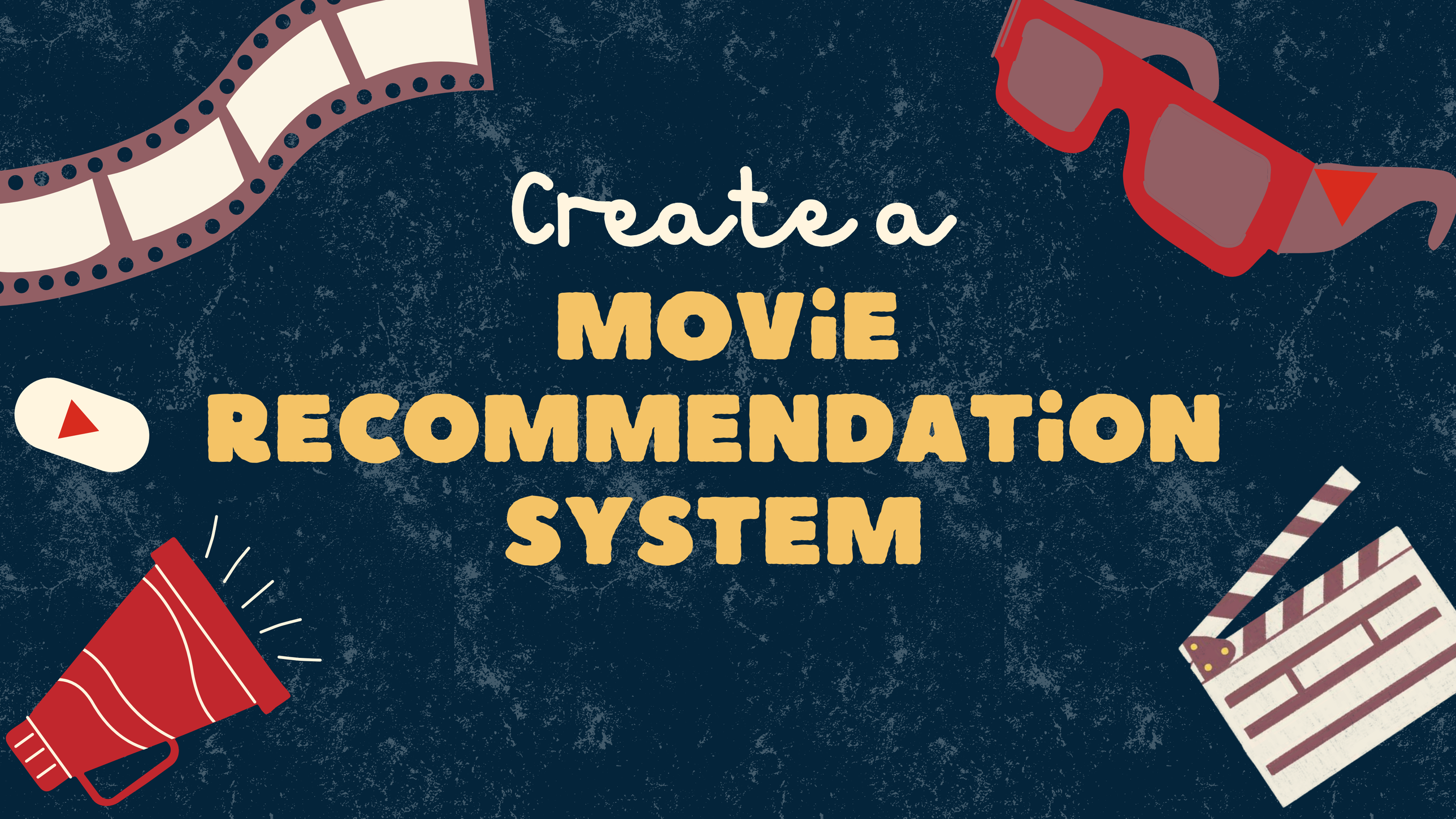

Create a
MOVIE
RECOMMENDATION
SYSTEM

TEAM MEMBERS

ARWA ABOU-ATTIA

HABIBA MOSTAFA

BASMALLA AYMAN

KARIM NASSER

AHMED SAMIR

KHALED ABOU-ELATTA

SPECIAL GREETINGS TO:

Dr. Sayed Haggag

CONTENTS

1. Talking About Recommendation systems
2. About code
3. Deployment

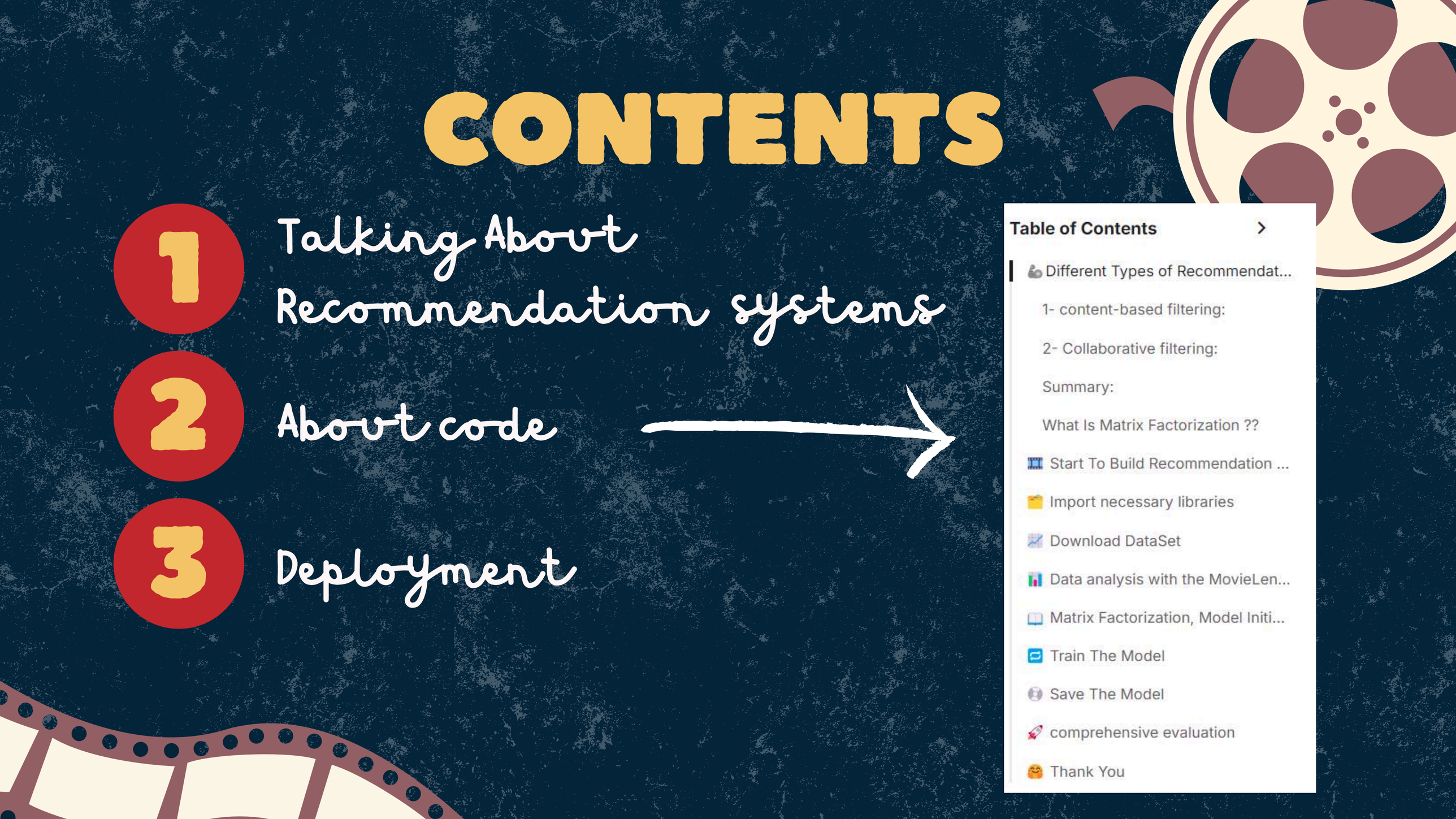


Table of Contents

- Different Types of Recommendat...
  - 1- content-based filtering:
  - 2- Collaborative filtering:
  - Summary:
  - What Is Matrix Factorization ??
- Start To Build Recommendation ...
- Import necessary libraries
- Download DataSet
- Data analysis with the MovieLen...
- Matrix Factorization, Model Initi...
- Train The Model
- Save The Model
- comprehensive evaluation
- Thank You

Talking about

MOViE RECOMMENDATiON SYSTEM

RECOMMENDATiON SYSTEM

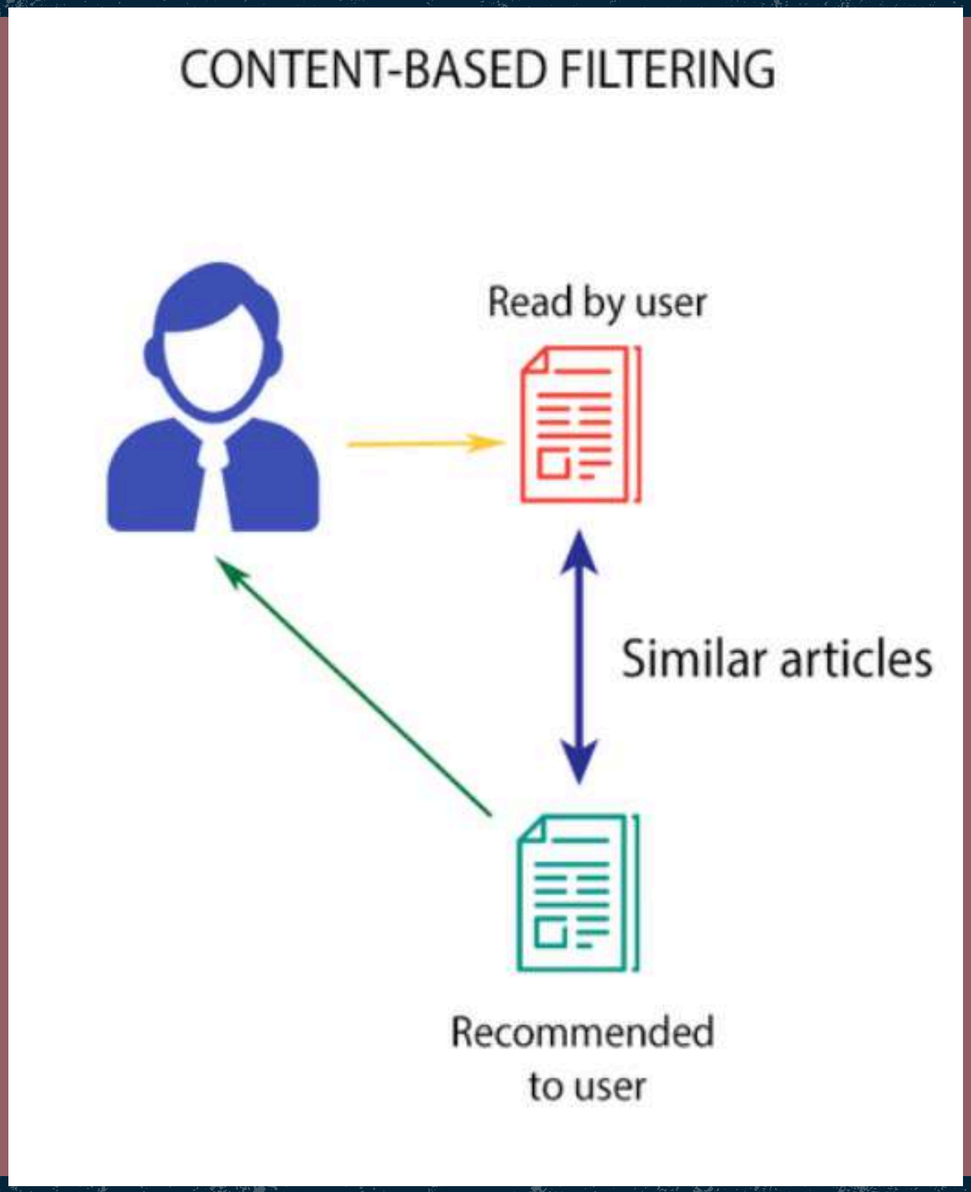
In the realm of machine learning, three predominant techniques for constructing recommendation engines are :

1. Content-based filtering.
2. Collaborative filtering.
3. Hybrid filtering.

1- CONTENT-BASED FILTERING:

This method generates suggestions based on items you have previously liked. It uses historical data such as purchase records and search history to identify similar products. For example, if you rated the movie "Inception" with five stars, the system will recommend similar movies such as "Interstellar".

However, if all recommendation systems solely relied on your viewing history, discovering new genres and films would be challenging. This is where Collaborative filtering comes into play. But what exactly does it entail?.

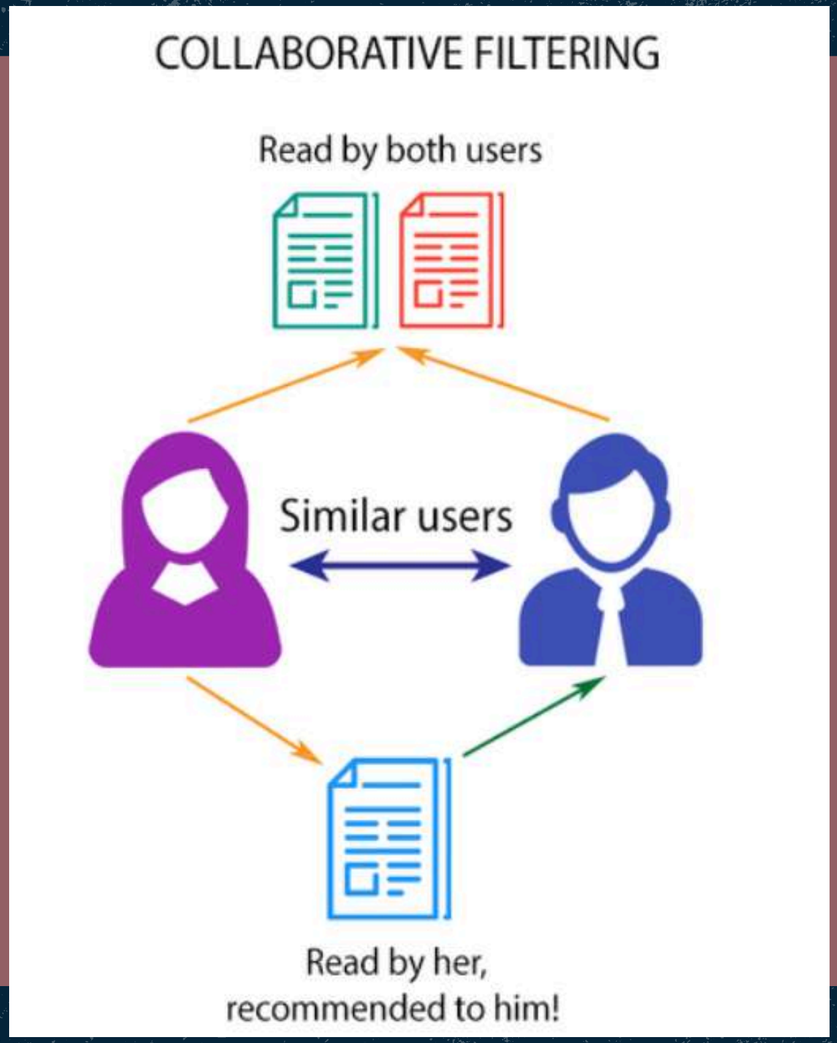

CONTENT-BASED FILTERING
Read by user
Similar articles
Recommended
to user

2- COLLABORATIVE FILTERING:

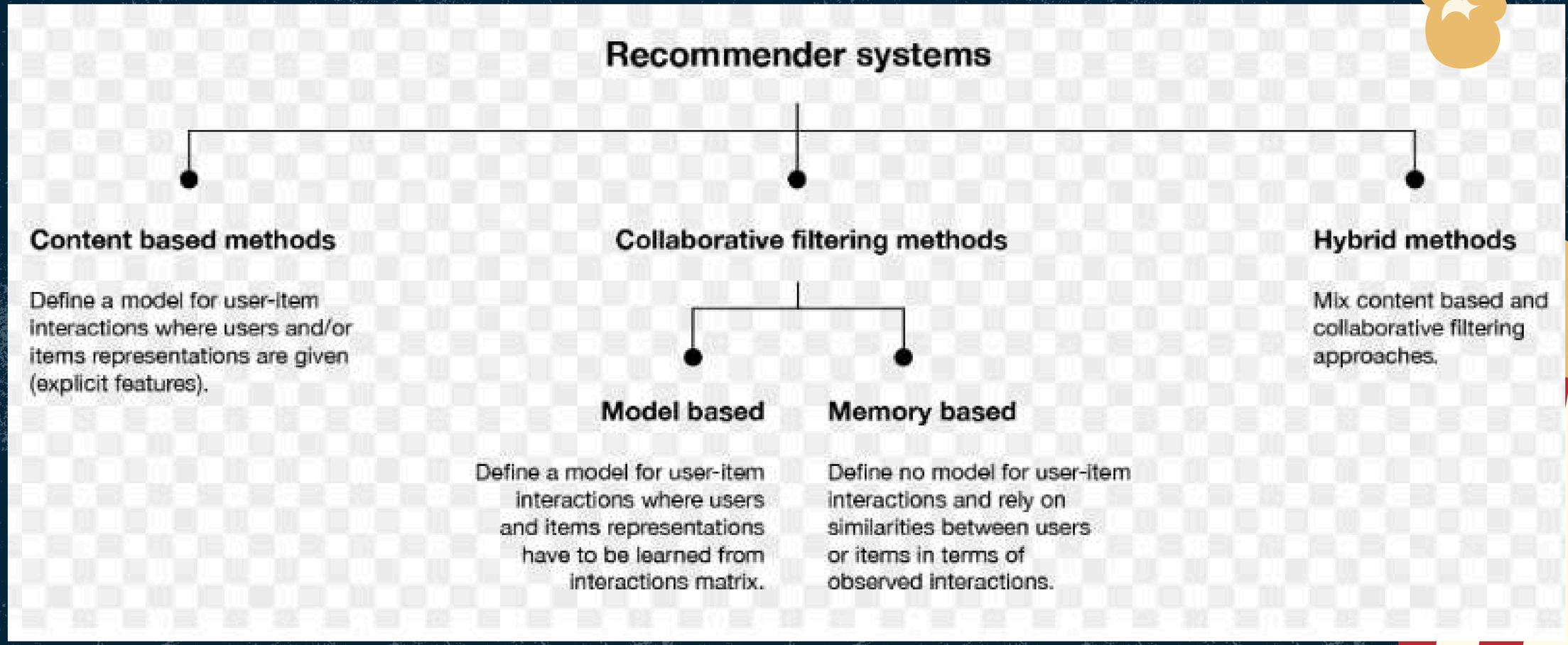
This method identifies other users who have similar preferences to you, and recommends items based on their choices.

For example, if you and your friend both like the movie "The Shawshank Redemption", and your friend also likes "Forrest Gump", the system will recommend "Forrest Gump" to you.

2- COLLABORATiVE FiLTERiNG:



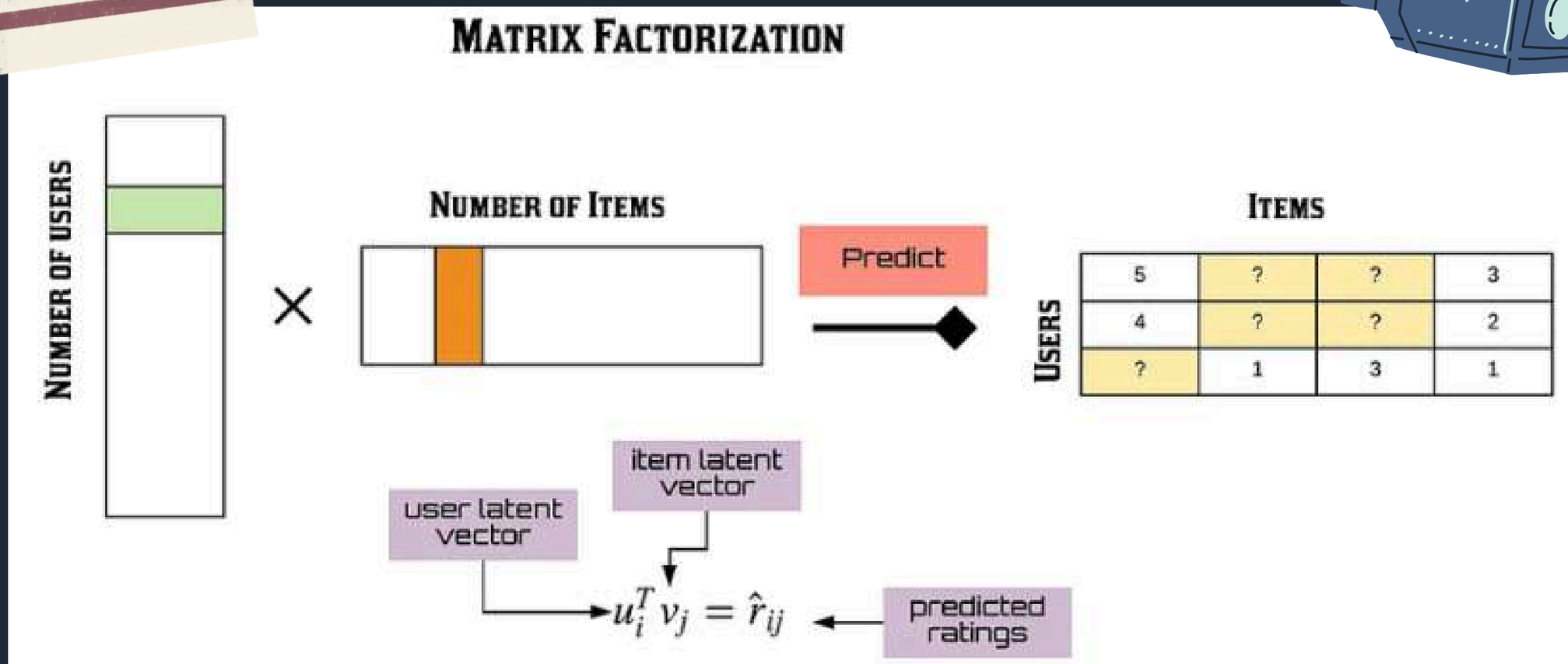
RECOMMENDATiON SYSTEM



OUR SYSTEM

We are going to attempt to create a Recommendation Systems based on collaborative filtering using matrix factorization to get all my embeddings straight and then I'm going to top it off with a k-means clustering algorithm to get my prediction to see in general how well i have done collaborative filtering .

MATRIX FACTORIZATION



BUiLD Recommendation SYSTEMS

DATASET

Download DataSet ¶

I will use `MovieLens` dataset for my project, which is a popular dataset for movie ratings. I will download it from the GroupLens website.

In [2]:
```
! curl http://files.grouplens.org/datasets/movielens/ml-latest-small.zip -o ml-latest-small.zip
```


2- Collaborative filtering:

Summary:

What Is Matrix Factorization ??

Start To Build Recommendation ...

Import necessary libraries

Download DataSet

Data analysis with the MovieLen...

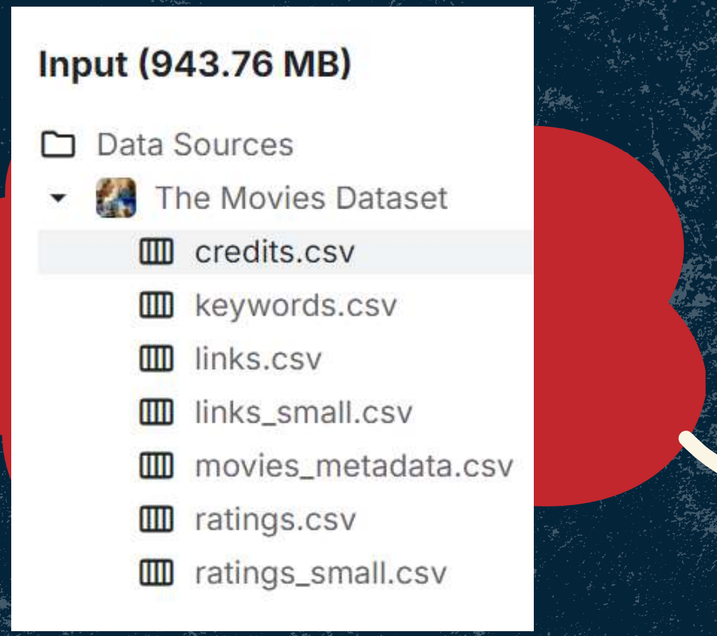
Matrix Factorization, Model Initi...

movielens

Non-commercial, personalized movie recommendations.

sign up now or sign in

RATiNGS.CSV

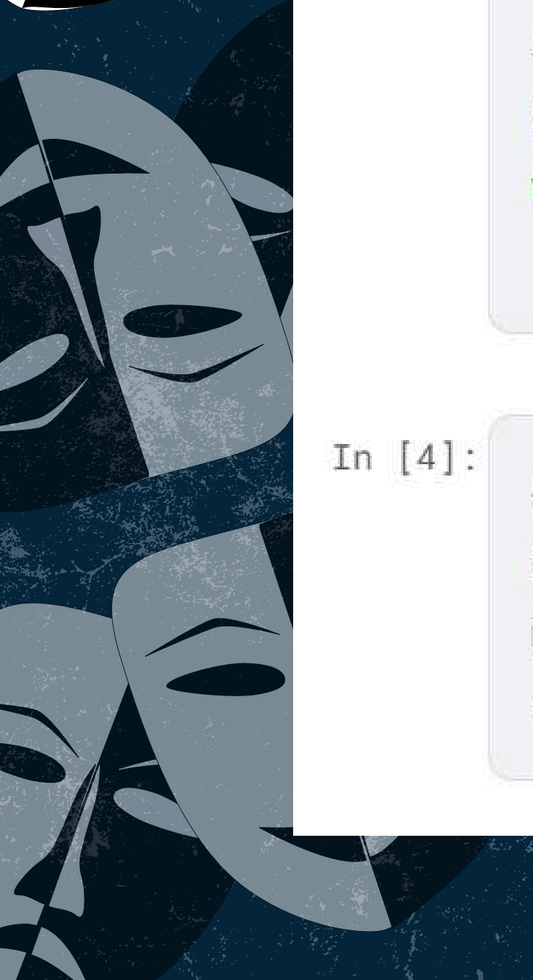


MOViES.CSV

```
# extracts the contents of the ml-latest-small.zip file to a dire
ctory named data.
# This is useful when you need to access the files inside the ZIP
archive.
import zipfile
with zipfile.ZipFile('ml-latest-small.zip', 'r') as zip_ref:
    zip_ref.extractall('data')
```

In [4]:

```
# import the dataset
import pandas as pd
movies_df = pd.read_csv('data/ml-latest-small/movies.csv')
ratings_df = pd.read_csv('data/ml-latest-small/ratings.csv')
```



DATASET

1

```
In [6]: movies_df
```

Out[6]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

2

```
In [7]: ratings_df
```

Out[7]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

SEABORN AND MATPLOTLIB: FOR VISUALIZING AND ANALYZING DATA.

PANDAS AND NUMPY: FOR DATA MANIPULATION AND ANALYSIS.

PYTORCH: FOR CREATING DEEP MODELS.

SCIKIT-LEARN: FOR MATRIX ANALYSIS AND IMPLEMENTING ALGORITHMS LIKE K-MEANS.

IMPORT NECESSARY LIBRARIES

DATA ANALYSiS

📊 Data analysis with the MovieLens dataset.

In [5]:

```
print('The dimensions of movies dataframe are:', movies_df.shape,'\nThe dimensions of ratings dataframe are:', ratings_df.shape)
```

```
The dimensions of movies dataframe are: (9742, 3)
The dimensions of ratings dataframe are: (100836, 4)
```

[6]:
```
movies_df
```

t[6]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

[7]:
```
ratings_df
```

t[7]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

NO NULL VALUES

In [8]:
```
print(movies_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   movieId  9742 non-null   int64 
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
None
```

In [9]:
```
print(ratings_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
None
```

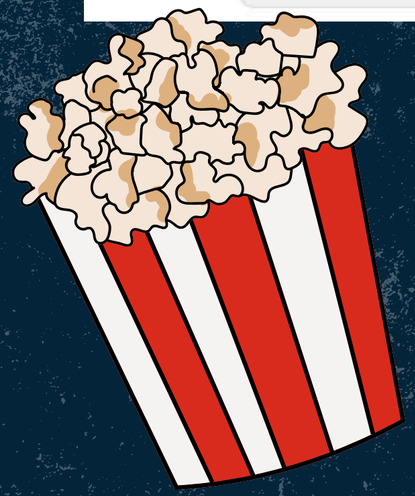
In [10]:

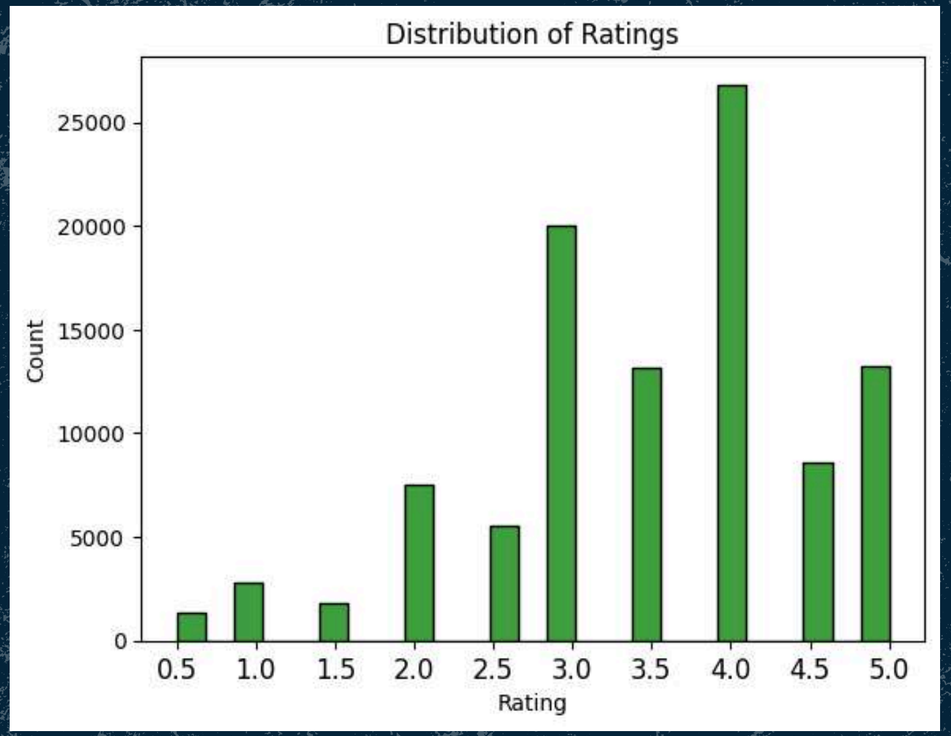
```
# Distribution of ratings
sns.histplot(ratings_df['rating'], bins=25, color='green', edge
color='black')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')

# Sort unique values
unique_ratings = np.sort(ratings_df['rating'].unique())

# Modify the label on the X axis using ordered values
plt.xticks(ticks=unique_ratings, labels=unique_ratings, fontsiz
e=12, rotation=0)

plt.show()
```





```
    'Standard deviation of ratings': ratings_df['rating'].std
(),
    'Number of active users': len(ratings_df['userId'].unique
()),
    'Average number of ratings per user': len(ratings_df) / len
(ratings_df['userId'].unique())
}

for key, value in stats.items():
    print(f"{key}: {value:.2f}")
```

```
Total number of ratings: 100836.00
Average overall rating: 3.50
Standard deviation of ratings: 1.04
Number of active users: 610.00
Average number of ratings per user: 165.30
```

```
n_users = len(ratings_df.userId.unique())
n_items = len(ratings_df.movieId.unique())

# Print the results
print("Number of unique users:", n_users)
print("Number of unique movies:", n_items)
print("The full rating matrix will have:", n_users * n_items,
'elements.')
print('----------------------------------------------------
-------------')
print("Number of ratings:", len(ratings_df))
print("Therefore: ", len(ratings_df) / (n_users * n_items) * 10
0, '% of the matrix is filled.')
```

```
Number of unique users: 610
Number of unique movies: 9724
The full rating matrix will have: 5931640 elements.
------------------------------------------------------------
```


start to
TRAiNiNG
MODEL

DATASET CLASS:

This part initializes the ratings database and converts it into a form that can be processed using PyTorch. It encodes user and movie IDs and converts ratings into tensors.

In [14]:

```
# 1. Define Dataset
class MovieDataset(Dataset):
    def __init__(self, ratings_df, user_to_idx, movie_to_idx):
        self.users = torch.tensor([user_to_idx[user] for user i
n ratings_df['userId']], dtype=torch.long)
        self.movies = torch.tensor([movie_to_idx[movie] for mov
ie in ratings_df['movieId']], dtype=torch.long)
        self.ratings = torch.tensor(ratings_df['rating'].value
s, dtype=torch.float)

    def __len__(self):
        return len(self.ratings)

    def __getitem__(self, idx):
        return self.users[idx], self.movies[idx], self.ratings
[idx]
```

Building a collaborative filtering model:

The following part represents the mathematical model used to analyze matrices, where we rely on User & Item Embeddings to predict missing ratings:

In [15]:

```python
# 2. Define Matrix Factorization model
class MatrixFactorization(nn.Module):
    def __init__(self, n_users, n_movies, n_factors=50):
        super().__init__()
        self.user_factors = nn.Embedding(n_users, n_factors)
        self.movie_factors = nn.Embedding(n_movies, n_factors)
        self.user_biases = nn.Embedding(n_users, 1)
        self.movie_biases = nn.Embedding(n_movies, 1)

    def forward(self, user, movie):
        user_embedding = self.user_factors(user)
        movie_embedding = self.movie_factors(movie)
        user_bias = self.user_biases(user)
        movie_bias = self.movie_biases(movie)

        prediction = (user_embedding * movie_embedding).sum(dim=1, keepdim=True)
        prediction = prediction + user_bias + movie_bias
        return prediction.squeeze()

    def get_embeddings(self, user, movie):
        """Extract user and movie embeddings"""
        user_embedding = self.user_factors(user)
        movie_embedding = self.movie_factors(movie)
        return user_embedding, movie_embedding
```

HOW THE MODEL WORKS:

Every time we pass in the “user ID” and the “movie”, we get the hidden vector for each. We use the inner product (Dot Product) of these vectors to predict the expected rating.

(OPTiMiZATiON TECHNiQUES)

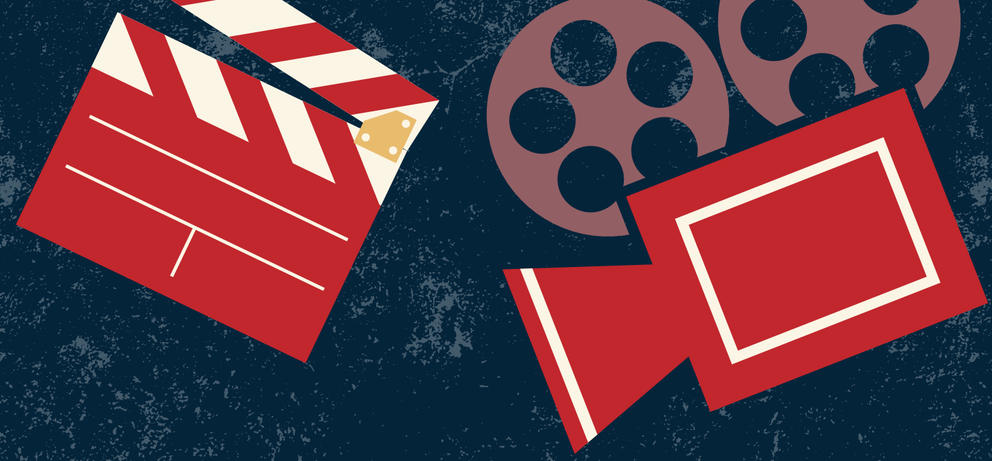
We use the Stochastic Gradient Descent (SGD) algorithm to update the vectors based on the errors computed from the predictions:

```
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

We calculate the error between the predicted and actual valuation using a cost function such as MSE and then update the coefficients based on this error.

```
"""Train Matrix Factorization model"""
criterion = nn.MSELoss()
```

K-MEANS CLUSTERiNG :



After the matrix analysis and rating prediction are done, the K-means algorithm is used to cluster the movies based on the similarity in ratings. This helps in classifying the movies into groups that have similar characteristics based on the users’ preferences.

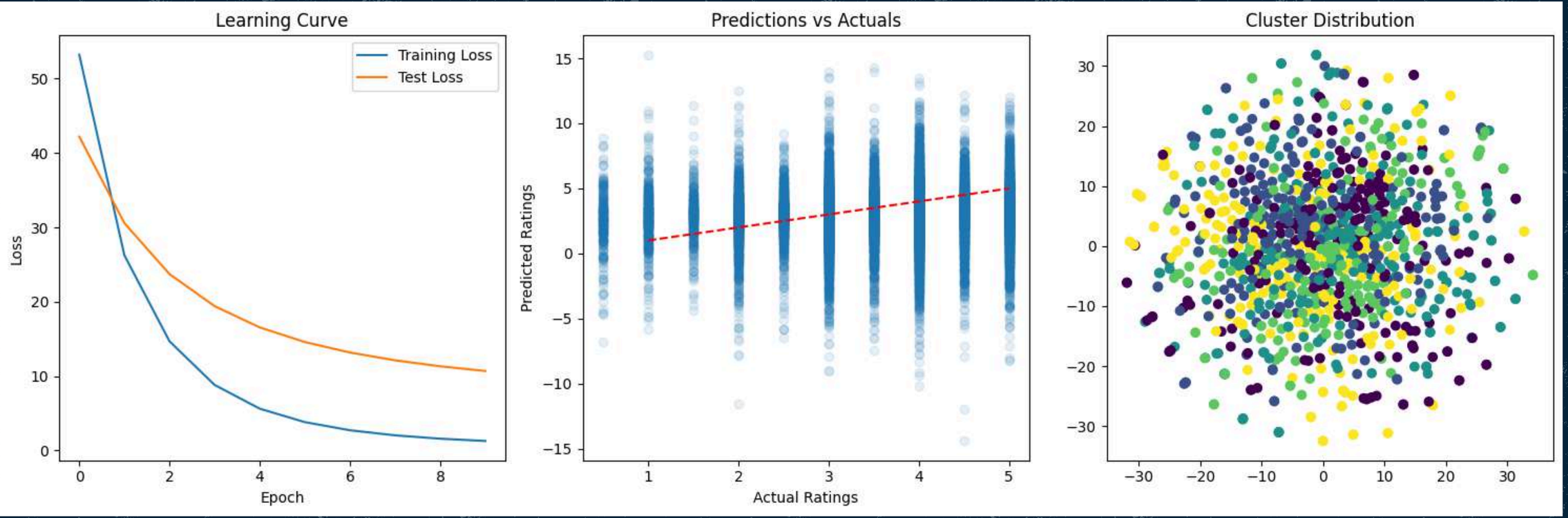
SUMMARY OF WHAT WAS EXPLAINED:

1. We used collaborative filtering based on matrix analysis to recommend movies.

2. The large matrix was divided into smaller matrices representing user and movie preferences, with performance optimization techniques such as Gradient Descent used to improve accuracy.

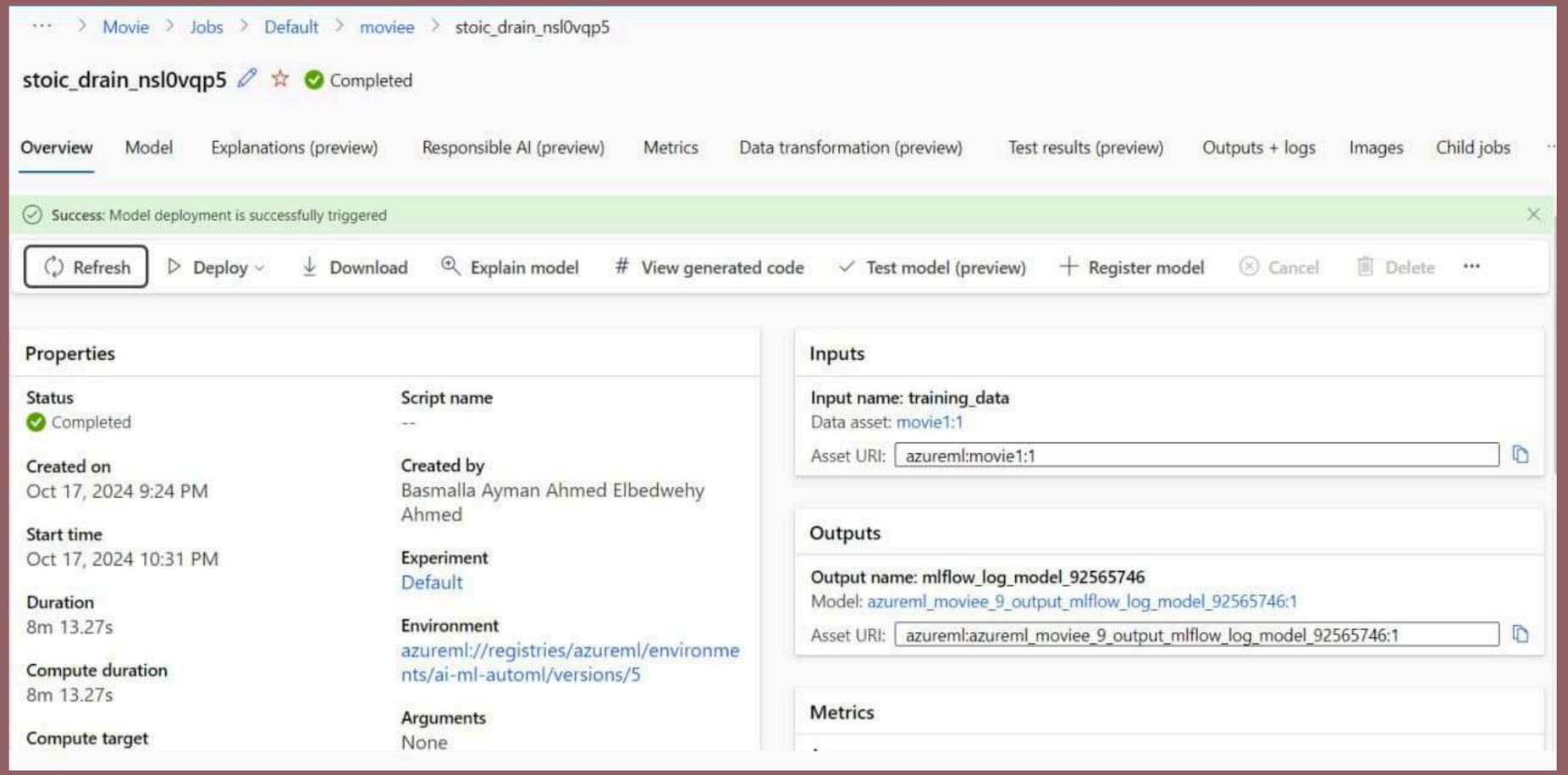
3. The model was evaluated using MSE and MAE.

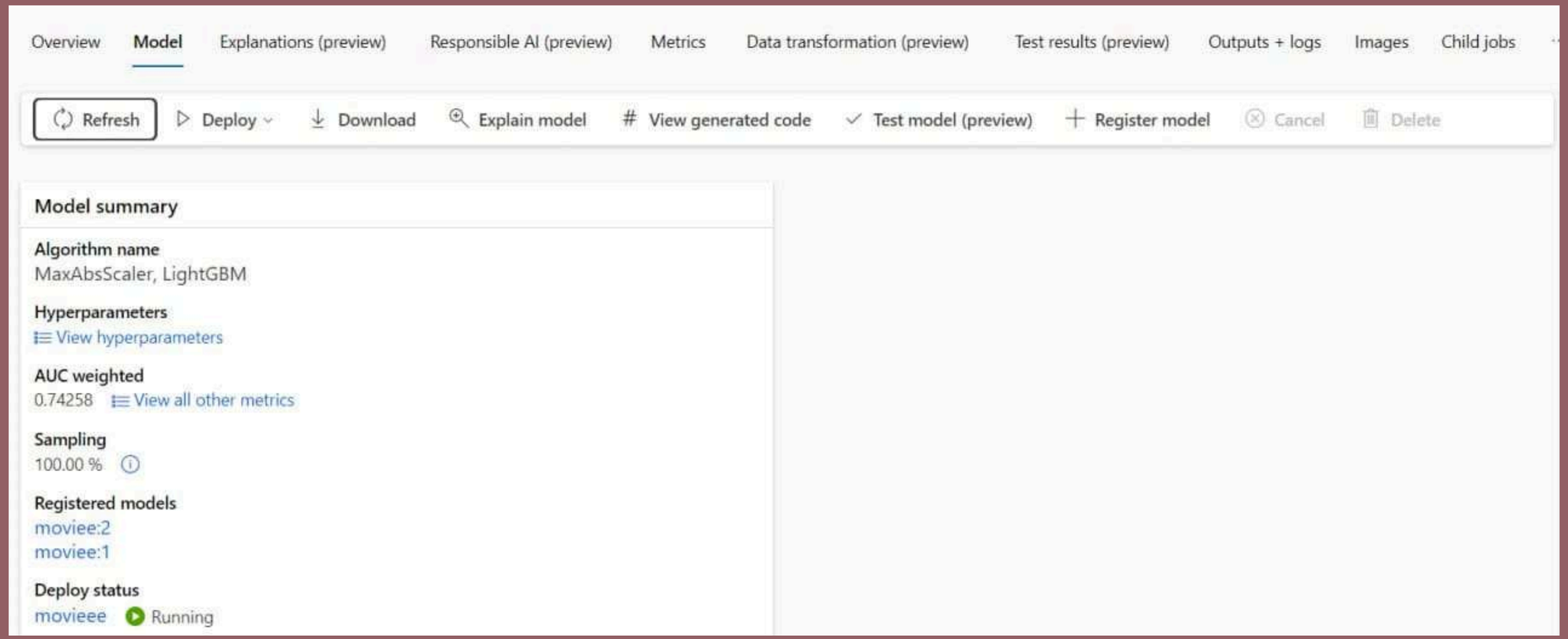
4. We used K-means to cluster movies based on similarity in ratings.


Learning Curve
Training Loss
Test Loss
Loss
Epoch
Predictions vs Actuals
Predicted Ratings
Actual Ratings
Cluster Distribution

end of proj

MODEL DEPLOYMENT


Movie > Jobs > Default > moviee > stoic_drain_nsl0vqp5
stoic_drain_nsl0vqp5
Completed
Overview
Model
Explanations (preview)
Responsible AI (preview)
Metrics
Data transformation (preview)
Test results (preview)
Outputs + logs
Images
Child jobs
Success: Model deployment is successfully triggered
Refresh
Deploy
Download
Explain model
View generated code
Test model (preview)
Register model
Cancel
Delete
Properties
Status
Completed
Created on
Oct 17, 2024 9:24 PM
Start time
Oct 17, 2024 10:31 PM
Duration
8m 13.27s
Compute duration
8m 13.27s
Compute target
Script name
--
Created by
Basmalla Ayman Ahmed Elbedwehy Ahmed
Experiment
Default
Environment
azureml://registries/azureml/environments/ai-ml-automl/versions/5
Arguments
None
Inputs
Input name: training_data
Data asset: movie1:1
Asset URI: azureml:movie1:1
Outputs
Output name: mlflow_log_model_92565746
Model: azureml_moviee_9_output_mlflow_log_model_92565746:1
Asset URI: azureml:azureml_moviee_9_output_mlflow_log_model_92565746:1
Metrics


Overview
Model
Explanations (preview)
Responsible AI (preview)
Metrics
Data transformation (preview)
Test results (preview)
Outputs + logs
Images
Child jobs
Refresh
Deploy
Download
Explain model
View generated code
Test model (preview)
Register model
Cancel
Delete
Model summary
Algorithm name
MaxAbsScaler, LightGBM
Hyperparameters
View hyperparameters
AUC weighted
0.74258
View all other metrics
Sampling
100.00 %
Registered models
moviee:2
moviee:1
Deploy status
movieee
Running

THE END