {"id":...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...
1	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...
2	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...	[{"id": 470, "name": "spy"}, {"id": 818, "name...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...

Here, it looks like all the data is in a list of dictionaries.

However, there might be a few rows that may be in string format but are a list of dictionaries! To eliminate that factor and ensure all our rows are list of dictionaries, we can use a python's module **literal_eval()** which would safely check for us what datatype our data is meant to be and convert it into the same (only if it a string, otherwise no changes happen):

```
from ast import literal_eval

features = ['cast', 'crew',
'keywords', 'genres']
for feature in features:
    df2[feature] =
df2[feature].apply(literal_eval)

df2.dtypes
```

Here, we have listed down all the features that we want to evaluate,

	iterated over these features and then applied the **literal_eval()** function to all the values of the feature column in our DataFrame **df2.**	

```
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    df2[feature] = df2[feature].apply(literal_eval)

df2.dtypes
```

```
budget                   int64
genres                  object
homepage                object
id                       int64
keywords                object
original_language       object
original_title          object
overview                object
popularity             float64
production_companies    object
production_countries    object
release_date            object
revenue                  int64
runtime                float64
spoken_languages        object
status                  object
tagline                 object
title                   object
vote_average           float64
vote_count               int64
tittle                  object
cast                    object
crew                    object
dtype: object
```

	Okay, now we're good to go! We currently have all the data as a list of dictionaries, however it would be	

	extremely easy for us to have the data as a list! Data like Cast, Keywords and Genres should be as a list of elements instead of a list of dictionaries. Next, we also want to know the name of the director, which is available in the crew column! It would be great for us if we could have the names of the directors of these movies in a separate column in our dataframe.	
	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**	
	<u>ACTIVITY</u> • **Student codes to filter data and apply cosine similarity** • **Student finish the recommendation system**	
Step 3: Student-Led Activity (20 min)	*<Teacher would be required to help the student with coding all the elements of this analysis>*	

	Please write a function to find out the name of the director of the movie and store it in a separate DataFrame.
	
	Here, make sure that if you cannot find the name of the director, then store NaN value in the DataFrame. NaN in Pandas Dataframe is used to represent None values.
	
	`def get_director(x):`
	`    for i in x:`
	`        if i['job'] == 'Director':`
	`            return i['name']`
	`    return np.nan`
	
	`df2['director'] = df2['crew'].apply(get_director)`
	
	Here, we defined a function **get_director()** in which we are iterating over all the dictionaries that we have in the crew column of the movie. We are checking if the key **job** in any of the dictionaries matches with the word **Director** and if it does, we are returning the name of the director.
	
	If not, we are returning the **np.nan** value, which represents None in DataFrame. We placed this line outside the for loop so that if, and only if no value was returned earlier (none
		Student codes to filter the name of the directors and save it in a separate column.

	of the values satisfied the if condition) then we return NaN. Finally, we are applying this function on the **crew** column of our Dataframe, and saving the value it returns to a new column **director.**	

```
[21] def get_director(x):
         for i in x:
             if i['job'] == 'Director':
                 return i['name']
         return np.nan

     df2['director'] = df2['crew'].apply(get_director)
```

	Next, we want to convert the list of dictionaries in the columns **cast, keywords** and **genres** into simple lists.
	
	Now for this, make sure that you cross check if the value of the column is a list or not. Python has a function **isinstance()** to do the same.
	
	Write a function that can do so, and apply the function on these 3 columns.
	
	`def get_list(x):`
	`    if isinstance(x, list):`
	`        names = [i['name'] for i in x]`
	`        return names`
	`    return []`
		The student codes the function and applies it to dataframe columns.

```
features = ['cast', 'keywords',
'genres']
for feature in features:
   df2[feature] =
df2[feature].apply(get_list)
```

Here, we have a function named **get_list()** in which we are first checking if the value x is an instance of list or not using the **isinstance()** method. If it's not, we are returning an empty list since it was not already a list of dictionaries. If it is, we are finding all the **names** values in all dictionaries inside the list and storing them in a list. We are finally returning this list with all the names.

Lastly, we define the names of the columns on which we want to apply this function. We are then iterating over all the features/names of the columns and then using the **appy()** function to apply the function we created on all the columns!

```
[ ] def get_list(x):
        if isinstance(x, list):
            names = [i['name'] for i in x]
            return names
        return []

    features = ['cast', 'keywords', 'genres']
    for feature in features:
        df2[feature] = df2[feature].apply(get_list)
```

Cross check if we now have the **director** column and if other columns are converted into a list instead of a list of dictionaries.

```
df2[['title', 'cast',
'director', 'keywords',
'genres']].head(3)
```

Student codes to check the value.

```
[ ] df2[['title', 'cast', 'director', 'keywords', 'genres']].head(3)
```

	title	cast	director	keywords	genres
0	Avatar	[Sam Worthington, Zoe Saldana, Sigourney Weave...	James Cameron	[culture clash, future, space war, space colon...	[Action, Adventure, Fantasy, Science Fiction]
1	Pirates of the Caribbean: At World's End	[Johnny Depp, Orlando Bloom, Keira Knightley, ...	Gore Verbinski	[ocean, drug abuse, exotic island, east india ...	[Adventure, Fantasy, Action]
2	Spectre	[Daniel Craig, Christoph Waltz, Léa Seydoux, R...	Sam Mendes	[spy, based on novel, secret agent, sequel, mi...	[Action, Adventure, Crime]

Great! Now let's think a little. There might be multiple actors with the same name. Is our computer smart enough to find the difference between **Johnny** with a Capital J and **johnny** with a small j?

Also, as we move forward, we can create a string that contains all the metadata of a movie (info about keywords, actors, director and genres) and compare these strings to find similarity between them. The more similar two strings are, the more chances we have that the user will like that movie. That's how content based filtering works and to find this similarity, we will be using the cosine similarity method!

Student codes the function to convert data to lowercase and remove spaces. Then apply it to dataframe columns.

	Now, let's first write a code where we are converting all the values in columns **cast, keywords, director** and **genres** to lowercase. Let's also remove the spaces from the elements of these lists since we are going to create a metadata string for all the movies.
	
	Please note that while the columns **cast, keywords** and **genres** are a list of elements, **director** is a string. We need to handle these differently.
	
	`def clean_data(x):`
	`    if isinstance(x, list):`
	`        return [str.lower(i.replace(" ", "")) for i in x]`
	`    else:`
	`        if isinstance(x, str):`
	`            return str.lower(x.replace(" ", ""))`
	`        else:`
	`            return ''`
	
	`features = ['cast', 'keywords', 'director', 'genres']`
	`for feature in features:`
	`    df2[feature] = df2[feature].apply(clean_data)`
	
	Here, we have defined a function **clean_data().** Here, we are first checking if the value is a list. If it is, we are converting all the values to	

	lowercase using the **lower()** function and replacing the “ ” with “”. Since **lower()** is a method that can be applied on Strings, we are using it with **str**, which is a default Python’s data type for strings. If, however, our item was not a list, we are then checking if it is a string. Remember that we saved **NaN** values where we didn’t find the name of the director, hence, this is necessary to check. If it is a string, we are again converting it to lowercase and replacing “ ” with “”. If it was not a string (and was **NaN** since that is the only value it can be), we are returning an empty string. We are finally applying this function to the columns **cast, keywords, director** and **genres**.	

```
[ ] def clean_data(x):
        if isinstance(x, list):
            return [str.lower(i.replace(" ", "")) for i in x]
        else:
            if isinstance(x, str):
                return str.lower(x.replace(" ", ""))
            else:
                return ''

    features = ['cast', 'keywords', 'director', 'genres']
    for feature in features:
        df2[feature] = df2[feature].apply(clean_data)
```

	Now, we create the string metadata. Let’s explore a new method before we do this! To convert a list of elements into a string, we have **join()** method. What the join method does is that it will take all the elements of the list, join them together and then return it in string format.
	
	For example:
	
	`a = ["hello,", "how", "are", "you?"]`
	`" ".join(a)`
	
	Gives:
	hello, how are you?
	
	Here, it joined all the elements with a space “ ”	

	`a = ["hello,", "how", "are", "you?"]`
	`" ".join(a)`
	`'hello, how are you?'`	

	Okay, now let’s create the string of metadata. We want to separate all the elements of **keywords, cast** and **genres** with space and we want to keep space between these column values as well.
	
	`def create_soup(x):`
	`    return ' '.join(x['keywords']) + ' ' + '`
		Student codes to create the string of metadata and store it in a new column of our dataframe.

<table>
<tr>
<td></td>
<td>
<pre>'.join(x['cast']) + ' ' +
x['director'] + ' ' + '
'.join(x['genres'])
df2['soup'] =
df2.apply(create_soup, axis=1)</pre>
Here, we are first joining the keywords, then the cast, then adding the name of the director to it and then joining the genres in create_soup() function. This will give us a string of all the metadata about a movie.
Finally, we are applying the function create_soup on our dataframe. Here, since we are not applying the function to a particular column but to an entire row of dataframe, we are giving an additional attribute axis=1 to our apply() function, which means that we want the returning value to be treated along a column soup, instead of considering it as the index.
</td>
<td></td>
</tr>
<tr>
<td colspan="3">
<pre>[ ] def create_soup(x):
      return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
    df2['soup'] = df2.apply(create_soup, axis=1)</pre>
</td>
</tr>
<tr>
<td></td>
<td>
<Teacher takes over from here>
Now, we are all set to proceed with the final steps. First off, we want to remove the stop words. Stop words are the words that do not add any value to a given sentence but are only there to make it grammatically
</td>
<td>Student observes.</td>
</tr>
</table>

correct. Words like **The, And, But, etc.** are all stop words.

Secondly, it is easier for computers to compare two arrays. We want to create an array that will count all the words in our metadata string and maintain a count for each of the words. This will help us find similarity between two movies!

For this, we will use the **CountVectorizer** method from sklearn's library.

```
from
sklearn.feature_extraction.text
import CountVectorizer
count =
CountVectorizer(stop_words='engl
ish')
count_matrix =
count.fit_transform(df2['soup'])
```

Here, we are first importing the CountVectorizer. We are then counting all the words after removing the stop words and then we are converting it into a matrix, or a 3d array (list of lists).

```
[ ]  from sklearn.feature_extraction.text import CountVectorizer
     count = CountVectorizer(stop_words='english')
     count_matrix = count.fit_transform(df2['soup'])
```

<table>
<tr>
<td></td>
<td>We are then finally going to import the cosine_similarity function from sklearn and create a classifier based on our data with it.

<pre>from sklearn.metrics.pairwise
import cosine_similarity
cosine_sim2 =
cosine_similarity(count_matrix,
count_matrix)</pre></td>
<td>Student observes.</td>
</tr>
<tr>
<td colspan="3"><pre>[ ] from sklearn.metrics.pairwise import cosine_similarity
    cosine_sim2 = cosine_similarity(count_matrix, count_matrix)</pre></td>
</tr>
<tr>
<td></td>
<td>Next, we want to change the index of our movie data to the name of the movies.

<pre>df2 = df2.reset_index()
indices = pd.Series(df2.index,
index=df2['title'])</pre>
Here, we are resetting our data for df2 and then we are changing the index to the title of the movie.</td>
<td>Student observes.</td>
</tr>
<tr>
<td colspan="3"><pre>df2 = df2.reset_index()
indices = pd.Series(df2.index, index=df2['title'])</pre></td>
</tr>
</table>

	Finally, we will create the function that will get recommendations for us using our cosine_similarity classifier that we created earlier.
	
	`def get_recommendations(title, cosine_sim):`
	`    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 df2['title'].iloc[movie_indices]`
	
	Here, we are passing the title of the movie that the user likes and our classifier. We are then finding the index of the movie in our dataframe using the **indices** variable we created earlier, which contains the indexes of all the movies in the dataframe. We created this when we changed the index of our dataframe to the title of the movie.
	
	Next, we are creating a list of all the scores of the movies. This is the score of similarity of each movie with what the user likes. We are then using the sorted function on our data to sort the scores of all the movies
		Student observes.

and we are reversing its order with **reverse=True** attribute.

We are then taking elements from 1:11. We are not starting with 0 since the movie that the user likes will have the highest score (perfect score). We are then taking out the indexes of all the movies that we want to recommend and finally we are returning the titles of all the movies that our system recommends!

```
[37] def get_recommendations(title, cosine_sim):
         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 df2['title'].iloc[movie_indices]
```

Let’s test it!

```
get_recommendations('Fight Club', cosine_sim2)
```

```
get_recommendations('The Shawshank Redemption', cosine_sim2)
```

```
get_recommendations('The Godfather', cosine_sim2)
```

Let’s see what results we get!

```
[36] get_recommendations('Fight Club', cosine_sim2)

[1553, 946, 421, 4564, 45, 4462, 3863, 3043, 1010, 4101]
1553                     Se7en
946                   The Game
421                     Zodiac
4564    Straight Out of Brooklyn
45                 World War Z
4462        The Young Unknowns
3863                    August
3043           End of the Spear
1010                Panic Room
4101              Full Frontal
Name: title, dtype: object
```

```
get_recommendations('The Shawshank Redemption', cosine_sim2)

4638    Amidst the Devil's Wings
690               The Green Mile
4408              Jimmy and Judy
1247             City By The Sea
4502               Water & Power
4529           Hurricane Streets
559                 The Majestic
1752              Kiss the Girls
2818                     Witness
4564    Straight Out of Brooklyn
Name: title, dtype: object
```

```
[ ] get_recommendations('The Godfather', cosine_sim2)

2731      The Godfather: Part II
867      The Godfather: Part III
4638    Amidst the Devil's Wings
4209            The Conversation
3293                 10th & Wolf
2255                   The Yards
1394               Donnie Brasco
3012                The Outsiders
```

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)**	So, in this project class, we built our own Content Based Recommendation System! Congratulations! How was your experience?	**ESR:** varied
	Amazing. Now in the next class, we will be starting out with building our mobile app for a movie recommender to the user but for that, we need to first build an API! We will be using Flask for that.	-
Teacher Clicks 		

Activity	Activity Name	Links
Teacher Activity 1	Solution	https://colab.research.google.com/drive/1KrkLSusDnztkxwAb64SG7SNw798xvzeC?usp=sharing