**1.INTRODUCTION**

**1.1 MOVIE RECOMMENDATION SYSTEM**

A recommendation system or recommendation engine is a model used for information filtering where it tries to predict the preferences of a user and provide suggests based on these preferences. These systems have become increasingly popular nowadays and are widely used today in areas such as movies, music, books, videos, clothing, restaurants, food, places and other utilities. These systems collect information about a user's preferences and behaviour, and then use this information to improve their suggestions in the future. Movies are a part and parcel of life. There are different types of movies like some for entertainment, some for educational purposes, some are animated movies for children, and some are horror movies or action films. Movies can be easily differentiated through their genres like comedy, thriller, animation, action etc. Other way to distinguish among movies can be either by releasing year, language, director etc. Watching movies online, there are a number of movies to search in our most liked movies .

Movie Recommendation Systems helps us to search our preferred movies among all of these different types of movies and hence reduce the trouble of spending a lot of time searching our favourable movies. So, it requires that the movie recommendation system should be very reliable and should provide us with the recommendation of movies which are exactly same or most matched with our preferences. A large number of companies are making use of recommendation systems to increase user interaction and enrich a user's shopping experience. Recommendation systems have several benefits, the most important being customer satisfaction and revenue. Movie Recommendation system is very powerful and important system. But, due to the problems associated with pure collaborative approach, movie recommendation systems also suffers with poor recommendation quality and scalability issues.

**1.2 PROJECT OBJECTIVE**

In this hustling world, entertainment is a necessity for each one of us to refresh our mood and energy. Entertainment regains our confidence for work and we can work more enthusiastically. For revitalizing ourselves, we can listen to our preferred music or can watch movies of our choice. For watching favourable movies online we can utilize movie recommendation systems, which are more reliable, since searching of preferred movies will require more and more time which one cannot afford to waste.

**1.3 PROJECT SPECIFICATION**

This project aims at development of an movie recommendation system that faciliates the customer to manage their reservation and it also makes the ticket booking process as easier method which can be done from anywhere

**2.SYSTEM SPECIFICATION**

**2.1Hardware specification**

* Processor : Intel dual core
* Processor speed: 1.04GHZ
* Ram : 1GB
* Moniter
* Keyboard
* Mouse

**2.2** **Software** **specification**

* OS
* Language : Python
* Compiler : googlecolab

**3.PACKAGES**

**3.1 NUMPY**

* NumPy is a Python library used for working with arrays.
* It also has functions for working in domain of linear algebra, fourier transform, and matrices.
* NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
* NumPy stands for Numerical Python.

**INSTALLING NUMPY PACKAGE**

pip install numpy

## WHY USE NUMPY?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

**IMPORT NUMPY**

Once NumPy is installed, import it in your applications by adding the import keyword:

import numpy

## NUMPY AS np:

NumPy is usually imported under the np.

Create an np with the as keyword while importing:

import numpy as np

Now the NumPy package can be referred to as np instead of numpy.

**Example:**

import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

## 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

**3.2 PANDAS**

* Pandas is a Python library used for working with data sets.
* It has functions for analyzing, cleaning, exploring, and manipulating data.
* The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

## Why Use Pandas

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Pandas gives you answers about the data. Like:

* Is there a correlation between two or more columns?
* What is average value?
* Max value?
* Min value?
* Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

**INSTALLING PANDAS PACKAGE**

pip install pandas

## Import Pandas

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Now Pandas is imported and ready to use

**Example:**

Importpandas

mydataset={'cars':["BMW","Volvo","Ford"],'passings':[3,7,2]}  
myvar=pandas.DataFrame(mydataset)  
print(myvar)

## Pandas as pd

Pandas is usually imported under the pd

Create an pd with the as keyword while importing:

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

**3.3 MATPLOTLIB**

* Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.
* As such, it offers a viable open source alternative to **MATLAB.** Developers can also use matplotlib’s APIs(Application Programming Interfaces) to embed plots inGUI applications.

A Python matplotlib script is structured so that a fewlines of code are all that is required in most instancesto generate a visual data plot.

The matplotlib scripting layer overlays two APIs:

* The pyplot API is a hierarchy of Python codeobjects topped by matplotlib.pyplot
* An OO (Object-Oriented) API collection of objectsthat can be assembled with greater flexibility thanpyplot. This API provides direct access to Matplotlib’sbackend layers.

**Matplotlib and Pyplot in Python :**

The pyplot API has a convenient MATLAB-style statefulinterface. In fact, matplotlib was originally written as an open source alternative for MATLAB. The OO API and its interface is more customizable and powerful than pyplot, but considered more difficult to use. As a result, the pyplot interface is more commonly used, and is referred to by default in this article.

Understanding matplotlib’s pyplot API is key to understanding how to work with plots:

* **matplotlib.pyplot.figure**: Figure is the top-level container. It includes everything visualized in a plot including one or more Axes.
* **matplotlib.pyplot.axes**: Axes contain most of the elements in a plot: Axis, Tick, Line2D, Text, etc., and sets the coordinates. It is the area in which data is plotted. Axes include the X-Axis, Y-Axis, and possibly a Z-Axis, as well.

**Installing Matplotlib :**

pip install matplotlib

**3.3.1 MATPLOTLIB BAR PLOT:**

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. One of the axis of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

**Creating a bar plot:**

The matplotlib API in Python provides the bar() function which can be used in MATLAB style use or as an object-oriented API. The syntax of the bar() function to be used with the axes is as follows:- plt.bar(x, height, width, bottom, align).The function creates a bar plot bounded with a rectangle depending on the given parameters. Following is a simple example of the bar plot, which represents the number of students enrolled in different courses of an institute.

**EXAMPLE:**

import numpy as np

import matplotlib.pyplot as plt

data = {'C':20, 'C++':15, 'Java':30,'Python':35}

courses = list(data.keys())

values = list(data.values())

fig = plt.figure(figsize = (10, 5))

plt.bar(courses, values, color ='maroon',width = 0.4)

plt.xlabel("Courses offered")

plt.ylabel("No. of students enrolled")

plt.title("Students enrolled in different courses")

plt.show()

**Output:**

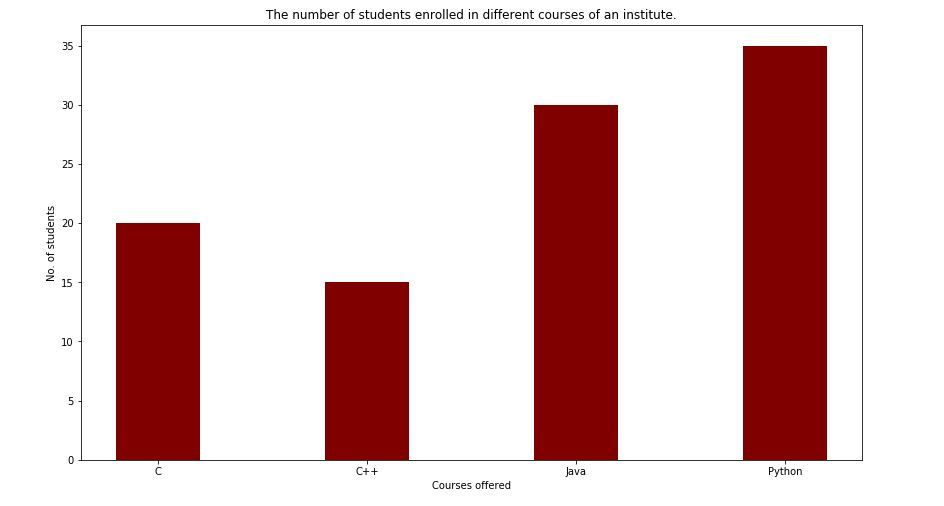


FIGURE:1-BAR CHART

**3.3.2 MATPLOTLIB HISTOGRAM:**

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph.

To construct a histogram, follow these steps −

* Bin the range of values.
* Divide the entire range of values into a series of intervals.
* Count how many values fall into each interval.

The bins are usually specified as consecutive, non-overlapping intervals of a variable.

The **matplotlib.pyplot.hist()** function plots a histogram. It computes and draws the histogram of x.

**EXAMPLE:**

from matplotlib import pyplot as plt

import numpy as np

fig,ax = plt.subplots(1,1)

a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])

ax.hist(a, bins = [0,25,50,75,100])

ax.set\_title("histogram of result")

ax.set\_xticks([0,25,50,75,100])

ax.set\_xlabel('marks')

ax.set\_ylabel('no. of students')

plt.show()

**OUTPUT:**

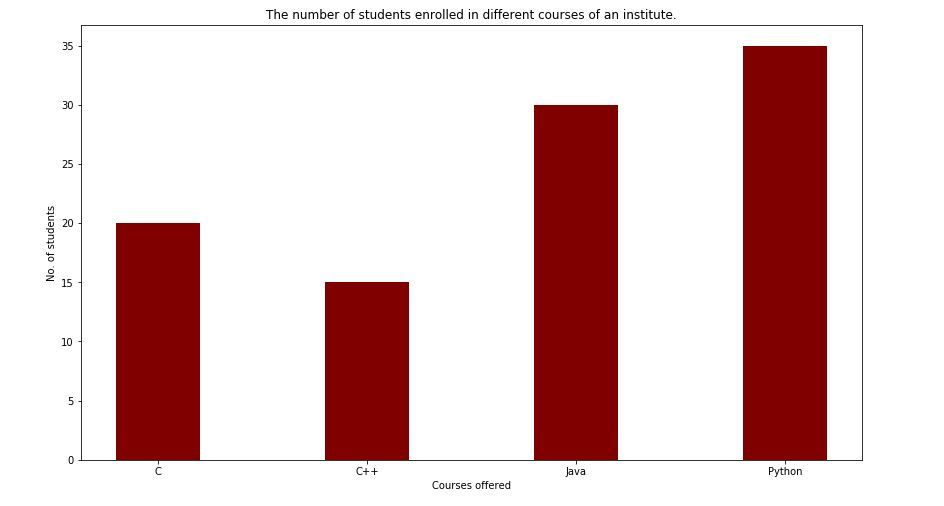


FIGURE:2-BAR CHART

**4.APPENDIX**

**4.1 SOURCE CODE**

import pandas as pd   
import numpy as np  
import warnings  
warnings.filterwarnings('ignore')

Next, we load in the data set using pandas read\_csv() utility. The dataset is tab separated so we pass in \t to the sep parameter. We then pass in the column names using the names parameter.

df = pd.read\_csv('u.data', sep='\t', names=['user\_id','item\_id','rating','titmestamp'])

Now let’s check the head of the data to see the data we are dealing with.

df.head()

It would be nice if we can see the titles of the movie instead of just dealing with the IDs. Let’s load in the movie titles and merge it with this dataset.

movie\_titles = pd.read\_csv('Movie\_Titles')

movie\_titles.head()

Since the item\_id columns are the same we can merge these datasets on this column.

df = pd.merge(df, movie\_titles, on='item\_id')

df.head()

Let’s look at what each column represents:

* user\_id - the ID of the user who rated the movie.
* item\_id - the ID of the movie.
* rating - The rating the user gave the movie, between 1 and 5.
* timestamp - The time the movie was rated.
* title - The title of the movie.

Using the describe or info commands we can get a brief description of our dataset. This is important in order to enable us to understand the dataset we are working with.

df.describe()

We can tell that the average rating is 3.52 and the max is 5. We also see that the dataset has 100003 records.

Let’s now create a data frame with the average rating for each movie and the number of ratings. We are going to use these ratings to calculate the correlation between the movies later. Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. Movies that have a high correlation coefficient are the movies that are most similar to each other. In our case, we shall use the Pearson correlation coefficient. This number will lie between -1 and 1. 1 indicates a positive linear correlation while -1 indicates a negative correlation. 0 indicates no linear correlation. Therefore, movies with a zero correlation are not similar at all. In order to create this data frame we use pandas groupby functionality. We group the dataset by the title column and compute its mean to obtain the average rating for each movie.

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())

ratings.head()

Next we would like to see the number of ratings for each movie. We do this by creating a number\_of\_ratings column. This is important so that we can see the relationship between the average rating of a movie and the number of ratings the movie got. It is very possible that a 5-star movie was rated by just one person. It is therefore statistically incorrect to classify that movie has a 5-star movie. We will, therefore, need to set a threshold for the minimum number of ratings as we build the recommender system. In order to create this new column, we use pandas groupby utility. We group by the title column and then use the count function to calculate the number of ratings each movie got. Afterward we view the new data frame by using the head() function.

ratings['number\_of\_ratings'] = df.groupby('title')['rating'].count()  
ratings.head()

Let’s now plot a Histogram using pandas plotting functionality to visualize the distribution of the ratings

import matplotlib.pyplot as plt  
%matplotlib inline  
ratings['rating'].hist(bins=50)

We can see that most of the movies are rated between 2.5 and 4. Next, let’s visualize the number\_of\_ratings column in as similar manner.

ratings['number\_of\_ratings'].hist(bins=60)

From the above histogram, it is clear that most movies have few ratings. Movies with most ratings are those that are most famous.

Let’s now check the relationship between the rating of a movie and the number of ratings. We do this by plotting a scatter plot using seaborn. Seaborn enables us to do this using the jointplot() function.

Import seaborn as sns  
sns.jointplot(x='rating', y='number\_of\_ratings', data=ratings)

From the diagram we can see that there is a positive relationship between the average rating of a movie and the number of ratings. The graph indicates that the more the ratings a movie gets the higher the average rating it gets. This is important to note especially when choosing the threshold for the number of ratings per movie.

Let’s now move on swiftly and create a simple item-based recommender system. In order to do this, we need to convert our dataset into a matrix with the movie titles as the columns, the user\_id as the index and the ratings as the values. By doing this we shall get a data frame with the columns as the movie titles and the rows as the user ids. Each column represents all the ratings of a movie by all users. The rating appears as NAN where a user didn't rate a certain movie. We shall use this matrix to compute the correlation between the ratings of a single movie and the rest of the movies in the matrix. We use pandas pivot\_table utility to create the movie matrix.

movie\_matrix = df.pivot\_table(index='user\_id', columns='title', values='rating')  
movie\_matrix.head()

Next let’s look at the most rated movies and choose two of them to work with in this simple recommender system. We use pandas  sort\_values utility and set ascending to false in order to arrange the movies from the most rated. We then use the head() function to view the top 10.

ratings.sort\_values('number\_of\_ratings', ascending=False).head(10)

Let’s assume that a user has watched Air Force One (1997) and Contact (1997). We would like to recommend movies to this user based on this watching history. The goal is to look for movies that are similar to Contact (1997) and Air Force One (1997 which we shall recommend to this user. We can achieve this by computing the correlation between these two movies’ ratings and the ratings of the rest of the movies in the dataset. The first step is to create a data frame with the ratings of these movies from our movie\_matrix.

AFO\_user\_rating = movie\_matrix['Air Force One (1997)']  
contact\_user\_rating = movie\_matrix['Contact (1997)']

We now have the data frame showing the user\_id and the rating they gave the two movies. Let's take a look at them below.

AFO\_user\_rating.head()

contact\_user\_rating.head()

In order to compute the correlation between two dataframes we use pandas corwith functionality. Corrwith computes the pairwise correlation of rows or columns of two data frames objects. Let's use this functionality to get the correlation between each movie's rating and the ratings of the Air Force One movie.

similar\_to\_air\_force\_one=movie\_matrix.corrwith(AFO\_user\_rating)

We can see that the correlation between Air Force One movie and Till There Was You (1997) is 0.867. This indicates a very strong similarity between these two movies.

similar\_to\_air\_force\_one.head()

Let’s move on and compute the correlation between Contact (1997) ratings and the rest of the movies ratings. The procedure is the same as the one used above.

similar\_to\_contact = movie\_matrix.corrwith(contact\_user\_rating)

We realize from the computation that there is a very strong correlation (of 0.904) between Contact (1997) and Til There Was You (1997).

similar\_to\_contact.head()

As noticed earlier our matrix had very many missing values since not all the movies were rated by all the users. We therefore drop those null values and transform correlation results into data frames to make the results look more appealing.

corr\_contact=pd.DataFrame(similar\_to\_contact,columns=['Correlation'])  
corr\_contact.dropna(inplace=True)  
corr\_contact.head()

corr\_AFO=pd.DataFrame(similar\_to\_air\_force\_one,columns=['correlation'])  
corr\_AFO.dropna(inplace=True)  
corr\_AFO.head()

These two data frames above show us the movies that are most similar to Contact (1997) and Air Force One (1997) movies respectively. However, we have a challenge in that some of the movies have very few ratings and may end up being recommended simply because one or two people gave them a 5-star rating. We can fix this by setting a threshold for the number of ratings. From the histogram earlier we saw a sharp decline in a number of ratings from 100. We shall, therefore, set this as the threshold, however, this is a number you can play around with until you get a suitable option. In order to do this, we need to join the two data frames with the number\_of\_ratings- column in the rating data frame.

corr\_AFO = corr\_AFO.join(ratings['number\_of\_ratings'])  
corr\_contact = corr\_contact.join(ratings['number\_of\_ratings'])

corr\_AFO .head()

corr\_contact.head()

We shall now obtain the movies that are most similar to Air Force One (1997) by limiting them to movies that have at least 100 reviews. We then sort them by the correlation column and view the first 10.

corr\_AFO[corr\_AFO['number\_of\_ratings']>100].sort\_values(by='correlation',ascending=False).head(10)

We notice that Air Force One (1997) has a perfect correlation with itself, which is not surprising. The next most similar movie to Air Force One (1997) is Hunt for Red October, The (1990) with a correlation of 0.554. Clearly, by changing the threshold for the number of reviews, we get different results from the previous way of doing it. Limiting the number of rating gives us better results and we can confidently recommend the above movies to someone who has watched Air Force One (1997).

Now let’s do the same for Contact (1997) movie and see the movies that are most correlated to it.

corr\_contact[corr\_contact['number\_of\_ratings']>100].sort\_values(by='Correlation',ascending=False).head(10)

Once again, we get different results. The most similar movie to Contact (1997) is Philadelphia (1993) with a correlation coefficient of 0.446 with 137 ratings. So, if somebody liked Contact (1997) we can recommend the above movies to them.

Obviously, this is a very simple way of building a recommender system and is nowhere close to industry standards.

**4.2 SCREENSHOT**

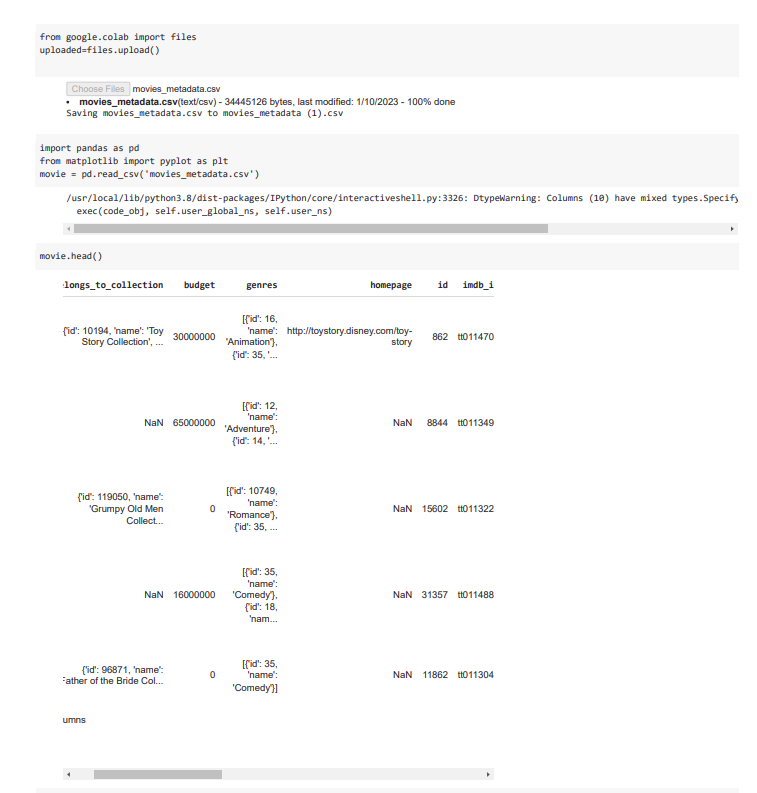
****

FIGURE 3- head of the datset

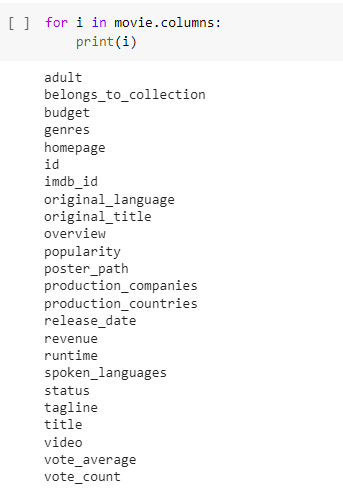
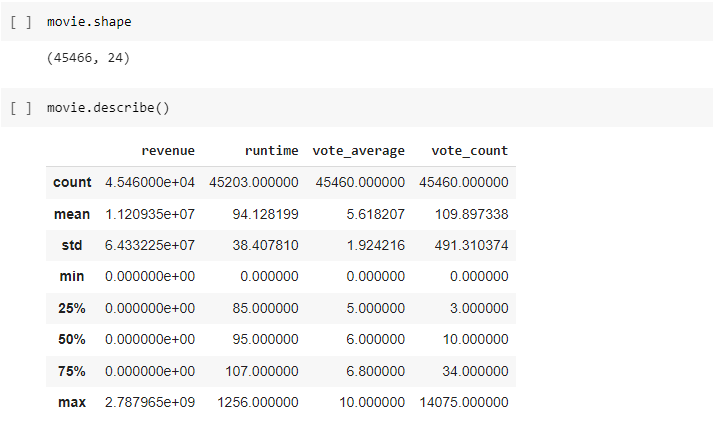
****

FIGURE 4- Columns of the dataset

****FIGURE: 5-shape and describe()

****

FIGURE:6- bar graph for original language

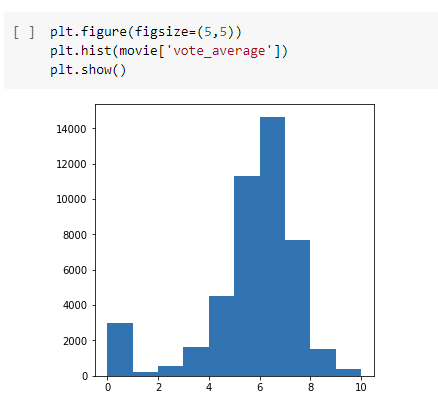
****

FIGURE:7- histogram

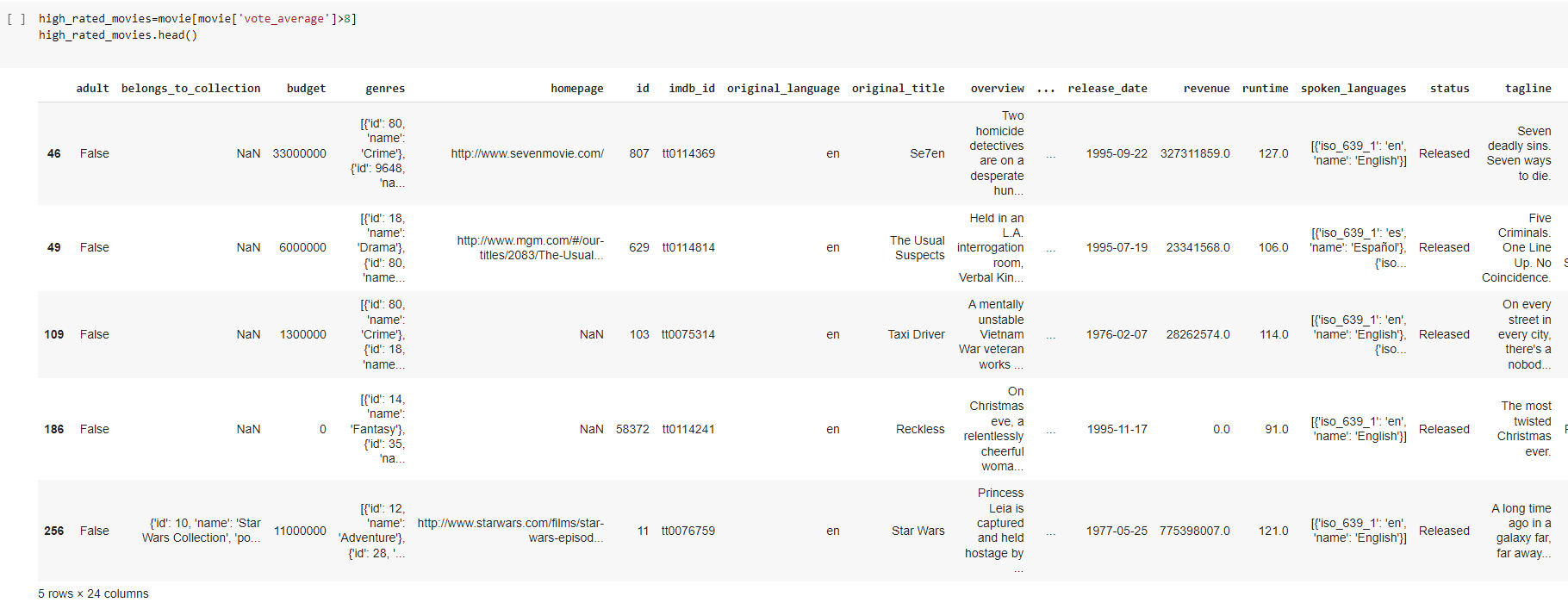
****

FIGURE:8- vote average>8

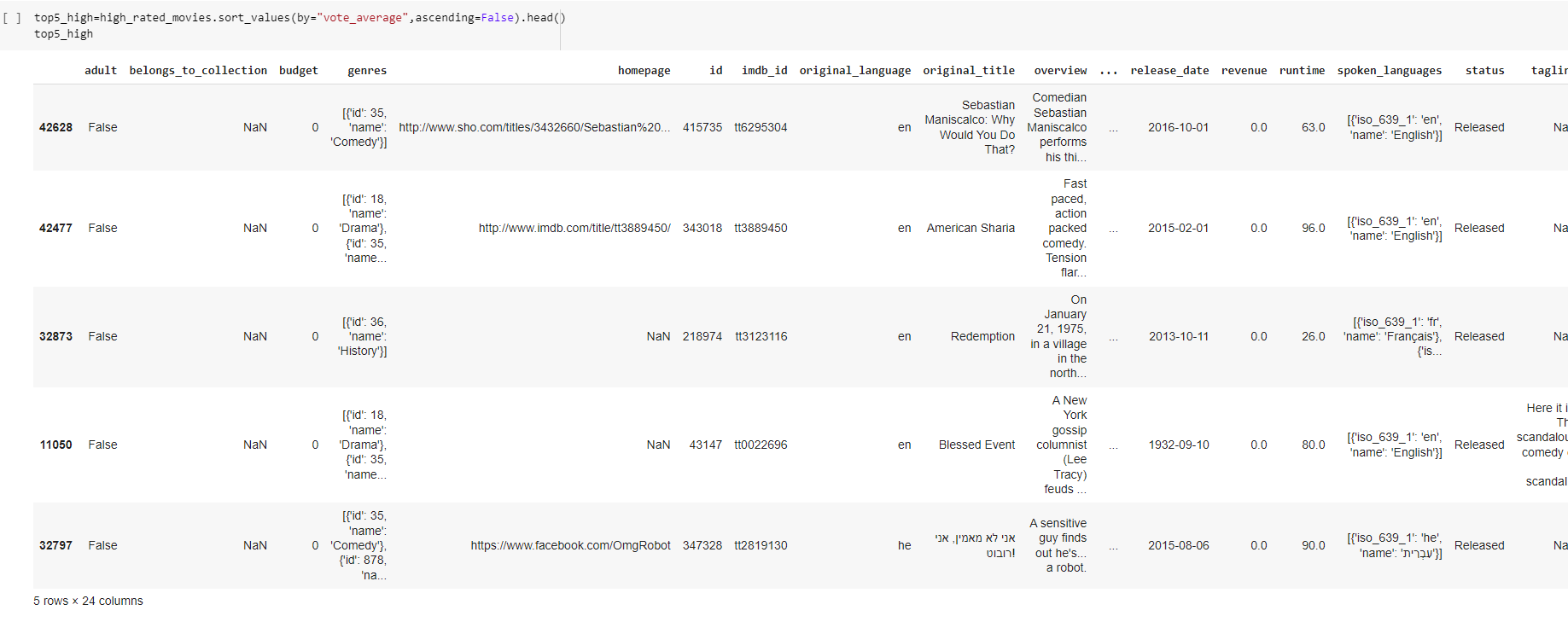
****

FIGURE:9 vote average

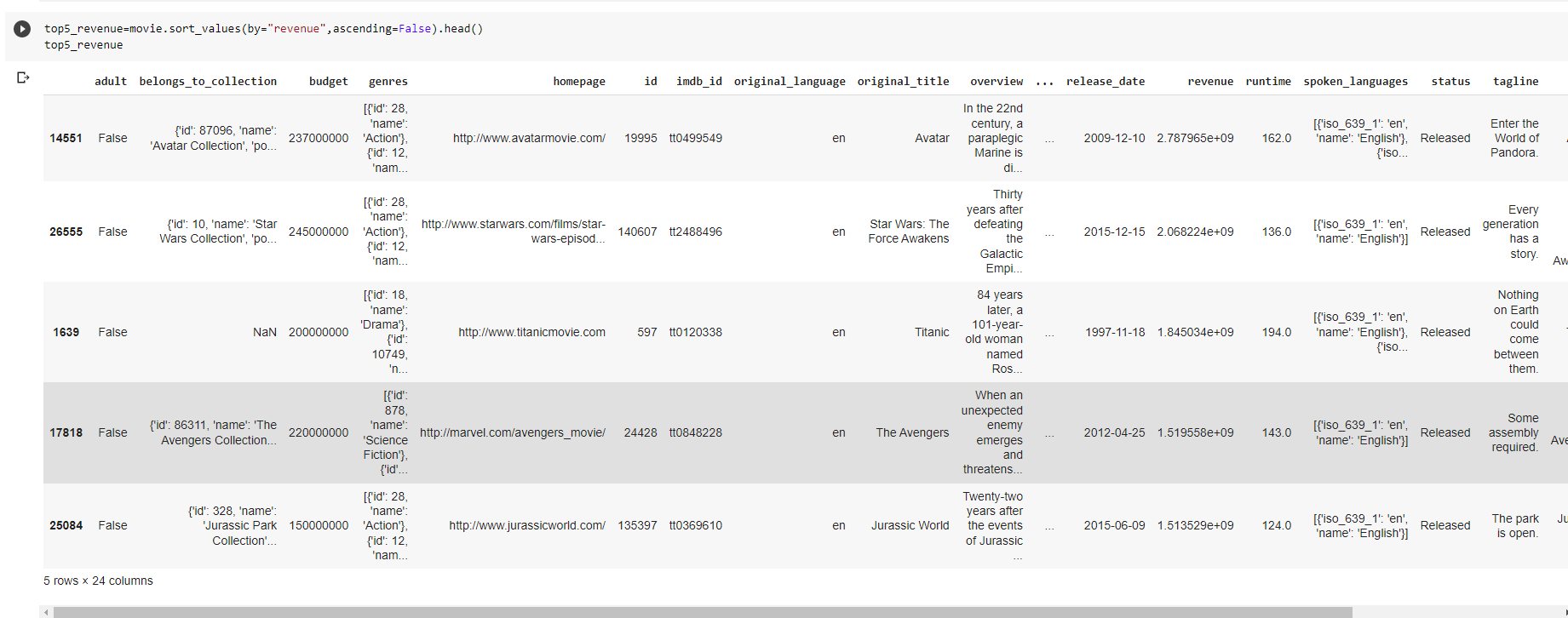
****

FIGURE:10--revenue

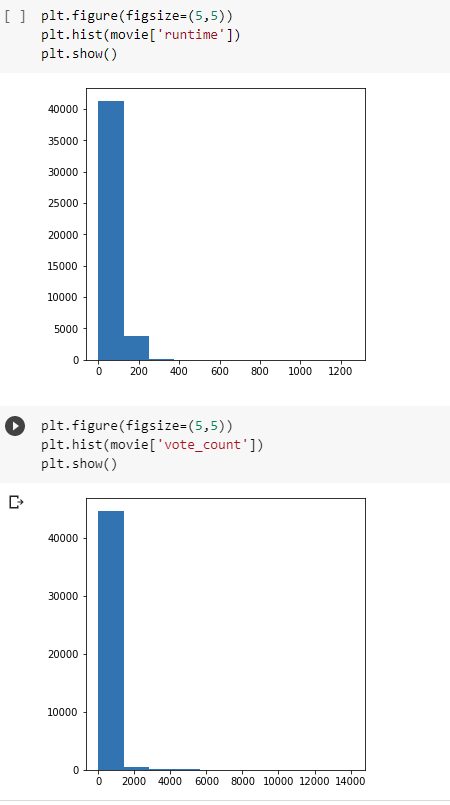
****

FIGURE:11­- histogram for votecount

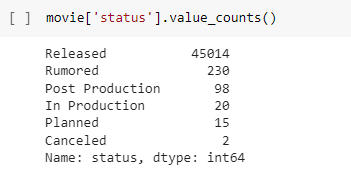
****

FIGURE:12-status

**5. CONCLUSION**

In the Movie Recommendation System, the Cosine Similarity algorithm has been used to recommend the best movies that are related to the movie entered by the user based on different factors such as the genre of the movie, overview, the cast as well as the ratings given to the movie. Cosine Similarity has given fair results even after running several tests on it and has been quite accurate at recommending the movies.

Sentiment analysis also plays an important role in this study. It basically aims to classify the reviews into positive or negative. Two algorithms have been used for the same. One of which is NB and other is SVC. The main reason behind using two algorithms is to find out what which is the best algorithm to classify the reviews because the reviews have huge diversity in them, so it is very important to choose the right algorithm for classification. Finally, the experimental results show that SVM Algorithm has better accuracy than NB by a very small margin.

**6.FUTURE WORK**

The common idea in different recommender systems is that there needs to be some measure of similarity.

The most popular ways in which recommendations are done are:

* Item-based: Finding a similarity between items, and recommending an item based on interest of a similar item. For example, if I watch an episode of *Marvel’s* *Agents of Shield*on Netflix, I would see other Marvel-related content in my TV recommendations. This is also called “Content-based filtering”. A benefit of using this approach is that you do not have to rely on a lot of user data since the similarity calculation is happening at the item level.
* Customer-based: Finding similarity between customers, and recommending an item that a similar customer has purchased/watched. This is also called “Collaborative filtering”. A benefit of this approach is that it works better for discovery, since items that look unrelated initially, might be liked by similar customers. Again taking the Netflix example, if I watch an episode of  Marvel Agent of shield ,and other people who have watched Marvel Agent of shield also watch  the office, then I would get a recommendation for the office even though the two shows seemingly have little in common.
* This system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we’d divide the data into a training set and a test set. We’d then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).

**7.REFERENCE**

WEBSITES:

https://www.researchgate.net/publication/341941415\_Movie\_Recommendation\_System\_PYTHON\_PROJECT\_REPORT/link/5eda62ae299bf1c67d41d7e3/download

https://www.w3schools.com/python/pandas/default.asp

https://www.sciencedirect.com/science/article/pii/S2666285X22000176#:~:text=Conclusion,-This%20paper%20is&text=For%20the%20Movie%20Recommendation%20System,ratings%20given%20to%20the%20movie.

|  |  |
| --- | --- |
| **PERFORMANCE (25)** |  |
| **VIVAVOCE (10)** |  |
| **MINI PROJECT (15)** |  |
| **TOTAL (50)** |  |