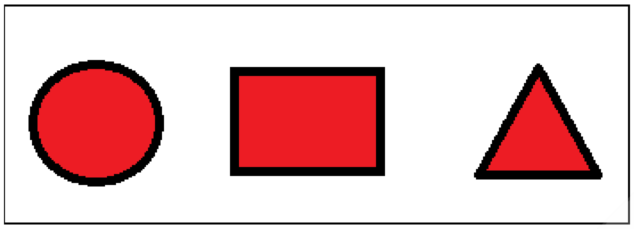
Topic	CONTENT-BASED FILTERING	
Class Description	**The student will do Content-Based Filtering and understand the concept of cosine similarity.**	
Class	**PRO C140**	
Class time	**45 mins**	
Goal	● Understand the concept of Cosine Similarity ● Perform content-based filtering on the data	
Resources Required	● 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**	**05 mins** **15 mins** **20 mins** **05 mins**

WARM-UP SESSION - 5 mins

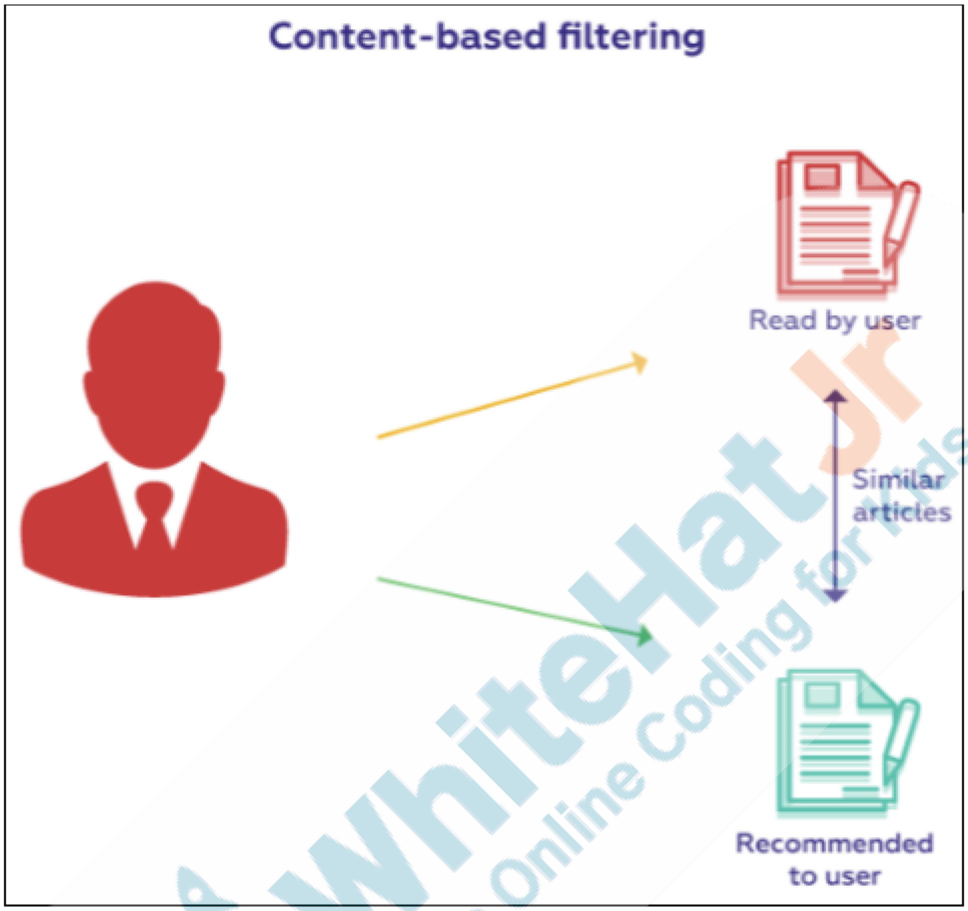
Teacher Starts Slideshow
Slide # to #
<**Note**: Only Applicable for Classes with VA>
Refer to speaker notes and follow the instructions on each slide.

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**
What do you understand by the term **content-based filtering?** ***Note:** Encourage the student to give answers and connect the answer with today's topic.* Great, a **content-based filtering system** works on the principle of **'ITEM SIMILARITY' or 'CONTENT**	**ESR:** Varied.

SIMILARITY', which means that if a person likes a particular item, he or she will also like an item that is similar to it. Also, can you tell me what we mean by the term **similarity parameter**? ***Note:*** *Let the student try and answer.* Great, in simple words, a **similarity parameter** is what makes objects similar? To understand it better, let's do a simple task. Can you tell me the **similarity parameter** in both the oranges? ***Note:*** *Let the student try and answer.* Great, they both are similar because they have common similarity parameters, **color and shape**.	 **ESR:** Varied. **ESR:** Sure. **ESR:** Varied.

Can you tell if the objects in the below given image are similar or not?  ***Note:*** *Let the student try and answer.* Great, they all are similar because they have common similarity parameter, **color,** but at the same time, they are dissimilar if you consider their **shape**. So we can say that, it is important to look for **similarity parameters,** if you are trying to classify items as similar or dissimilar. Great! Can you tell what all could be used as the **similarity parameters** to compare movies? ***Note:*** *Let the student try and answer.* If you closely look into your dataset, we have columns like, **Overview, Cast, Crew, Genres, Keywords, etc**. which can be used to find similarity among movies. For example, if one likes action movies (**Genres = Action**) like Iron man, it is certain that he or she will like other movies of the same genre **(Genre = Action)**, like **The Dark Knight, Avengers,** etc.	**ESR:** Varied. **ESR:** Varied.

WhiteHat Jr
Live online Coding for Kids



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's understand it with the help of an example! Consider the table below and answer the following question?	**ESR:** Sure.

If a person likes **Iron Man**, whose actor is Robert Downey Jr. Which movie will you recommend him next? *Note: Let the student try and answer!* Great, we will recommend **Iron man 2** first, because it is most similar (has least differences) to what the user likes.	**ESR:** Varied.

Movie	Actor	Keywords	Genres
Mission Impossible	Tom Cruise	Mission, ethan, country, MI6…	Action
Iron Man 2	Robert Downey Jr.	Technology, Suit, Iron man….	Action

Great! Now we are ready to start coding!

Teacher Ends Slideshow

TEACHER-LED ACTIVITY - 15 mins

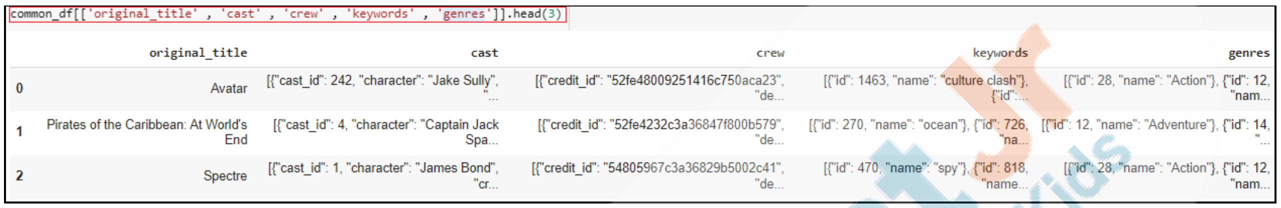
Teacher Initiates Screen Share

ACTIVITY

- **Converting all the data in a list format.**
- **Make students understand how to extract data from a DataFrame.**

Teacher Action	Student Action
To apply a similar approach in our dataset, let's first see how these columns look like in our DataFrame.	

For that, open this Teacher Activity link : Boilerplate , and in a new cell, write the command, **common_df[[‘original_title’ , ‘cast’ , ‘crew’ , ‘keywords’ , ‘genres’]].head(3),** which will print the top 3 rows of these columns.



```
common_df[['original_title' , 'cast' , 'crew' , 'keywords' , 'genres']].head(3)

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

Can you tell me the type of data in all of these columns?

ESR: Varied.

Note: Let the student try and answer!

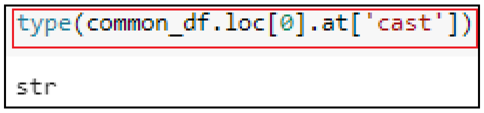
It looks like all the data is in the form of a list **of dictionaries**, but just to be sure let’s check the datatype of the first element under the **cast** column. To do so, let’s access the first row **(indexed as 0)** of the DataFrame using the command, **common_df.loc[0]** and to get to the first element of the **‘cast’ column,** let’s use the **‘at’** operator as, **common_df.loc[0].at[‘cast’]**.

To get the datatype for the first element under the cast column, use the **type()** method as, **type(common_df.loc[0].at[‘cast’])**

Can you tell the datatype?

ESR: It is in string format.

Yes, the data appears to be in the form of a **list of dictionaries**, but it is actually in the form of a **string**.



```
type(common_df.loc[0].at['cast'])

str
```

For easy processing, let's ensure that all the rows are in the form of a **list of dictionaries** only. To do so, 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

Note: Our data would be changed only if it is a string, otherwise no changes happen.

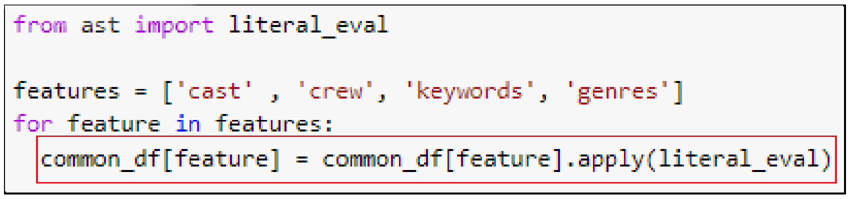
We need to perform this operation for the **cast, crew, keywords, and genre** columns.

To do so, import the **literal_eval** function from the **ast** module, using the command, **from ast import literal_eval**.

Next, make a list of features or columns in which we want to apply this function, using the command,
features = ['cast' , 'crew' , 'keywords' , 'genres']

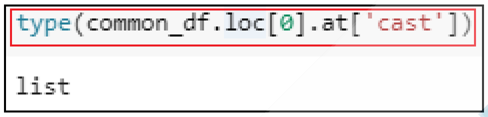
Now, let's iterate over these features using a **for** loop, and apply the **literal_eval** function on these **feature columns**, using the **.apply() method,** as

common_df[feature] = common_df[feature].apply(literal_eval)

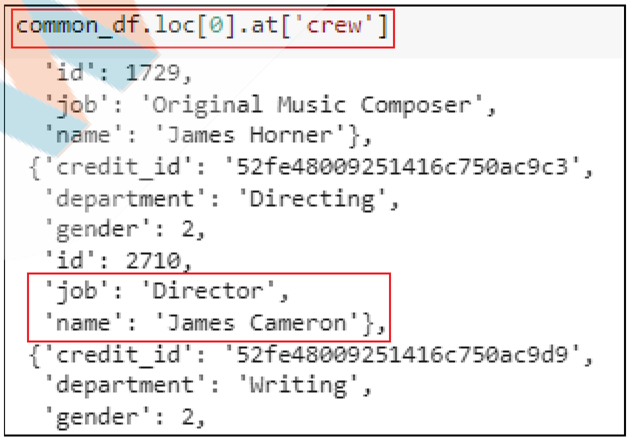


```
from ast import literal_eval

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

Once done, let's verify the datatype of the first element under the **cast** column, using the command, **type(common_df.loc[0].at['cast'])** Can you tell the datatype now? Great! We are good to go!	 **ESR:** It's a list.


`type(common_df.loc[0].at['cast'])`
 	
`list`	

Now, let's print the first element from the crew column and observe the data carefully. To do so use the command, **common_df.loc[0].at['crew']** Using this command, you will get details for all the crew member details, in the form of a list of dictionaries. From this chunk of information we want to know the name of the director. It would be useful for us later, if we could have the names of the directors of these movies in a separate column in our DataFrame.	



```
common_df.loc[0].at['crew']
  'id': 1729,
  'job': 'Original Music Composer',
  'name': 'James Horner'},
 {'credit_id': '52fe48009251416c750ac9c3',
  'department': 'Directing',
  'gender': 2,
  'id': 2710,
  'job': 'Director',
  'name': 'James Cameron'},
 {'credit_id': '52fe48009251416c750ac9d9',
  'department': 'Writing',
  'gender': 2,
```

Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
Teacher Starts Slideshow **Slide # to #** <**Note**: Only Applicable for Classes with VA> Refer to speaker notes and follow the instructions on each slide.	
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 filter data and apply cosine similarity.** • **Student finish the recommendation system**	
Teacher Action	**Student Action**
Open the Student Activity 1 : Boilerplate .	

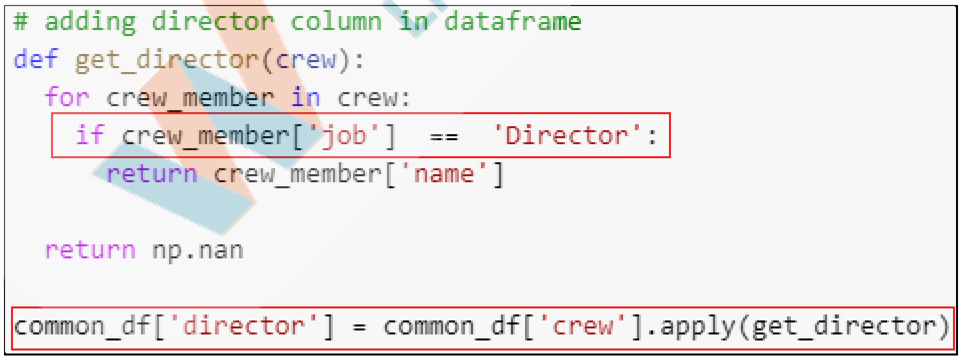
To extract the director name from the crew column, let's create a method and name it as **get_director(crew),** which will accept one argument **'crew',** which is the column value (list of dictionaries).

Note: 'crew' column is different from 'crew' variable which is an argument to get_director function.

The **get_director(crew)** method will be iterating over all the dictionaries that we have in the crew column of the movie and will check 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.

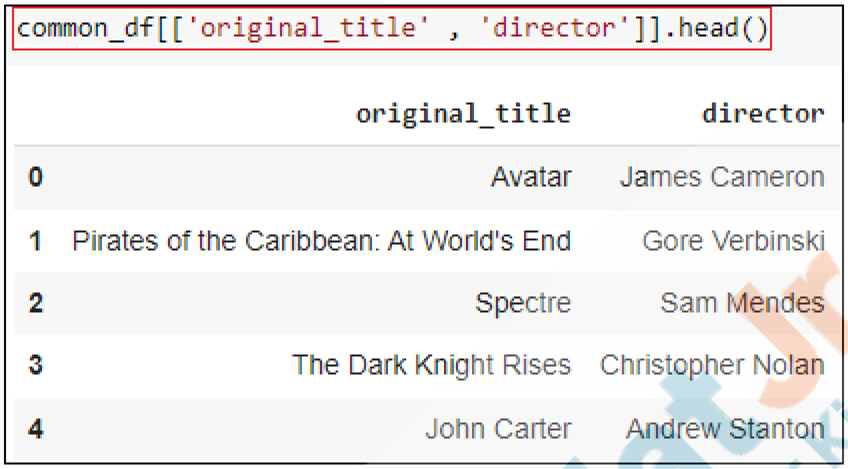
If not, we are returning the **np.nan** value, which represents '**not a number'**. We placed this line outside the **for** loop so that if, and only if no value was returned earlier (none of the values satisfied the **if** condition) then we return **NaN**.

Finally, we are applying this function on the **crew** column of our DataFrame, and saving the value it returns to a new column **director**.



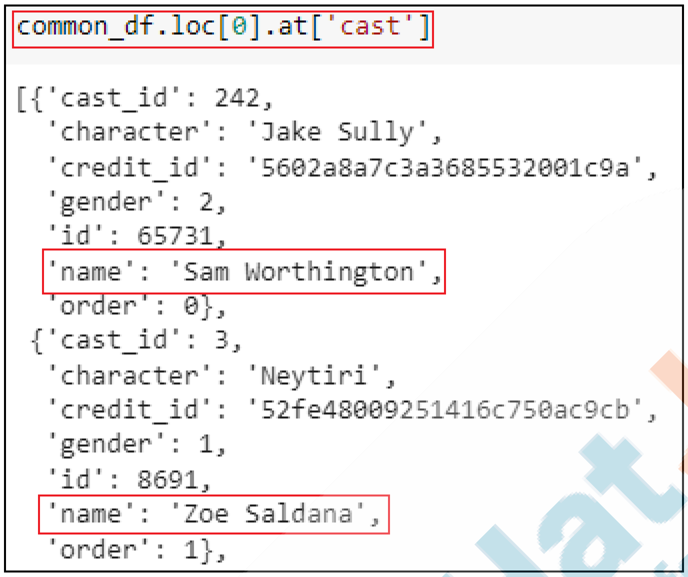
```
# adding director column in dataframe
def get_director(crew):
  for crew_member in crew:
    if crew_member['job']  ==  'Director':
      return crew_member['name']

  return np.nan

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

To verify, let's print the first 5 rows for **original_title** and **director** columns, using the **.head()** method as,

common_df[['original_title' , 'director']].head()	
	
Now, let's print the first element from the **cast** column so that we can look for the important information which will be useful for proper analysis. To do so, use the command, **common_df.loc[0].at['cast']** If we observe the data carefully, we have the names of all the casting members stored in the form of a dictionary with the key name as, **'name'**.	

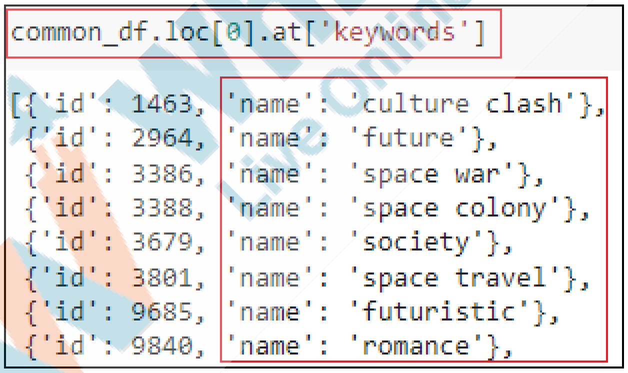
```
common_df[['original_title' , 'director']].head()
```

```
                           original_title            director
0                                  Avatar       James Cameron
1  Pirates of the Caribbean: At World's End     Gore Verbinski
2                                 Spectre          Sam Mendes
3                   The Dark Knight Rises   Christopher Nolan
4                              John Carter      Andrew Stanton
```



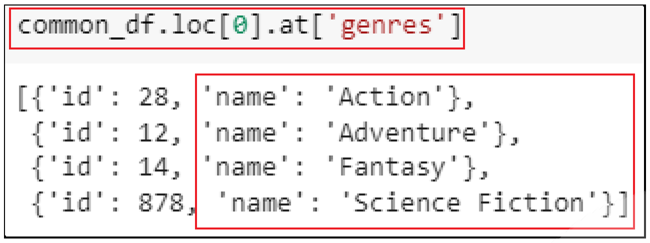
```
common_df.loc[0].at['cast']

[{'cast_id': 242,
  'character': 'Jake Sully',
  'credit_id': '5602a8a7c3a3685532001c9a',
  'gender': 2,
  'id': 65731,
  'name': 'Sam Worthington',
  'order': 0},
 {'cast_id': 3,
  'character': 'Neytiri',
  'credit_id': '52fe48009251416c750ac9cb',
  'gender': 1,
  'id': 8691,
  'name': 'Zoe Saldana',
  'order': 1},
```

If we perform the same process for the **keywords** column, we will find that all the keywords which will useful in analysis, are stored in the form of a dictionary with the key name as **'name'**.



```
common_df.loc[0].at['keywords']

[{'id': 1463, 'name': 'culture clash'},
 {'id': 2964, 'name': 'future'},
 {'id': 3386, 'name': 'space war'},
 {'id': 3388, 'name': 'space colony'},
 {'id': 3679, 'name': 'society'},
 {'id': 3801, 'name': 'space travel'},
 {'id': 9685, 'name': 'futuristic'},
 {'id': 9840, 'name': 'romance'},
```

Same goes for **genres** column as well.



```
common_df.loc[0].at['genres']

[{'id': 28, 'name': 'Action'},
 {'id': 12, 'name': 'Adventure'},
 {'id': 14, 'name': 'Fantasy'},
 {'id': 878, 'name': 'Science Fiction'}]
```

To extract all the values from cast, keywords and genres columns, which are associated with the **'name'** key, let's write a simple method and name it as **get_name_list(column_value),** which will accept **column_value** as an argument,

def get_name_list(column_value):

Next, create an empty list variable and name it as
name_list = []

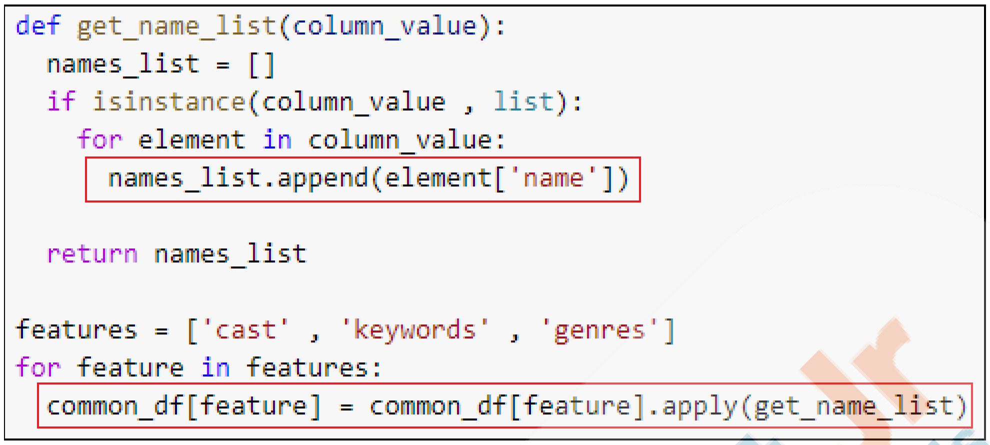
After that, first check, if the column value received is a list or not, using the **.isinstance()** method.

If the column value is a list, iterate through it using a for loop, and add all the values to the names_list, which are associated with the key **'name'**, using the **.append()** list method,

```
If isinstance(column_value , list):
for element in column_value:
    names_list.append(element['name'])
```

Finally return the names_list,
return names_list

Apply this method on **cast, keywords and genres** columns using the **.apply()** method, so that their value gets updated from a **list of dictionaries to a list of names**.

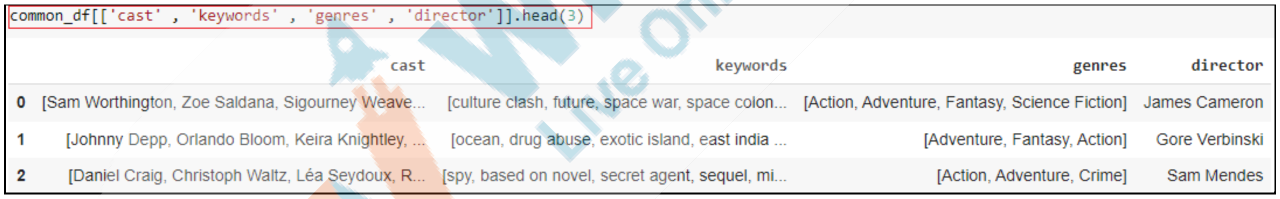


```
def get_name_list(column_value):
  names_list = []
  if isinstance(column_value , list):
    for element in column_value:
      names_list.append(element['name'])

  return names_list

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

To verify the changes in these columns, let's print the top 3 rows of these columns using the **.head(3)** method, as

common_df[['cast' , 'keywords' , 'genres' , 'director']].head(3)

We can clearly see that, from a big chunk of information, our columns only hold the **list of names** and the director columns hold string data.



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

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

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?

To remove duplicates and preprocess the data easily, let's remove spaces and convert all the data in lowercase, for **cast, keywords, genres and director** columns.

ESR: No.

To do so, let’s create a method and name it as **clean_data(column_value),** which will accept **column_value** as an argument,

```
def clean_data(column_value):
```

Next, create an empty list and an empty string variable and name it as **modified_list = []** and **modified_string = “”**

After that, let’s check if the column value received is a list or a string, using the **.isinstance()** method.

If the column value is a list, iterate through all the **string elements** using a for loop, and remove all the spaces using the **.replace()** string method.
Once all the spaces are removed, convert it to lowercase string, using the **.lower()** method.
Finally append it to the **modified_list** variable and return the list.

```
If isinstance(column_value , list):
for element in column_value:
   modified_string = element.replace(“ ” , “”)
   modified_list.append(modified_string.lower())
return modified_list
```

If the column value is a string, remove all the spaces using the **.replace()** string method and convert it to lowercase string, using the **.lower()** method.
Finally return the string.

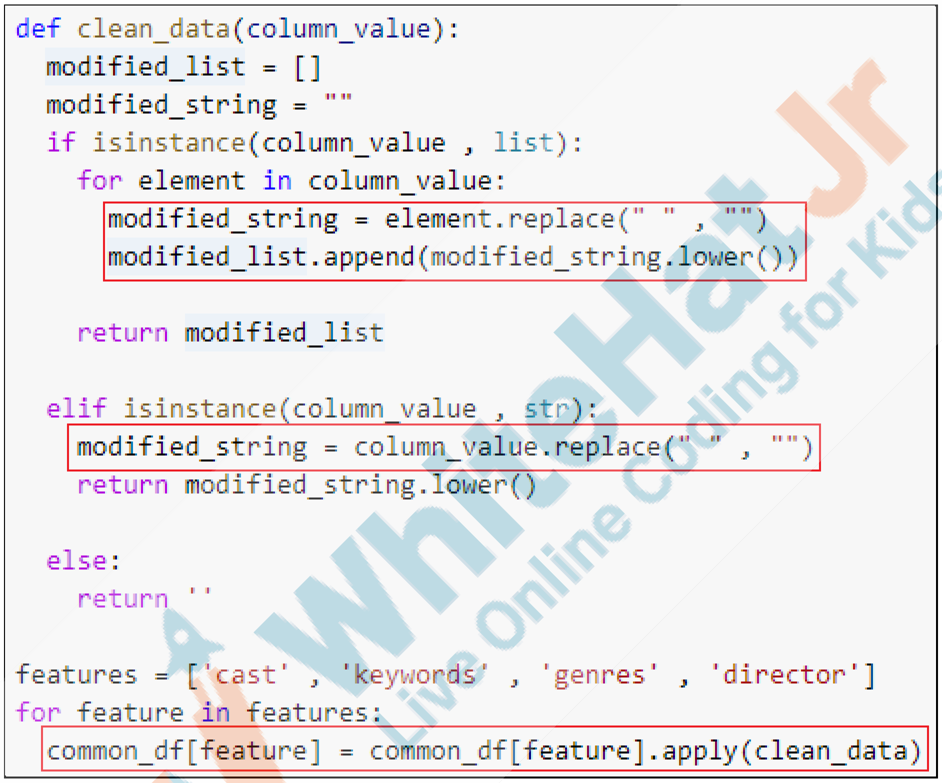
```
elif isinstance(column_value , str):
   modified_string = column_value.replace(“ ” , “”)
   return modified_string.lower()
```

If the **column_value** is neither list, nor string, return an empty string.

```
else:
```

return “”

Apply this method on **cast, keywords, genres and director** columns using the **.apply()** method, so that their value gets updated.



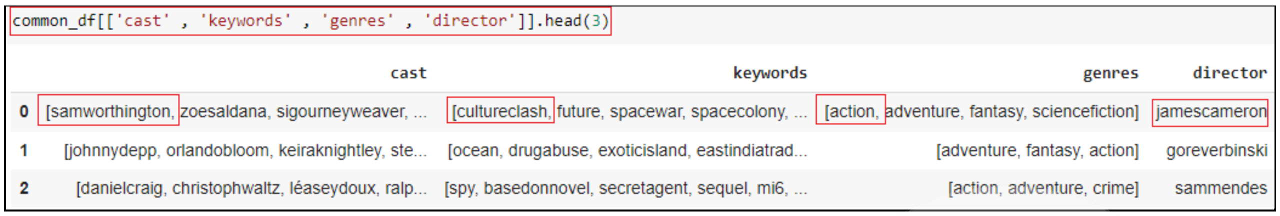
```
def clean_data(column_value):
  modified_list = []
  modified_string = ""
  if isinstance(column_value , list):
    for element in column_value:
      modified_string = element.replace(" " , "")
      modified_list.append(modified_string.lower())

    return modified_list

  elif isinstance(column_value , str):
    modified_string = column_value.replace(" " , "")
    return modified_string.lower()

  else:
    return ''

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

To verify the changes in these columns, let’s print the top 3 rows of these columns using the **.head(3)** method, as

common_df[[‘cast’ , ‘keywords’ , ‘genres’ , ‘director’]].head(3)

We can clearly see that all the spaces are removed from our data and it is converted to lowercase.

Great, our data is cleaned now!



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

	cast	keywords	genres	director
0	[samworthington, zoesaldana, sigourneyweaver, ...	[cultureclash, future, spacewar, spacecolony, ...	[action, adventure, fantasy, sciencefiction]	jamescameron
1	[johnnydepp, orlandobloom, keiraknightley, ste...	[ocean, drugabuse, exoticisland, eastindiatrad...	[adventure, fantasy, action]	goreverbinski
2	[danielcraig, christophwaltz, léaseydoux, ralp...	[spy, basedonnovel, secretagent, sequel, mi6, ...	[action, adventure, crime]	sammendes

After cleaning your data, let's create a string that contains all the **metadata** of a movie (information about keywords, actors, director and genres) and compare these strings to find similarity between them.

To create this string metadata we will use the **.join()** method, which will take all the elements of a list and convert them into a string.

For example, let's assume we have a list,
b = ['hello,' , 'how' , 'are' , 'you' , '?']

We want to join the elements of this list with a space in between, so we will use the command,

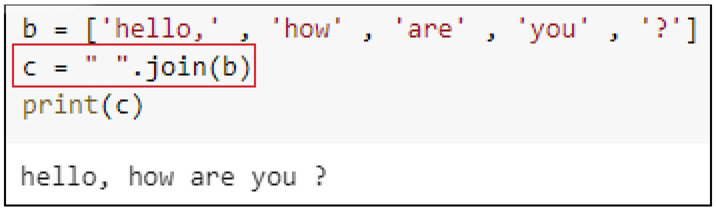
c = " ".join(b)

Can you predict the output?

Note: Let the student try and answer!

The output will be like,
hello, how are you ?

ESR: Varied.



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

hello, how are you ?
```

Now that we know how the join method works, let’s extract information from **keywords, cast, director, and genres columns** for each movie, and create a **common string** or **soup** for all the movies.

For that, let’s create a method named **create_soup(x)** which will,

- Accept a row of the DataFrame as an argument
- Extract the values associated with the **keywords, cast, director** and **genres** columns of that row.
- **Join** all the extracted information with a **space** in between them.
- **Return** the final string.

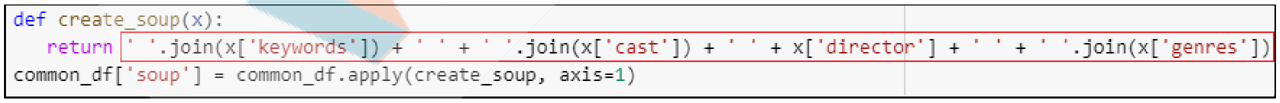
The code for this method would look like,

def create_soup(x):
return ‘ ‘.join(x[‘keywords’]) + ‘ ‘ + ‘ ‘.join(x[‘cast’]) + ‘ ‘ + x[‘director’] + ‘ ‘ + ‘ ‘.join(x[‘genres’])

Let’s apply this method on our DataFrame and add a new column named **soup** for each movie as,

common_df[‘soup’] = common_df.apply(create_soup , axis = 1)

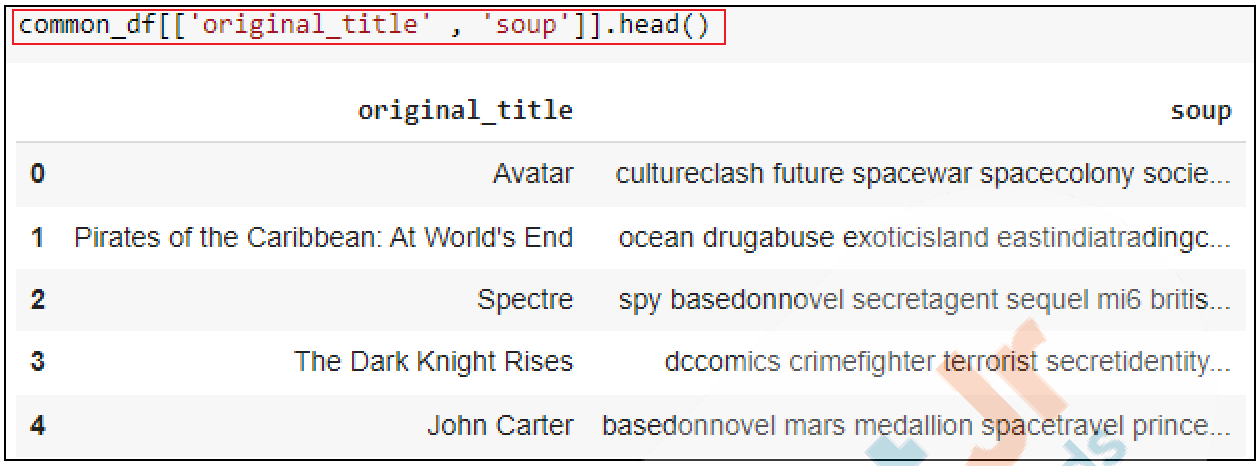
Here **axis = 1**, means that we want the returning value to be treated along a column soup.



```
def create_soup(x):
    return ' '.join(x['keywords']) + ' ' + ' '.join(x['cast']) + ' ' + x['director'] + ' ' + ' '.join(x['genres'])
common_df['soup'] = common_df.apply(create_soup, axis=1)
```

To verify the output, let’s print the **original_title** and soup column, using the **.head()** method, as

common_df[[‘original_title’ , ‘soup’]].head()



```
common_df[['original_title' , 'soup']].head()
```

```
                         original_title                                                soup
0                                Avatar    cultureclash future spacewar spacecolony socie...
1  Pirates of the Caribbean: At World's End    ocean drugabuse exoticisland eastindiatradingc...
2                               Spectre    spy basedonnovel secretagent sequel mi6 britis...
3                 The Dark Knight Rises        dccomics crimefighter terrorist secretidentity...
4                           John Carter    basedonnovel mars medallion spacetravel prince...
```

Now, we have the **soup** string ready for each movie. Remember, the more similar the **soup** strings, the more will be the similarity between the movies! Can you tell how we can check for similarity between two sentences? *Note: Let the student try and answer!*	**ESR:** Varied.

Let's try to understand the process of calculating similarity, with the help of an example. Consider the three sentences given in the table below.

Can you tell which sentence is more similar to **sentence 1**, sentence 2, or sentence 3?

Sno	Sentence
1	Yonex produces the strongest badminton racquets in the world. Yonex had a profit of 1 million dollars from badminton racquets this year. Their competitor Lining, marked a profit of 1.5 million dollars from badminton racquets.
2	Yonex has a monopoly in producing flexible tennis racquets all over the world. Their competitor, marked a profit of 1.5 million dollars from tennis racquets.

3	Lining makes the best badminton racquets in the world. Lining had a profit of 1.5 million dollars, while yonex had a profit of 1 million dollars this year.

To calculate similarity between the sentences,

a) First let's convert all the sentences into lowercase and remove all the **stop words** or the unnecessary words which do not add any meaning to the sentence. For example, a, an, is, am, the, etc.

Sno	**Sentence (Stop words removed)**
1	yonex produce strong badminton racquet world yonex profit 1 million dollar badminton racquet year competitor lining mark profit 1.5 million dollar badminton racquet
2	yonex monopoly produce flexible tennis racquet world yonex profit 5 million dollar tennis racquet competitor lining mark profit 1.5 million dollar tennis racquet
3	lining produce best badminton racquet world lining profit 1.5 million dollar yonex profit 1 million dollar year

b) Let's calculate the word count for each of the words in all the 3 sentences. This process is called **Vectorizing** the sentence.

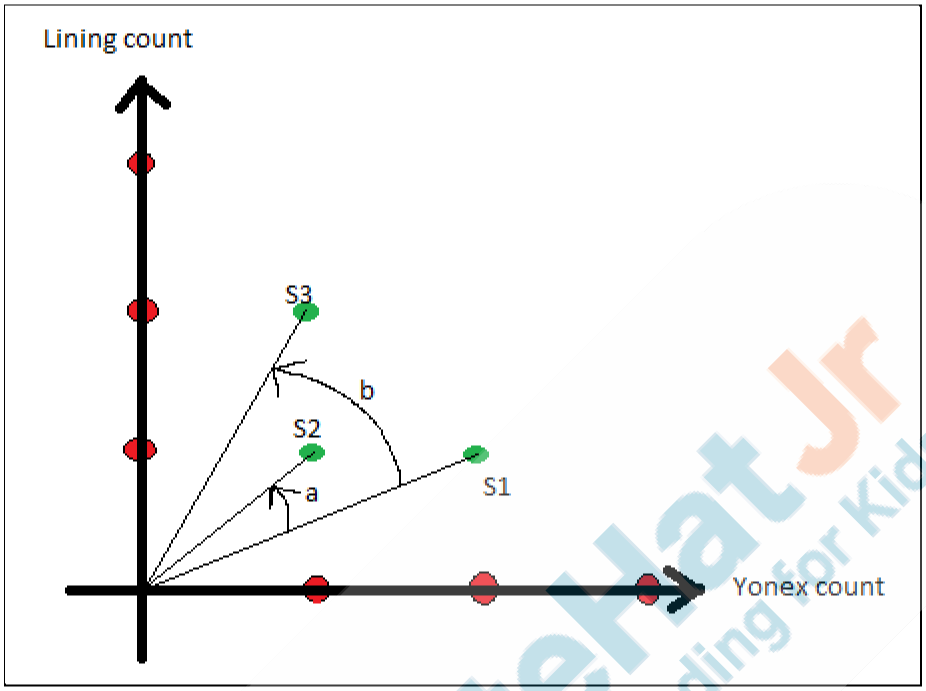
Words	Sentence 1	Sentence 2	Sentence 3
yonex	2	1	1
produce	1	1	1
strong	1		
badminton	2		1
racquet	3	3	1
world	1	1	1

competitor	1	1	
profit	2	2	2
million	2	2	2
dollar	2	2	2
year	1		1
lining	1	1	2
mark	1	1	
monopoly		1	
flexible		1	
tennis		3	
best			1

c) To understand the similarity calculation process, let's consider only 2 words for now, as we can easily visualize 2 dimensional vectors.

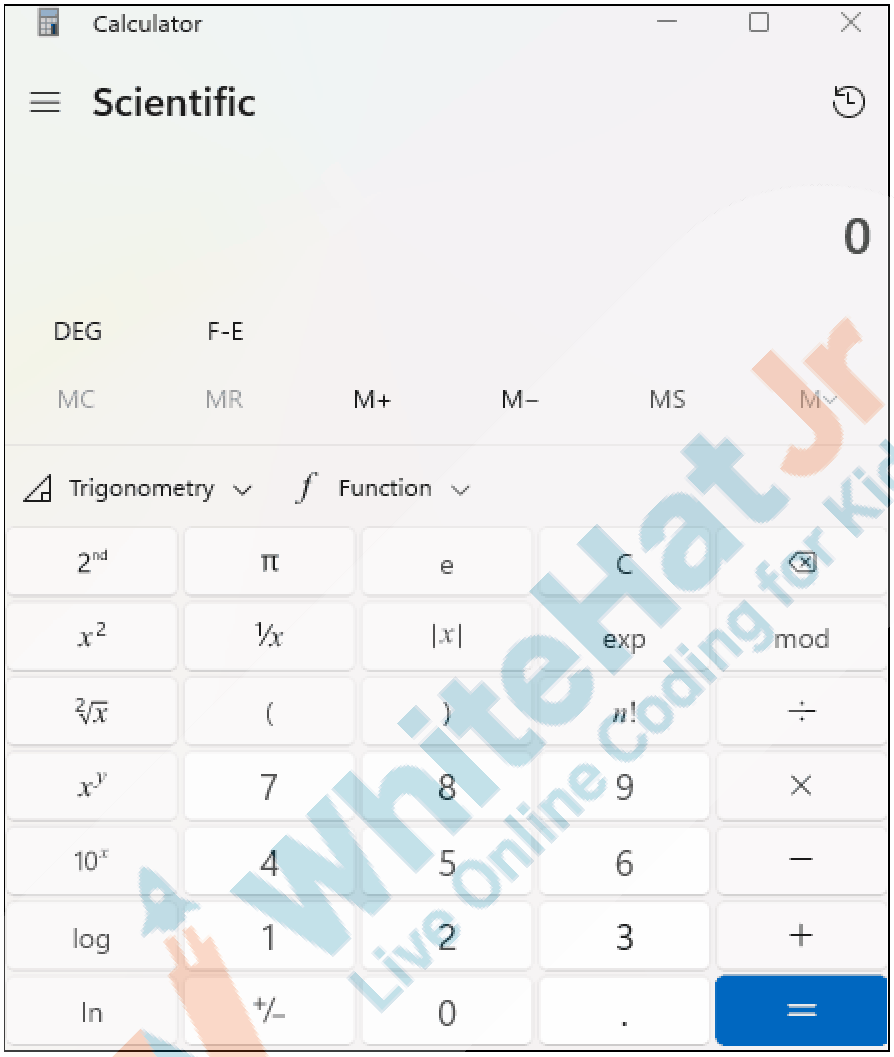
Words	**Sentence 1**	**Sentence 2**	**Sentence 3**
Yonex	2	1	1
Lining	1	1	2

d) Let's plot these sentences on a graph or create their **vectors** and calculate the angles between them.



e) Finally we will use a technique known as **'Cosine Similarity'** to calculate which sentence is more similar to sentence 1.

- We can clearly see that the **angle 'a'** is the angle between the vectors of sentence 1 and sentence 2, whereas the **angle 'b'** is the angle between the vectors of sentence 1 and sentence 3.
- Also, we can clearly see, **angle a is less than angle b (angle a < angle b).**
- Let's assume **angle a = 18 degrees** and **angle b = 36 degrees (angle a < angle b).**
- Let's calculate the cosine of these angles using the scientific calculator in your laptop or computer.



- Cosine of angle a, **cosine(18) = 0.95 or 95 percent.**
- Cosine of angle b, **cosine(36) = 0.80 or 80 percent.**

f) So now we can say that **sentence 2** is **more similar** to **sentence 1.**

We will use the same principle of **‘cosine similarity’** to calculate similarity between our movies. Can we do it manually, just like the way we did it above?	**ESR:** No.

Note: Let the student try and think of an answer.

We cannot do it manually as it would be impossible to create vectors for our soup string as there are more than hundred words in it.

For that, we will use the **CountVectorizer** class from **sklearn's library**, which will help us to create vectors in an easy manner.

Note : Refer C-132 for sklearn.

Import the **CountVectorizer** class from the **sklearn** module as,

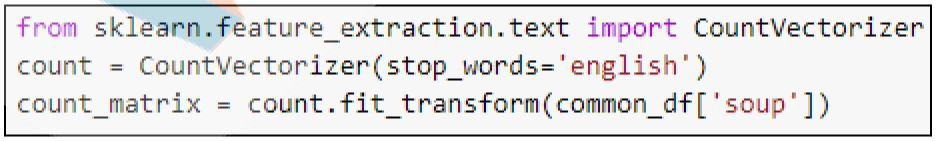
from sklearn.feature_extraction.text import CountVectorizer

Let's create an object of this class. Also, we want to remove the stop words.

count = CountVectorizer(stop_words = 'english')

Finally let's create a count matrix as,

Count_matrix = count.fit_transform(common_df['soup'])



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

Now we have our vectors ready, let's import the **cosine_similarity** class from **sklearn** and create a classifier based on our data with it.

```python
from sklearn.metrics.pairwise import cosine_similarity
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

Next, we want to change the index of our movie data to the name of the movies.

For that first we have to reset the index by using the command,

common_df = common_df.reset_index()

Here, we are resetting our data for **common_df** and then we are changing the index to the **title** of the movie.

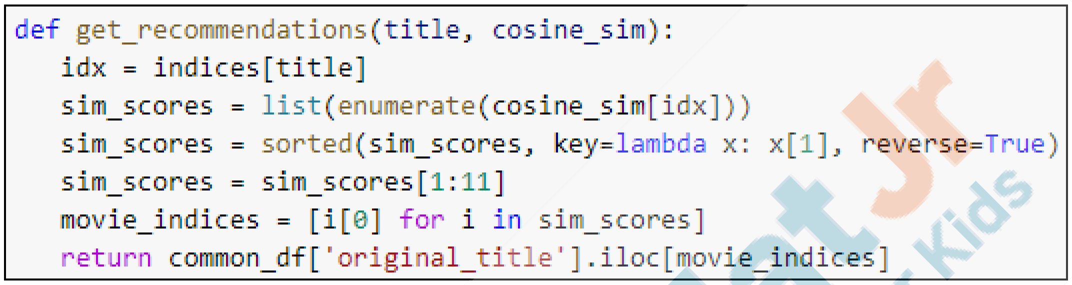
```python
common_df = common_df.reset_index()
indices = pd.Series(common_df.index, index=common_df['original_title'])
```

Finally, we will create the function that will get recommendations for us using our **cosine_similarity** classifier that we created earlier.

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.

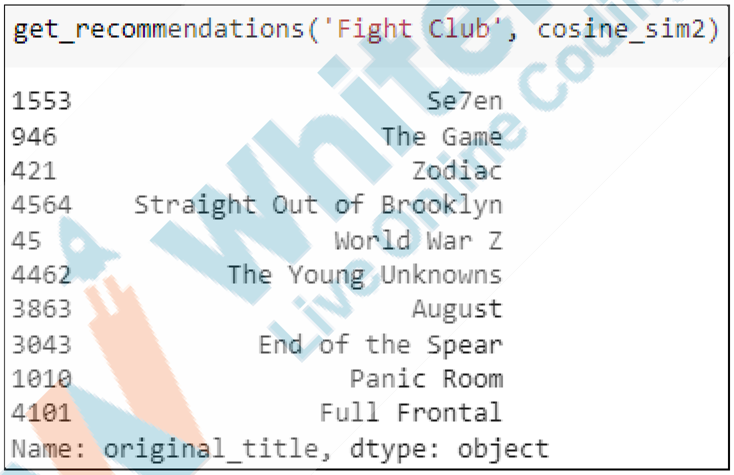
Next, we are creating a list of all the scores of the movies. This is the score of similarity of each movie with what the user likes. We are then using the sorted function on our data to sort the scores of all the movies and we are reversing its order with **reverse=True** attribute.

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!



```
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 common_df['original_title'].iloc[movie_indices]
```

Let's test it!

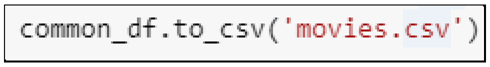
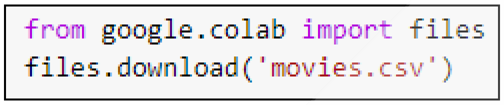
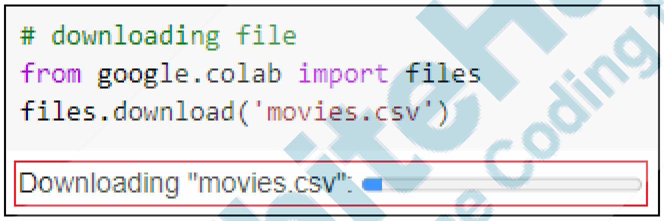
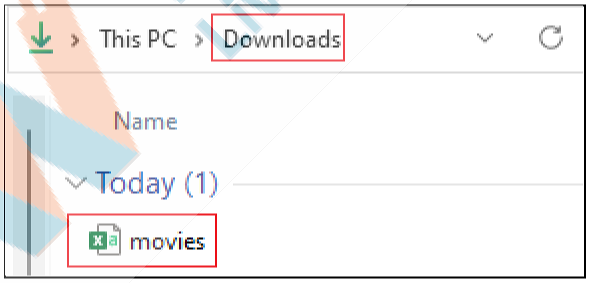


```
get_recommendations('Fight Club', cosine_sim2)

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: original_title, dtype: object
```

Great, it is working.

We have a lot of important information in our DataFrame which will help us in the later classes. Let's download it.

For that, we will first convert our DataFrame into a new csv file using the command,
common_df.to_csv('movies.csv')

 ```common_df.to_csv('movies.csv')```	
To download the **.csv file**, import the **files** class from **colab** module, and use the **.download()** method as, **files.download('movies.csv')**	
 ```from google.colab import files``` ```files.download('movies.csv')```	
Once you run this command, you will see a progress bar which says, **Downloading "movies.csv"**	
 ```# downloading file``` ```from google.colab import files``` ```files.download('movies.csv')``` ```Downloading "movies.csv":```	
Once the progress bar is completed, your **movies.csv** file will be downloaded in the **Downloads** directory	
 This PC > Downloads · Name · Today (1) · movies	
Continue WRAP-UP Session **Slide # to #** <**Note**: Only Applicable for Classes with VA>	

WhiteHat Jr
Live online Coding for Kids

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! Amazing. Now in the next class, we will be starting out with building our mobile app for a movie recommendation to the user but for that, we need to first build an API! We will be using Flask for that.	*Make sure you have given at least 2 hats-off during the class for:* Creatively Solved Activities +10 Great Question +10 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/1R2QrEvLg6SKS-e7wwvWeiOdnBEf2SDH1?usp=sharing
Teacher Activity 2	Reference Code	https://colab.research.google.com/drive/1kNL4wsEhVC0sJ-ClQ4lDeVpQGFJXqHT7?usp=sharing
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/6027bf0c-a68a-4c13-8fae-30a650ff80c9.pdf
Teacher Reference 2	Project Solution	https://colab.research.google.com/drive/1_ftiS5r-_hIFiiZWi2AL9TYm1LsJcwDZ?usp=sharing
Teacher Reference 3	Visual-Aid	Will be added after VA creation
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/0ebee9e5-b8a9-4d24-8670-fa72ebcff519.pdf
Student Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1Giq2PfakDIwiidLVx9-Vh_OtCdirdop6?usp=sharing