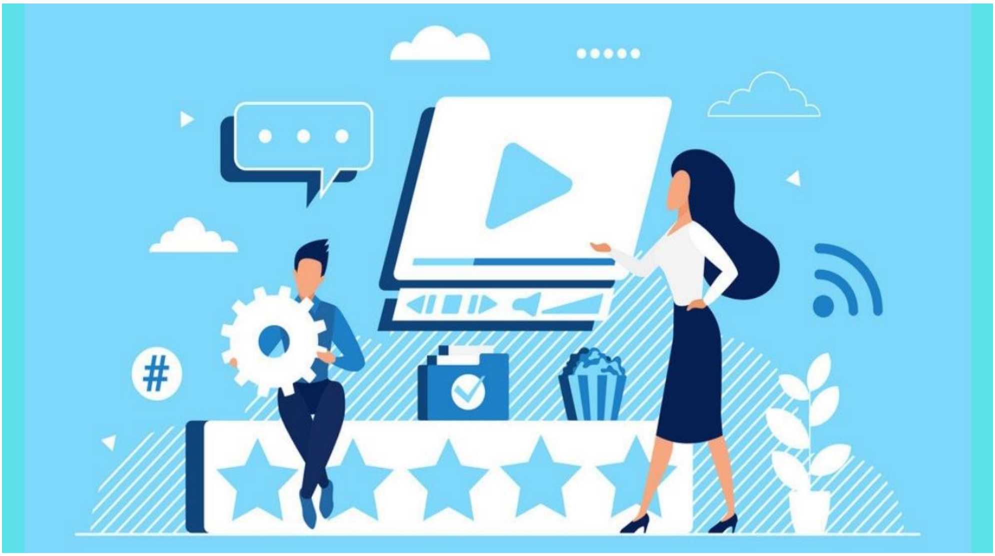
PROJECT- Movie Recommendation System using Machine Learning

Updated on Nov 20, 2023 16:09 IST

Every time you open up YouTube just to figure out the solution to your problem or just get the latest news, you end up spending more time. A similar thing happens when you decided on binging through a single movie/series from an OTT you end up watching more than what you had in your mind. Ever wondered how they were able to do such a thing? Most of the OTT platforms depend on their movie recommendation system.



Report By **Amit Singh**

Contents

But, what is a Recommendation System exactly?

A movie recommendation system is a fancy way to describe a process that tries to predict your preferred items based on your or people similar to you.

In layman's terms, we can say that a **Recommendation System** is a tool designed to predict/filter the items as per the user's behavior.

Why exactly do we need Recommendation Systems?

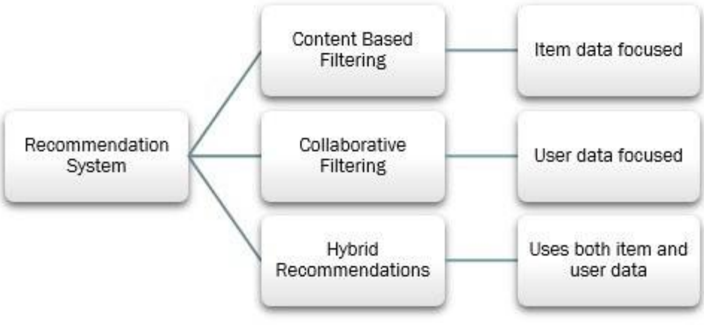
From a user's perspective, they are catered to fulfil the user's needs in the shortest time possible. For example, the type of content you watch on Netflix or Hulu. A person who likes to watch only *Korean drama* will see titles related to that only but a person who likes to watch *Action-based* titles will see that on their home screen.

From an organization's perspective, they want to keep the user as long as possible on the platform so that it will generate the most possible profit for them. With better recommendations, it creates positive feedback from the user as well. What good it will be to the organization to have a library of 500K+ titles when they cannot provide proper recommendations?

Recommendations are a great way to keep you watching but for Raghu the recommendations he gets wrong. But how? Well, as you know that recommendation systems are catered for **a user** but not for **multiple users.** Raghu lives in a joint family and everyone uses a single system to watch what they want. While OTT platforms give you a choice of adding multiple profiles but everyone else has already taken those and he is left with a single profile to share with his grandparents. So, Raghu decides to create his movie recommendation system. Before getting started he should understand the different types of recommendation systems.

Types of Recommendation Systems

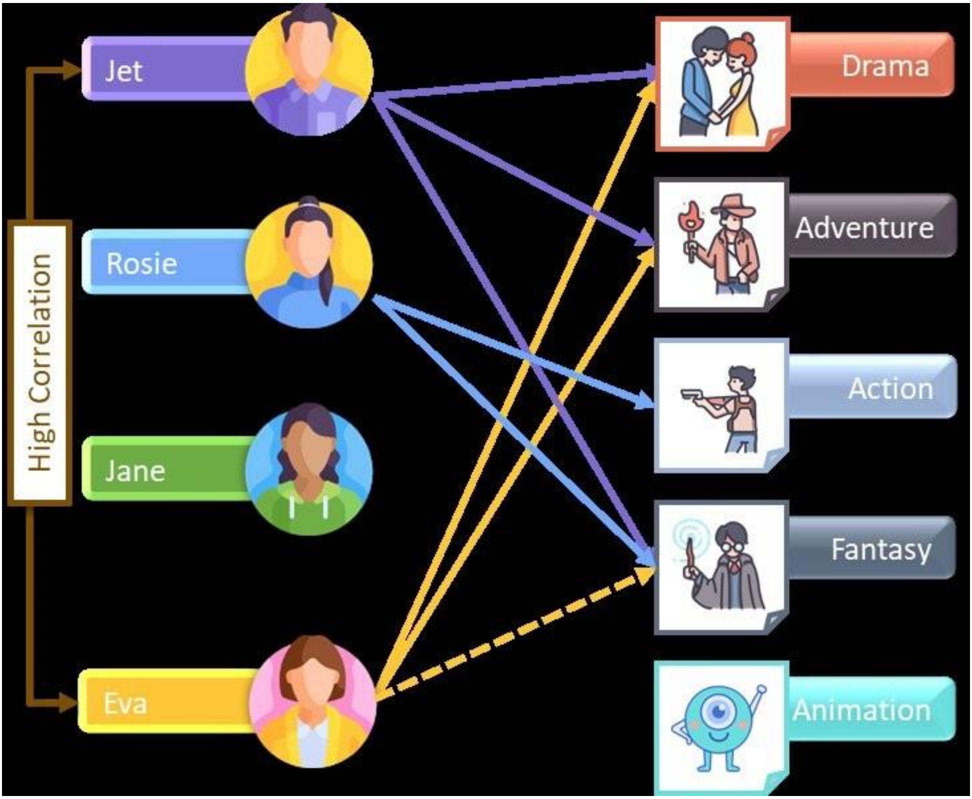
The following figure shows different kinds of recommender systems:



Collaborative Filtering

There are two types of collaborative filtering:

User-Based: Where we try to find similar users based on their item choices and recommend the items. A user-item rating matrix is created at first. Then, we find the correlations between the users and recommend items based on correlation.



Consider the above figure, we can see that:

- **Jet** likes **Drama, Adventure, and Fantasy-**based movies.
- **Rosie** likes **Action** and **Fantasy**-based movies.
- **Eva** likes **Drama** and **Adventure**-based movies.

From the above data, we can say that **Eva** is highly correlated to **Jet.** Thus, we can recommend her **Fantasy** movies as well.

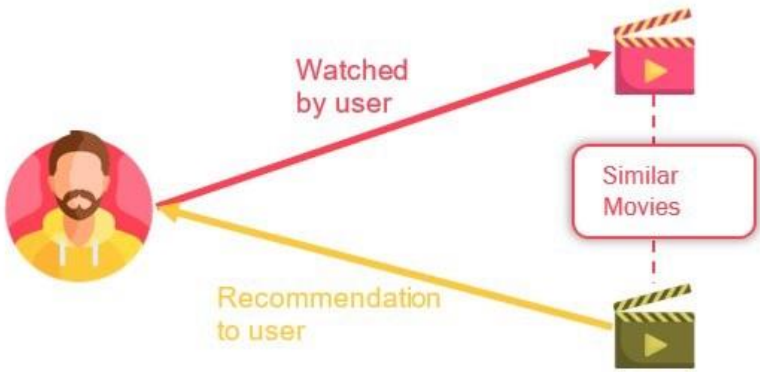
Item Based

Where we try to find a similar item based on their user's choices and recommend the items. A user-user item rating matrix is created at first. Then, we find the correlations between the items and recommend items based on correlation.

Using collaborative filtering becomes stale when either item or user choices differ.

Content-Based Filtering

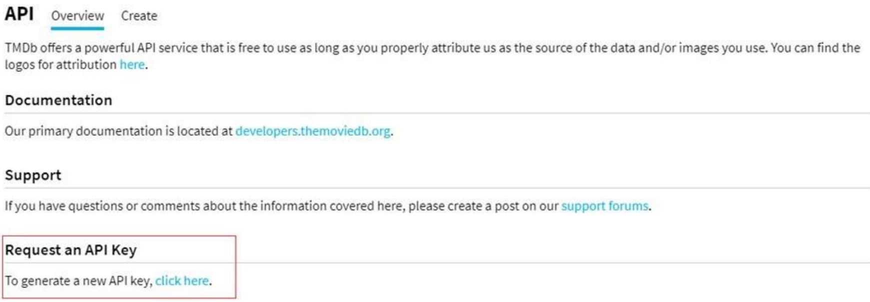
In this type, we will try to find similar items to the user's selected item. Consider the below figure:



Let's say Raghu watches a movie **X,** then in this case the model/method will try to find a similar movie based on its features like genres, actors and directors, etc. For example, *if a user likes to watch movies like say **Central Intelligence** where **Dwayne Johnson** is the protagonist, the model recommends the movies where **Dwayne Johnson** is either protagonist or has done some other part in it.*

Raghu wants the exact similar type of recommender system where he can input some movie names and related movies are given as recommendations. Let's see how he will apply machine learning to create a recommendation system.

To create the movie recommendation system Raghu has taken data from TMDB API. You can also request an API:



API Overview Create

TMDb offers a powerful API service that is free to use as long as you properly attribute us as the source of the data and/or images you use. You can find the logos for attribution here.

Documentation

Our primary documentation is located at developers.themoviedb.org.

Support

If you have questions or comments about the information covered here, please create a post on our support forums.

Request an API Key

To generate a new API key, click here.

Movie Dataset

The data gathered by Raghu has the following details:

- **Title**: Movie Title.
- **Overview**: Abstract of the Movie.
- **Popularity**: Movie popularity rating as per TMDB.
- **Vote_average**: Votes average out of 10.
- **Vote_count**: Number of votes from the users.
- **Release_date**: Date of release of the movie.
- **Keywords**: Keywords for the movie by TMDB in the list.
- **Genres**: Movie Genres in the list.
- **Cast**: Cast of the movie on the list.
- **Crew**: Crew of the movie in the list.

Reading Movies Data:

As Raghu loads the data, let's see how it looks:

Copy code

```
data =pd.read_csv('tmdb.csv.zip' ,compression ='zip',index_col='id') data .head ()
```

id	Unnamed: 0	title	overview	popularity	vote_average	vote_count	release_date	keywords	genres	cast	crew
19404	0	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	31.222	8.7	3323	1995-10-20	[]	['Comedy', 'Drama', 'Romance']	['Shah Rukh Khan', 'Kajol', 'Amrish Puri', 'An...	['Aditya Chopra']
278	1	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	76.654	8.7	20434	1994-09-23	['prison', 'corruption', 'police brutality', '...	['Drama', 'Crime']	['Tim Robbins', 'Morgan Freeman', 'Bob Gunton'...	['Frank Darabont']
238	2	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	75.306	8.7	15270	1972-03-14	['italy', 'loss of loved one', 'love at first ...	['Drama', 'Crime']	['Marlon Brando', 'Al Pacino', 'James Caan', '...	['Francis Ford Coppola']
724089	3	Gabriel's Inferno Part II	Professor Gabriel Emerson finally learns the t...	21.501	8.6	1369	2020-07-31	['based on novel or book']	['Romance']	['Melanie Zanetti', 'Giulio Berruti', 'James A...	['Tosca Musk']
424	4	Schindler's List	The true story of how businessman Oskar Schind...	40.585	8.6	12202	1993-11-30	['based on novel or book', 'factory', 'concent...	['Drama', 'History', 'War']	['Liam Neeson', 'Ben Kingsley', 'Ralph Fiennes...	['Steven Spielberg']

Cleaning Data

As you can see that before applying any machine learning models or even exploring the data we need to clean the data:

Removing Unnamed Column:

The Unnamed Columns are irritating as we cannot delete is normally. To remove this, Raghu gets the list of columns and renames the **"Unnamed: 0"** column and later removes it:

Copy code

```
data .columns=['temp' , 'title' , 'overview' , 'popularity' , 'vote_average'  , 'vote_count' , 'release_
```

id	title	overview	popularity	vote_average	vote_count	release_date	keywords	genres	cast	crew
19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	31.222	8.7	3323	1995-10-20	[]	['Comedy', 'Drama', 'Romance']	['Shah Rukh Khan', 'Kajol', 'Amrish Puri', 'An...	['Aditya Chopra']
278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	76.654	8.7	20434	1994-09-23	['prison', 'corruption', 'police brutality', '...	['Drama', 'Crime']	['Tim Robbins', 'Morgan Freeman', 'Bob Gunton'...	['Frank Darabont']
238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	75.306	8.7	15270	1972-03-14	['italy', 'loss of loved one', 'love at first ...	['Drama', 'Crime']	['Marlon Brando', 'Al Pacino', 'James Caan', '...	['Francis Ford Coppola']
724089	Gabriel's Inferno Part II	Professor Gabriel Emerson finally learns the t...	21.501	8.6	1369	2020-07-31	['based on novel or book']	['Romance']	['Melanie Zanetti', 'Giulio Berruti', 'James A...	['Tosca Musk']
424	Schindler's List	The true story of how businessman Oskar Schind...	40.585	8.6	12202	1993-11-30	['based on novel or book', 'factory', 'concent...	['Drama', 'History', 'War']	['Liam Neeson', 'Ben Kingsley', 'Ralph Fiennes...	['Steven Spielberg']

The output after dropping the column:

Changing Data Type

After filling the null values for empty columns, Raghu realizes that he will have to change the data type for most of them:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9480 entries, 19404 to 580
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   title         9480 non-null   object 
 1   overview      9464 non-null   object 
 2   popularity    9480 non-null   float64
 3   vote_average  9480 non-null   float64
 4   vote_count    9480 non-null   int64  
 5   release_date  9480 non-null   object 
 6   keywords      9480 non-null   object 
 7   genres        9480 non-null   object 
 8   cast          9480 non-null   object 
 9   crew          9480 non-null   object 
dtypes: float64(2), int64(1), object(7)
memory usage: 814.7+ KB
```

He creates a dictionary with columns as keys and their new type as values. Then, changes the datatype:

Copy code

```
new_types ={'title' : str,  'overview' : str,  'release_date'  : 'datetime64'  ,} for  col in new_types .
```

It seems that he has not treated the list columns. The list columns still have some empty values if he changes the type as a **list** directly he will get the following error:

```
~\anaconda3\lib\site-packages\pandas\core\dtypes\missing.py in <genexpr>(.0)
    677         # bytes, generic], Sequence[Union[int, float, complex, str, bytes, generic]],
    678         # Sequence[Sequence[Any]], _SupportsArray]"
--> 679         checker(arr[i : i + chunk_len]).all()  # type: ignore[arg-type]
    680         for i in range(0, total_len, chunk_len)
    681     )

~\anaconda3\lib\site-packages\pandas\core\dtypes\missing.py in <lambda>(x)
    668         # error: Incompatible types in assignment (expression has type "Callable[[Any],
    669         # Any]", variable has type "ufunc")
--> 670         checker = lambda x: _isna_array(  # type: ignore[assignment]
    671             x, inf_as_na=INF_AS_NA
    672         )

~\anaconda3\lib\site-packages\pandas\core\dtypes\missing.py in _isna_array(values, inf_as_na)
    252             result = ~np.isfinite(values)
    253         else:
--> 254             result = np.isnan(values)
    255 
    256     return result

TypeError: ufunc 'isnan' not supported for the input types, and the inputs could not be safely coerced to any supported types according to the casting rule ''safe''
```

The error means that it does not support the **list** datatype as of now. Instead, he creates that column as string type and keeps the values as **comma** separated:

Copy code

```
for col in ['keywords', 'genres', 'cast', 'crew']:    for val in ['[',']','\'']:        data[col]=data [col].str
```

id	title	overview	popularity	vote_average	vote_count	release_date	keywords	genres	cast	crew
19404	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	31.222	8.7	3323	1995-10-20		Comedy, Drama, Romance	Shah Rukh Khan, Kajol, Amrish Puri, Anupam Khe...	Aditya Chopra
278	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	76.654	8.7	20434	1994-09-23	prison, corruption, police brutality, prison c...	Drama, Crime	Tim Robbins, Morgan Freeman, Bob Gunton, Willi...	Frank Darabont
238	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	75.306	8.7	15270	1972-03-14	italy, loss of loved one, love at first sight,...	Drama, Crime	Marlon Brando, Al Pacino, James Caan, Robert D...	Francis Ford Coppola
724089	Gabriel's Inferno Part II	Professor Gabriel Emerson finally learns the t...	21.501	8.6	1369	2020-07-31	based on novel or book	Romance	Melanie Zanetti, Giulio Berruti, James Andrew ...	Tosca Musk
424	Schindler's List	The true story of how businessman Oskar Schind...	40.585	8.6	12202	1993-11-30	based on novel or book, factory, concentration...	Drama, History, War	Liam Neeson, Ben Kingsley, Ralph Fiennes, Caro...	Steven Spielberg

Data Exploration

After cleaning the data, Raghu wants to do some analysis of the data. He creates two functions for list columns:

- **get_unique(data,col):** Returns a list of unique items.

Copy code

```
def get_uniques(data ,col):
    '''
    data: Dataframe object
    col: column name with comma separated values
    ---
    returns: a list of unique category values in that column
    '''
    out=set([val.strip().lower() for val in ','.join(data[col].unique()).split(',')])
    try:
        out.remove('')
    except :
        return list(out)
    return list(out)
```

- **get_counts(data,col,categories):** Returns the counts for the unique items

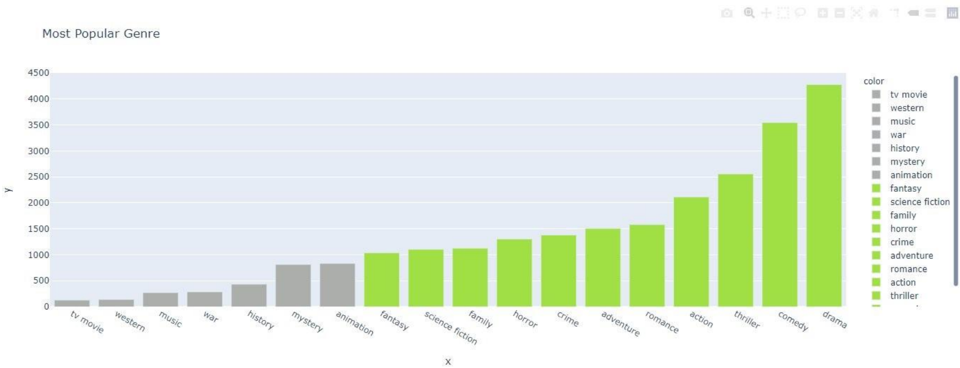
Copy code

```
def get_counts(data, col, categories):
    '''
    data: dataframe object
    col: name of the column
    categories: categories present
    ----
    return a dictionary with counts of each category
    '''
    categ = {category:None for category in categories}
    for category in tqdm(categories):
        val=0
        for index in data.index:
            if category in data.at[index,col].lower():
                val+=1
        categ[category]=val
    return categ
```

Using the two functions he creates a plotly chart to see most popular genres:

Copy code

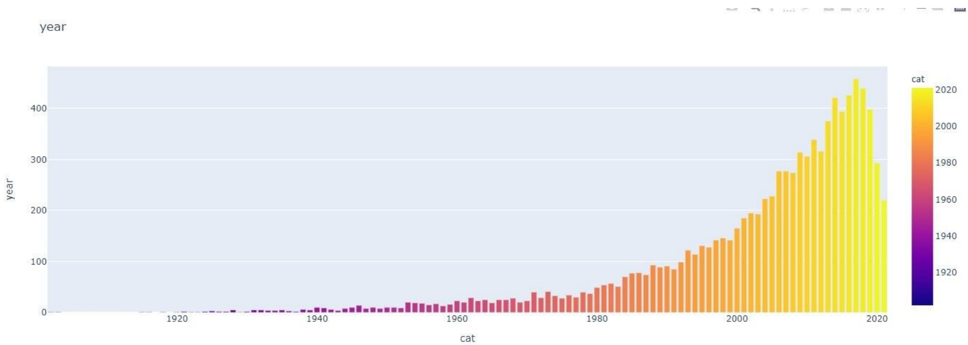
```
# Get the base counts of for each category and sort them by counts
base_counts  = get_counts (data , 'genres' , genres )
base_counts  = pd.DataFrame (index=base_counts .keys(),
                    data=base_counts .values (),
                    columns=['Counts' ])
base_counts .sort_values (by='Counts' , inplace=True )
# Plot the chart which shows top genres and separate by color where genre<1000
colors=['#abaeab'  if i<1000 else '#A0E045'  for i in  base_counts .Counts ]
fig  = px.bar (x=base_counts .index,
          y=base_counts ['Counts' ],
          title='Most Popular Genre' ,color_discrete_sequence =colors ,color=base_counts .inc
fig .show()
```



Later, he finds how plots movie release per year:

Copy code

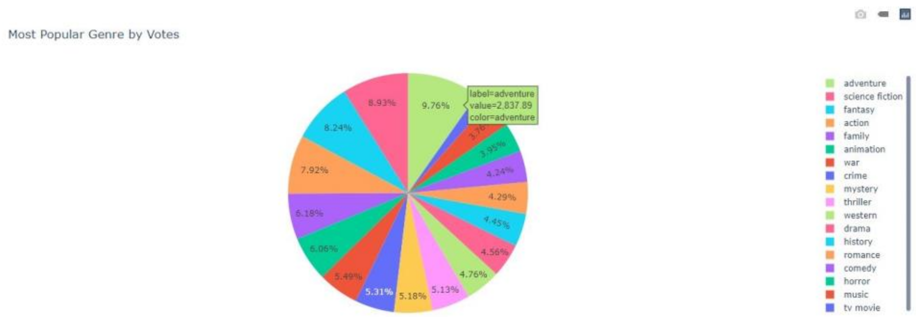
```python
# Function to plot value counts plots
def plot_value_counts_bar (data , col):
    '''
    data: Dataframe
    col: Name of the column to be plotted
    ----
    returns a plotly figure
    '''
    vc = pd.DataFrame (data [col].value_counts ())
    vc['cat' ] = vc.index
    fig = px.bar(vc, x='cat' , y=col, color='cat' , title =col)
    fig.update_layout ()

    return fig

data ['year' ]=data .release_date .dt .year
plot_value_counts_bar (data ,'year' )
```



Then, he creates another function to find the ratings by **popularity, vote_count, vote_average:**

Copy code

```python
def get_ratings(data, col,ratings_col, categories):
   '''
    data: dataframe object
    col: name of the column
    categories: categories present
    ----
    return a dictionary with average ratings of each category
    '''
    categ = {category : None for category in categories}
    for category in tqdm(categories):
        val=0
        ratings=0
        for index in data.index:
            if category in data.at[index,col].lower():
                val+=1
                ratings+=data.at[index,ratings_col]
        categ[category]=round(ratings/val,2)
    return categ
base_counts = get_ratings(data, 'genres','vote_count', genres)
base_counts = pd.DataFrame(index=base_counts.keys(),
                           data=base_counts.values(),
                           columns=['Counts'])
base_counts.sort_values(by='Counts', inplace=True)
fig = px.pie(names=base_counts.index,
             values=base_counts['Counts'],
             title='Most Popular Genre by Votes',color=base_counts.index)
fig.show()
```



You can explore more using the above functions like **most popular crew, most voted crew.**

Building Model

Raghu will be building the model in two ways:

Using CountVectorizer

It converts a collection of text into a matrix of counts with each hit.

Take an example with 3 sentences:
I enjoy Marvel movies.

I like Dwayne.

I like Iron Man.

The count vectorizer will create a matrix where it determines the frequency of each word.

	I	Like	Enjoy	Marvel	Movies	Dwayne	Iron	Man	.
I	0	2	1	0	0	0	0	0	0
Like	2	0	0	0	0	1	1	0	0
Enjoy	1	0	0	1	0	0	0	0	0
Marvel	0	0	0	0	1	0	0	0	0
Movies	0	0	0	1	0	0	0	0	1
Dwayne	0	1	0	0	0	0	0	0	1
Iron	0	1	0	0	0	0	1	0	0
Man	0	0	0	0	0	0	0	1	1
.	0	0	0	0	0	0	0	0	0

Focusing on the first row, “**like**” and “**enjoy**” are besides “**I**” for 2 and 1 times respectively. Similarly, other rows are calculated.

Raghu, creates the sentences for the **CountVectorizer**:

Copy code

```
def create_soup (data ):
   # Creating a simple text for countvectorizer to work with
   att  = data ['title' ].lower()
   for  i in data [1:]:
       att = att  + ' ' + str(i)
   return att

model_data =data .copy()
model_data =model_data [['title' ,'keywords' ,'genres' ,'cast' ,'crew']]
model_data ['soup']=model_data .apply(create_soup ,axis=1)
```

He gets the data in the following way:

```
id
19404     dilwale dulhania le jayenge  Comedy  Drama  Ro...
278       the shawshank redemption prison  corruption  p...
238       the godfather italy  loss of loved one  love a...
724089    gabriel's inferno part ii based on novel or bo...
424       schindler's list based on novel or book  facto...
                              ...
21435     french fried vacation 3 holiday  sardinia  ita...
17711     the adventures of rocky & bullwinkle helicopte...
17532     s. darko sequel  stranded  end of the world  s...
13908     the master of disguise disguise  aftercreditss...
580       jaws: the revenge shark attack  bahamas  dying...
Length: 9480, dtype: object
```

Now, he gets the cosine similarity scores:

Copy code

```
count = CountVectorizer (stop_words ='english')
count_matrix = count.fit_transform (model_data ['soup'])
cosine_sim2 = cosine_similarity(count_matrix )
```

Since we have the cosine similarity scores we can now get the recommendations. The below functions get the top 10 movies sorted by **popularity**:

Copy code

```
def get_recommendations_new(title, data, orig_data, cosine_sim=cosine_sim2):
    '''
    title: movie title
    data: model_data
    orig_data: original dataframe
    cosine_sim: cosine similarity matrix to use.
    ---
    returns: Table plot of plotly where top 10 movies by popularity are sorted.
    '''
    indices = pd.Series(data.index, index=data['title'])
    idx = indices[title]
    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    out=orig_data[[
        'title', 'vote_average', 'genres', 'crew', 'popularity'
    ]].iloc[movie_indices]
    out.genres = out.genres.str.replace(',', '<br>')
    out.crew = out.crew.str.replace(',', '<br>')
    final=out.sort_values(by='popularity',ascending=False)
    colorscale = [[0, '#477BA8'], [.5, '#ece4db'], [1, '#d8e2dc']]
    fig = ff.create_table(final, colorscale=colorscale, height_constant=70)
    return fig
```

Let’s try for “**The Shawshank Redemption**”:

title	vote_average	genres	crew	popularity
Avengers: Endgame	8.3	Adventure Science Fiction Action	Anthony Russo	458.39
Real Steel	6.9	Action Science Fiction Drama	Shawn Levy	160.405
Interstellar	8.4	Adventure Drama Science Fiction	Christopher Nolan	132.126
Justice Society: World War II	7.8	Animation War Science Fiction	Jeff Wamester	94.824
X-Men: Days of Future Past	7.5	Action Adventure Fantasy Science Fiction	Bryan Singer	74.254
War for the Planet of the Apes	7.1	Drama Science Fiction War	Matt Reeves	64.411
The Boy and the Beast	8.1	Animation Fantasy Action Adventure Drama	Mamoru Hosoda	40.892
Final Fantasy: The Spirits Within	6.2	Adventure Action Fantasy Science Fiction Thriller Romance	Hironobu Sakaguchi	14.793
The Postman	6.2	Science Fiction Adventure Action War	Kevin Costner	14.403
Like Father	6.3	Comedy Drama	Lauren Miller	7.118

Let’s see for another title “**Spirited Away**”:

title	vote_average	genres	crew	popularity
Soul	8.2	Animation Comedy Fantasy Family	Pete Docter	245.536
Onward	7.8	Family Animation Adventure Comedy Fantasy	Dan Scanlon	69.415
Alvin and the Chipmunks: Chipwrecked	5.7	Comedy Fantasy Family Music Animation	Mike Mitchell	66.424
Alvin and the Chipmunks: The Squeakquel	5.7	Comedy Family Animation Fantasy Music	Betty Thomas	61.828
Trolls	6.7	Family Animation Fantasy Adventure Comedy Music	Mike Mitchell	60.679
The Book of Life	7.5	Animation Adventure Comedy Family Fantasy	Jorge R. Gutierrez	59.088
Rock Dog	6.1	Adventure Animation Comedy Family Music	Ash Brannon	22.548
UglyDolls	6.7	Family Animation Comedy Adventure Fantasy Music	Kelly Asbury	22.379
The Brave Little Toaster	6.9	Fantasy Adventure Animation Comedy Family Music	Jerry Rees	11.571
Sweet and Lowdown	6.9	Comedy Drama Music	Woody Allen	9.29

Using NearestNeighbors

We can use **NearestNeighbors** as well to create our recommendation system. Before training the model, we need to process the data for optimal performance:

Copy code

```
nn_data=data.copy()
def fill_genre(value,col,categories=genres):
    if col in value.lower() :
        return 1
    else:
        return 0
# Create genre columns
for col in genres:
    nn_data[col]=None
for index in tqdm(nn_data.index):
    for col in genres:
        nn_data.at[index,col]=fill_genre(nn_data.at[index,'genres'],col)
for col in genres:
    nn_data[col]=nn_data.genres.apply(fill_genre,args=(col,))
nn_data.drop(['overview','release_date','genres','title'],axis=1,inplace=True)
for col in ['keywords','cast','crew']:
    nn_data[col]=LabelEncoder().fit_transform(nn_data[col])
```

Traning the model:

Copy code

```
model_knn = NearestNeighbors (metric ='cosine',
                              algorithm='auto' ,
                              n_neighbors=20,
                              n_jobs=-1)
model_knn.fit (nn_data )
```

Now, Let’s test our model:

Copy code

```
# Create a function to recommend top 10 movies
def recommend_movies(movie,nn_data,orig_data):
    orig_data.reset_index(inplace=True)
    nn_data.reset_index(inplace=True,drop=True)
    movie_index=nn_data[orig_data.title==movie].index
    distances, indices = model_knn.kneighbors(np.array(nn_data.iloc[movie_index]).reshape(
    1, -1),n_neighbors=10)

    out=orig_data[[
        'title', 'vote_average', 'genres', 'crew', 'popularity'
    ]].iloc[indices[0]]
    out.genres = out.genres.str.replace(',', '<br>')
    out.crew = out.crew.str.replace(',', '<br>')
    final=out.sort_values(by='popularity',ascending=False)
    colorscale = [[0, '#fad2e1'], [.5, '#fde2e4'], [1, '#fff1e6']]
    fig = ff.create_table(final, colorscale=colorscale, height_constant=70)
    return fig
```

Let’s check for the movie “**Thor**”:

title	vote_average	genres	crew	popularity
Doctor Strange	7.4	Action Adventure Fantasy Science Fiction	Scott Derrickson	322.507
Coco	8.2	Family Animation Fantasy Music Comedy Adventure	Lee Unkrich	231.649
Thor	6.8	Adventure Fantasy Action	Kenneth Branagh	201.812
Captain America: The First Avenger	7.0	Action Adventure Science Fiction	Joe Johnston	155.534
Fantastic Beasts and Where to Find Them	7.4	Adventure Fantasy	David Yates	146.861
Guardians of the Galaxy Vol. 2	7.6	Adventure Action Science Fiction	James Gunn	134.225
Inglourious Basterds	8.2	Drama Action Thriller War	Quentin Tarantino	100.238
Guardians of the Galaxy	7.9	Action Science Fiction Adventure	James Gunn	80.407
Batman Begins	7.7	Action Crime Drama	Christopher Nolan	60.649
Get Out	7.6	Mystery Thriller Horror	Jordan Peele	48.406

Let's try for "**Eternals**":

title	vote_average	genres	crew	popularity
Venom: Let There Be Carnage	7.2	Science Fiction Action Adventure	Andy Serkis	5011.79
Shang-Chi and the Legend of the Ten Rings	7.8	Action Adventure Fantasy	Destin Daniel Cretton	2417.301
Don't Look Up	7.3	Drama Comedy Science Fiction	Adam McKay	1907.587
Eternals	7.1	Action Adventure Fantasy Science Fiction	Chloé Zhao	1605.268
Demon Slayer -Kimetsu no Yaiba- The Movie: Mugen Train	8.4	Animation Action Adventure Fantasy	Haruo Sotozaki	974.253
After We Fell	7.2	Romance Drama	Castille Landon	918.194
Movie 43	4.5	Comedy	Griffin Dunne	76.959
The Fountain	6.9	Drama Adventure Science Fiction Romance	Darren Aronofsky	12.237
Sense and Sensibility	7.5	Drama Romance	Ang Lee	12.211
Regression	5.6	Horror Mystery Thriller Crime	Alejandro Amenábar	11.046

Run this demo in Colab – Try it Yourself!

Conclusion

In this Report, we have learned how to create a recommendation system using machine learning. Apart from movie recommendations, you can try making recommender systems from shopping products, news, typing assistance, and so on.