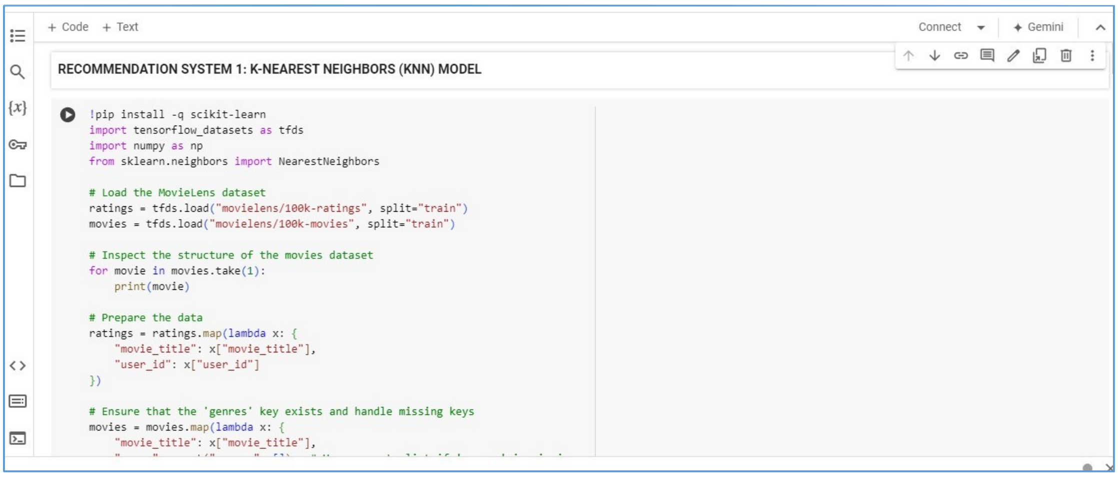
HARPY AEROSPACE INTERNSHIP

AIOT PROJECT – Recommendation system

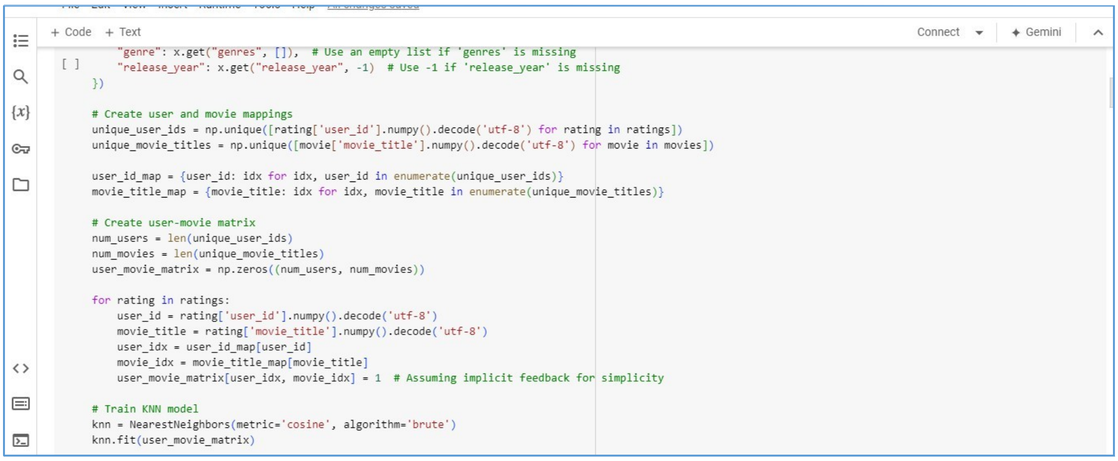
By:
Haritha K
2022506014
B.Tech (Information Technology)
Madras Institute Of Technology

RECOMMENTATION SYSTEM 1 - K-NEAREST NEIGHBORS MODEL (KNN)

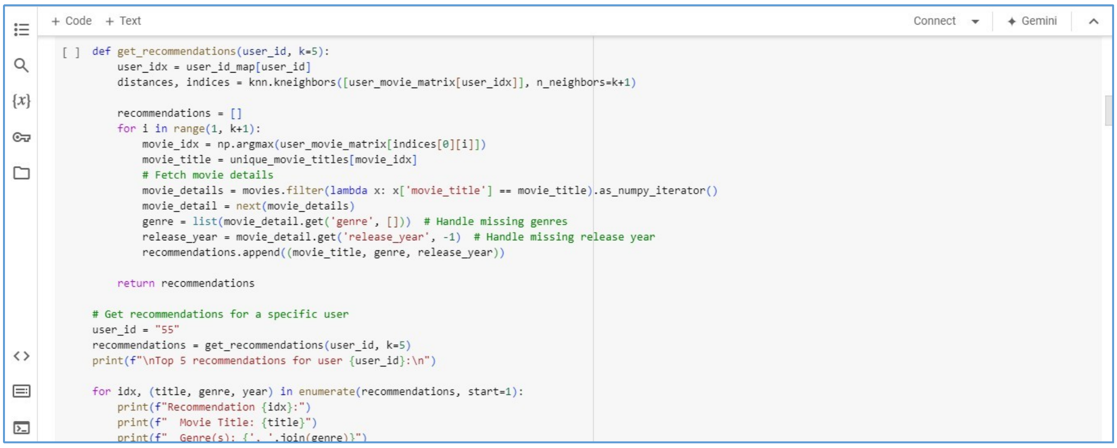


RECOMMENDATION SYSTEM 1: K-NEAREST NEIGHBORS (KNN) MODEL

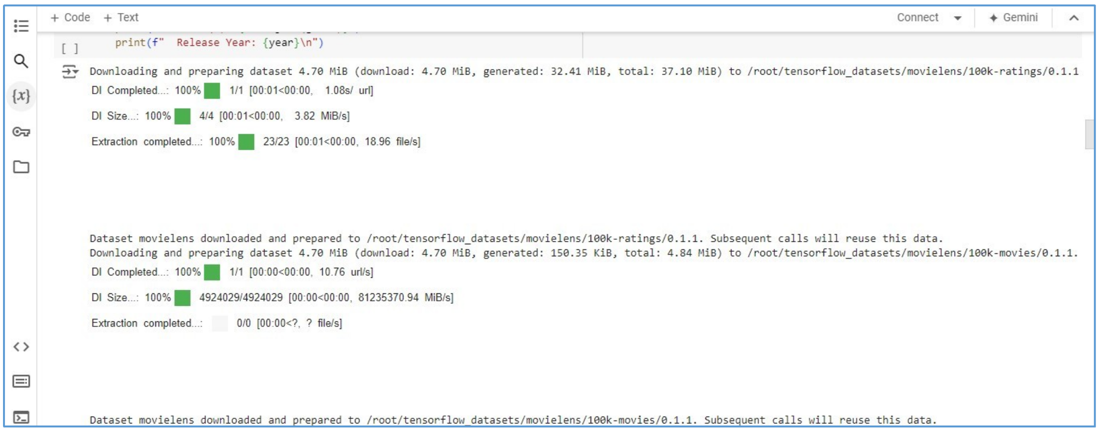
```
!pip install -q scikit-learn
import tensorflow_datasets as tfds
import numpy as np
from sklearn.neighbors import NearestNeighbors

# Load the MovieLens dataset
ratings = tfds.load("movielens/100k-ratings", split="train")
movies = tfds.load("movielens/100k-movies", split="train")

# Inspect the structure of the movies dataset
for movie in movies.take(1):
    print(movie)

# Prepare the data
ratings = ratings.map(lambda x: {
    "movie_title": x["movie_title"],
    "user_id": x["user_id"]
})

# Ensure that the 'genres' key exists and handle missing keys
movies = movies.map(lambda x: {
    "movie_title": x["movie_title"],
```



```
    "genre": x.get("genres", []),  # Use an empty list if 'genres' is missing
    "release_year": x.get("release_year", -1)  # Use -1 if 'release_year' is missing
})

# Create user and movie mappings
unique_user_ids = np.unique([rating['user_id'].numpy().decode('utf-8') for rating in ratings])
unique_movie_titles = np.unique([movie['movie_title'].numpy().decode('utf-8') for movie in movies])

user_id_map = {user_id: idx for idx, user_id in enumerate(unique_user_ids)}
movie_title_map = {movie_title: idx for idx, movie_title in enumerate(unique_movie_titles)}

# Create user-movie matrix
num_users = len(unique_user_ids)
num_movies = len(unique_movie_titles)
user_movie_matrix = np.zeros((num_users, num_movies))

for rating in ratings:
    user_id = rating['user_id'].numpy().decode('utf-8')
    movie_title = rating['movie_title'].numpy().decode('utf-8')
    user_idx = user_id_map[user_id]
    movie_idx = movie_title_map[movie_title]
    user_movie_matrix[user_idx, movie_idx] = 1  # Assuming implicit feedback for simplicity

# Train KNN model
knn = NearestNeighbors(metric='cosine', algorithm='brute')
knn.fit(user_movie_matrix)
```

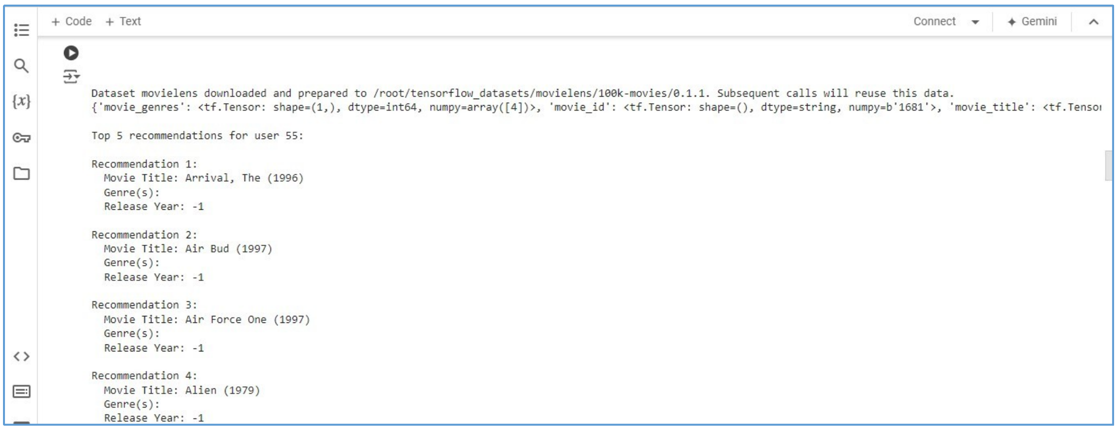


```
def get_recommendations(user_id, k=5):
    user_idx = user_id_map[user_id]
    distances, indices = knn.kneighbors([user_movie_matrix[user_idx]], n_neighbors=k+1)

    recommendations = []
    for i in range(1, k+1):
        movie_idx = np.argmax(user_movie_matrix[indices[0][i]])
        movie_title = unique_movie_titles[movie_idx]
        # Fetch movie details
        movie_details = movies.filter(lambda x: x['movie_title'] == movie_title).as_numpy_iterator()
        movie_detail = next(movie_details)
        genre = list(movie_detail.get('genre', []))  # Handle missing genres
        release_year = movie_detail.get('release_year', -1)  # Handle missing release year
        recommendations.append((movie_title, genre, release_year))

    return recommendations

# Get recommendations for a specific user
user_id = "55"
recommendations = get_recommendations(user_id, k=5)
print(f"\nTop 5 recommendations for user {user_id}:\n")

for idx, (title, genre, year) in enumerate(recommendations, start=1):
    print(f"Recommendation {idx}:")
    print(f"  Movie Title: {title}")
    print(f"  Genre(s): {', '.join(genre)}")
```

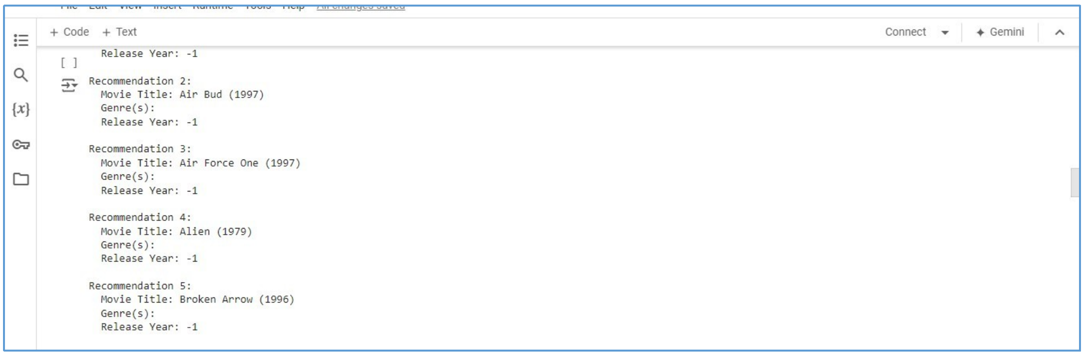


```
    print(f"  Release Year: {year}\n")
```

```
Downloading and preparing dataset 4.70 MiB (download: 4.70 MiB, generated: 32.41 MiB, total: 37.10 MiB) to /root/tensorflow_datasets/movielens/100k-ratings/0.1.1
 Dl Completed...: 100% 1/1 [00:01<00:00,  1.08s/ url]

 Dl Size...: 100% 4/4 [00:01<00:00,  3.82 MiB/s]

 Extraction completed...: 100% 23/23 [00:01<00:00, 18.96 file/s]



Dataset movielens downloaded and prepared to /root/tensorflow_datasets/movielens/100k-ratings/0.1.1. Subsequent calls will reuse this data.
Downloading and preparing dataset 4.70 MiB (download: 4.70 MiB, generated: 150.35 KiB, total: 4.84 MiB) to /root/tensorflow_datasets/movielens/100k-movies/0.1.1.
 Dl Completed...: 100% 1/1 [00:00<00:00, 10.76 url/s]

 Dl Size...: 100% 4924029/4924029 [00:00<00:00, 81235370.94 MiB/s]

 Extraction completed...:  0/0 [00:00<?, ? file/s]



Dataset movielens downloaded and prepared to /root/tensorflow_datasets/movielens/100k-movies/0.1.1. Subsequent calls will reuse this data.
```

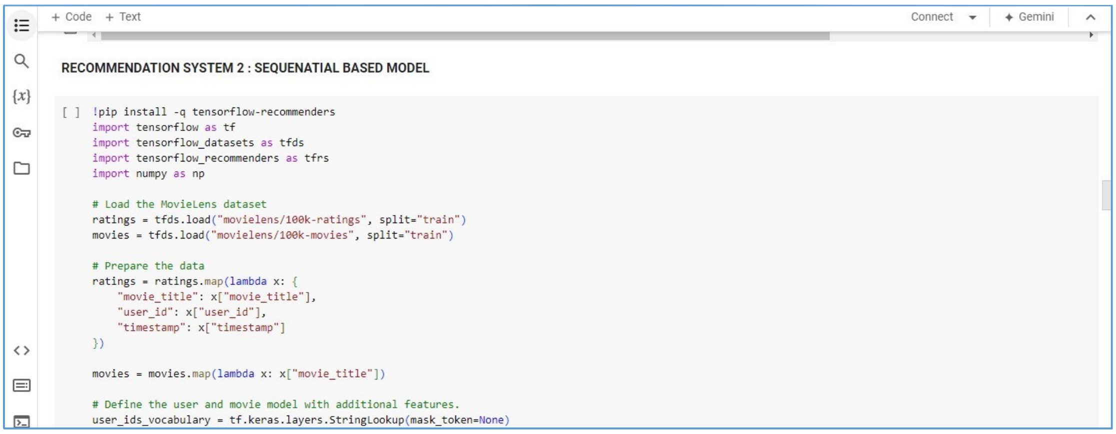


```
Dataset movielens downloaded and prepared to /root/tensorflow_datasets/movielens/100k-movies/0.1.1. Subsequent calls will reuse this data.
{'movie_genres': <tf.Tensor: shape=(1,), dtype=int64, numpy=array([4])>, 'movie_id': <tf.Tensor: shape=(), dtype=string, numpy=b'1681'>, 'movie_title': <tf.Tensoı

Top 5 recommendations for user 55:

Recommendation 1:
  Movie Title: Arrival, The (1996)
  Genre(s):
  Release Year: -1

Recommendation 2:
  Movie Title: Air Bud (1997)
  Genre(s):
  Release Year: -1

Recommendation 3:
  Movie Title: Air Force One (1997)
  Genre(s):
  Release Year: -1

Recommendation 4:
  Movie Title: Alien (1979)
  Genre(s):
  Release Year: -1
```



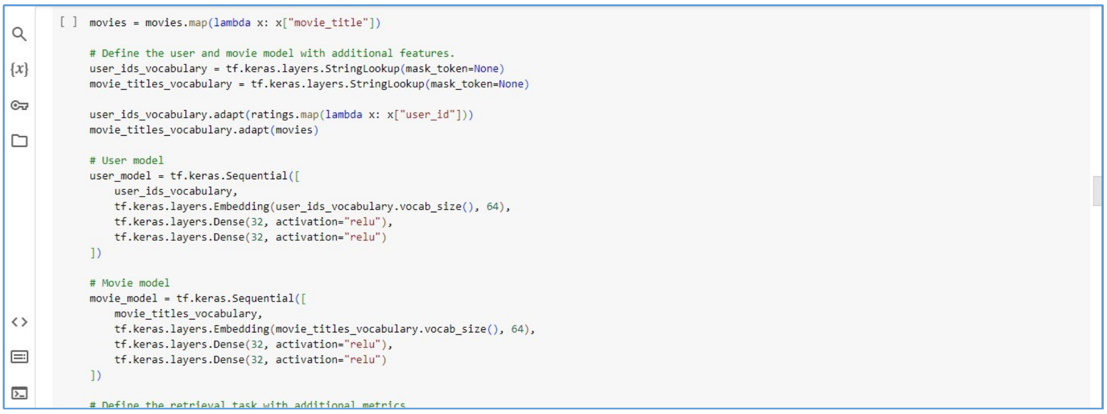
```
  Release Year: -1

Recommendation 2:
  Movie Title: Air Bud (1997)
  Genre(s):
  Release Year: -1

Recommendation 3:
  Movie Title: Air Force One (1997)
  Genre(s):
  Release Year: -1

Recommendation 4:
  Movie Title: Alien (1979)
  Genre(s):
  Release Year: -1

Recommendation 5:
  Movie Title: Broken Arrow (1996)
  Genre(s):
  Release Year: -1
```

RECOMMENDATION SYSTEM 2 - SEQUENATIAL BASED MODEL

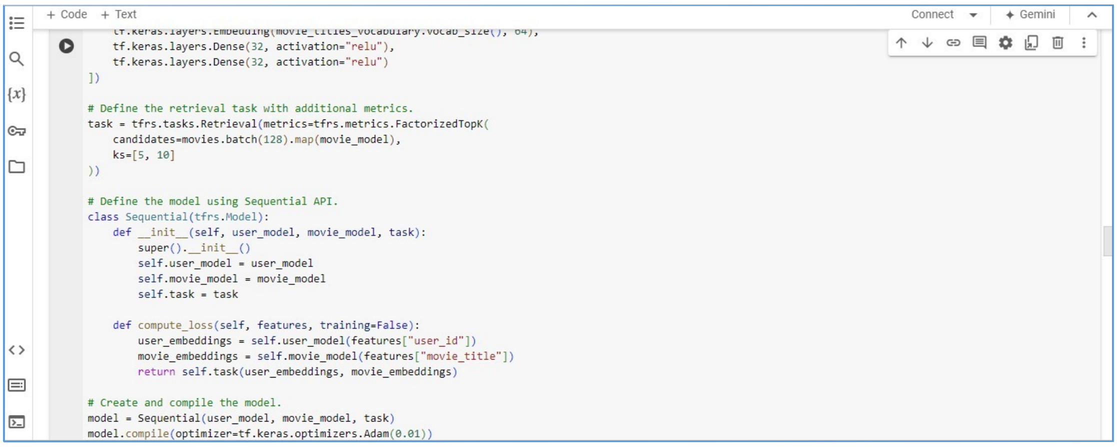


RECOMMENDATION SYSTEM 2 : SEQUENATIAL BASED MODEL

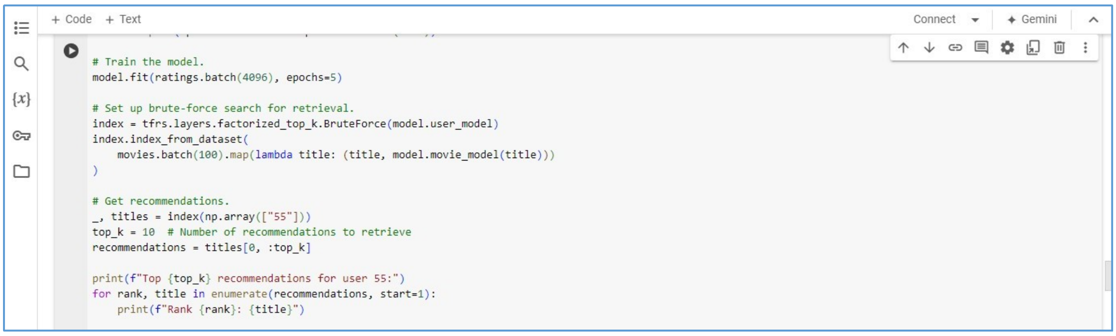
```
[ ] !pip install -q tensorflow-recommenders
    import tensorflow as tf
    import tensorflow_datasets as tfds
    import tensorflow_recommenders as tfrs
    import numpy as np

    # Load the MovieLens dataset
    ratings = tfds.load("movielens/100k-ratings", split="train")
    movies = tfds.load("movielens/100k-movies", split="train")

    # Prepare the data
    ratings = ratings.map(lambda x: {
        "movie_title": x["movie_title"],
        "user_id": x["user_id"],
        "timestamp": x["timestamp"]
    })

    movies = movies.map(lambda x: x["movie_title"])

    # Define the user and movie model with additional features.
    user_ids_vocabulary = tf.keras.layers.StringLookup(mask_token=None)
```



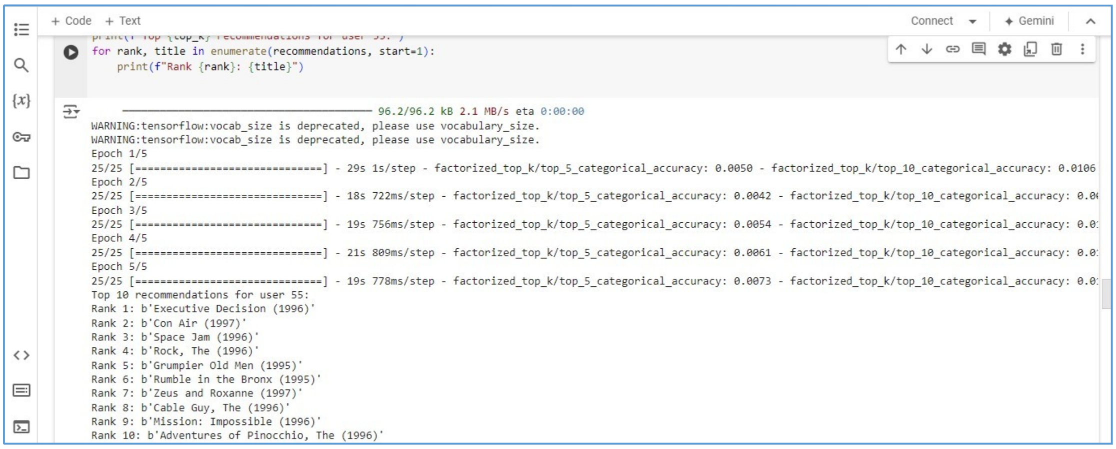
```
[ ] movies = movies.map(lambda x: x["movie_title"])

    # Define the user and movie model with additional features.
    user_ids_vocabulary = tf.keras.layers.StringLookup(mask_token=None)
    movie_titles_vocabulary = tf.keras.layers.StringLookup(mask_token=None)

    user_ids_vocabulary.adapt(ratings.map(lambda x: x["user_id"]))
    movie_titles_vocabulary.adapt(movies)

    # User model
    user_model = tf.keras.Sequential([
        user_ids_vocabulary,
        tf.keras.layers.Embedding(user_ids_vocabulary.vocab_size(), 64),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(32, activation="relu")
    ])

    # Movie model
    movie_model = tf.keras.Sequential([
        movie_titles_vocabulary,
        tf.keras.layers.Embedding(movie_titles_vocabulary.vocab_size(), 64),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(32, activation="relu")
    ])

    # Define the retrieval task with additional metrics
```



```
    tf.keras.layers.Embedding(movie_titles_vocabulary.vocab_size(), 64),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dense(32, activation="relu")
])

# Define the retrieval task with additional metrics.
task = tfrs.tasks.Retrieval(metrics=tfrs.metrics.FactorizedTopK(
    candidates=movies.batch(128).map(movie_model),
    ks=[5, 10]
))

# Define the model using Sequential API.
class Sequential(tfrs.Model):
    def __init__(self, user_model, movie_model, task):
        super().__init__()
        self.user_model = user_model
        self.movie_model = movie_model
        self.task = task

    def compute_loss(self, features, training=False):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        return self.task(user_embeddings, movie_embeddings)

# Create and compile the model.
model = Sequential(user_model, movie_model, task)
model.compile(optimizer=tf.keras.optimizers.Adam(0.01))
```



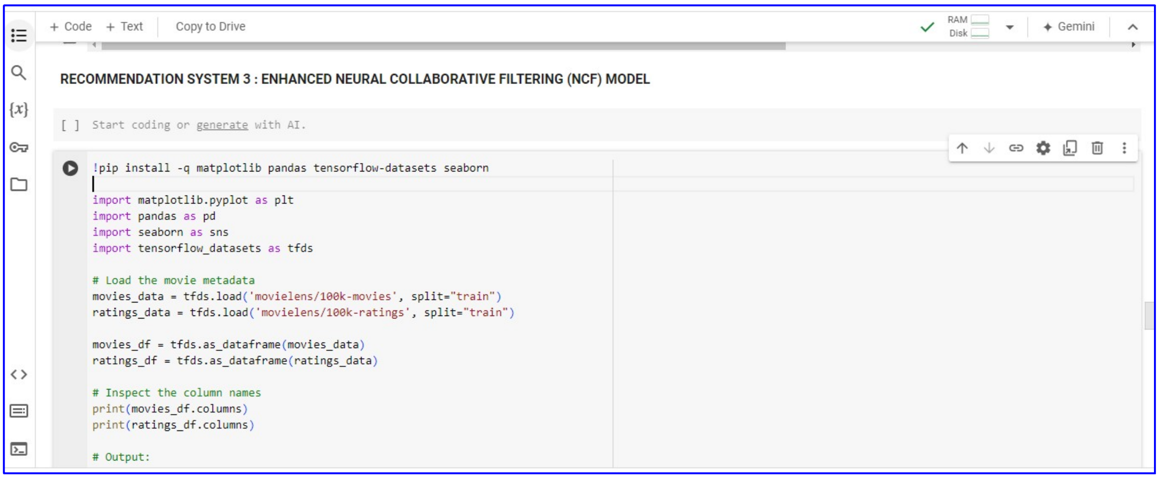
```
# Train the model.
model.fit(ratings.batch(4096), epochs=5)

# Set up brute-force search for retrieval.
index = tfrs.layers.factorized_top_k.BruteForce(model.user_model)
index.index_from_dataset(
    movies.batch(100).map(lambda title: (title, model.movie_model(title)))
)

# Get recommendations.
_, titles = index(np.array(["55"]))
top_k = 10  # Number of recommendations to retrieve
recommendations = titles[0, :top_k]

print(f"Top {top_k} recommendations for user 55:")
for rank, title in enumerate(recommendations, start=1):
    print(f"Rank {rank}: {title}")
```



```
print(f"Top {top_k} recommendations for user 55:")
for rank, title in enumerate(recommendations, start=1):
    print(f"Rank {rank}: {title}")
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 96.2/96.2 kB 2.1 MB/s eta 0:00:00
WARNING:tensorflow:vocab_size is deprecated, please use vocabulary_size.
WARNING:tensorflow:vocab_size is deprecated, please use vocabulary_size.
Epoch 1/5
25/25 [==============================] - 29s 1s/step - factorized_top_k/top_5_categorical_accuracy: 0.0050 - factorized_top_k/top_10_categorical_accuracy: 0.0106
Epoch 2/5
25/25 [==============================] - 18s 722ms/step - factorized_top_k/top_5_categorical_accuracy: 0.0042 - factorized_top_k/top_10_categorical_accuracy: 0.0
Epoch 3/5
25/25 [==============================] - 19s 756ms/step - factorized_top_k/top_5_categorical_accuracy: 0.0054 - factorized_top_k/top_10_categorical_accuracy: 0.0
Epoch 4/5
25/25 [==============================] - 21s 809ms/step - factorized_top_k/top_5_categorical_accuracy: 0.0061 - factorized_top_k/top_10_categorical_accuracy: 0.0
Epoch 5/5
25/25 [==============================] - 19s 778ms/step - factorized_top_k/top_5_categorical_accuracy: 0.0073 - factorized_top_k/top_10_categorical_accuracy: 0.0
Top 10 recommendations for user 55:
Rank 1: b'Executive Decision (1996)'
Rank 2: b'Con Air (1997)'
Rank 3: b'Space Jam (1996)'
Rank 4: b'Rock, The (1996)'
Rank 5: b'Grumpier Old Men (1995)'
Rank 6: b'Rumble in the Bronx (1995)'
Rank 7: b'Zeus and Roxanne (1997)'
Rank 8: b'Cable Guy, The (1996)'
Rank 9: b'Mission: Impossible (1996)'
Rank 10: b'Adventures of Pinocchio, The (1996)'
```

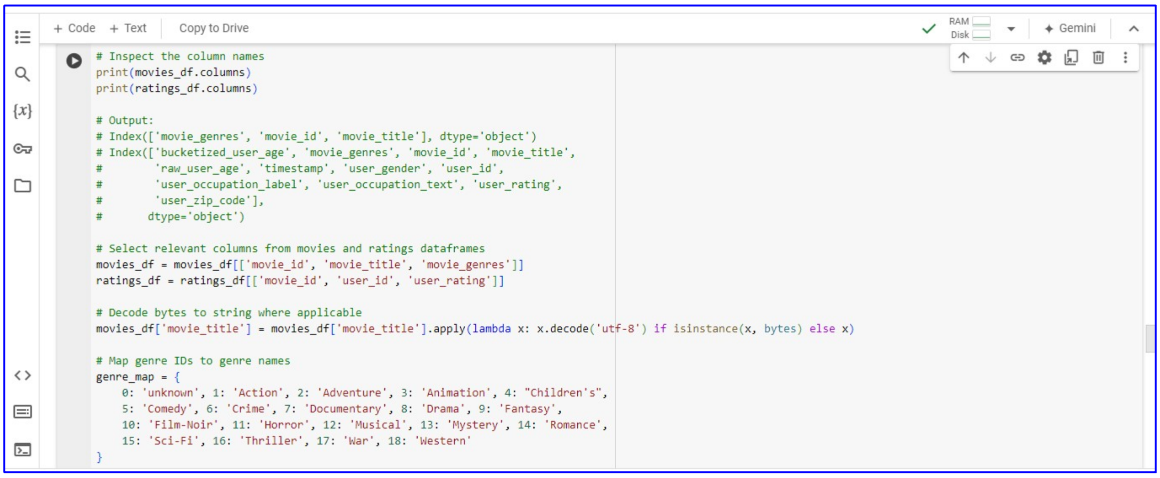
RECOMMENDATION SYSTEM 3 - ENHANCED NEURAL COLLABORATIVE FILTERING (NCF) MODEL



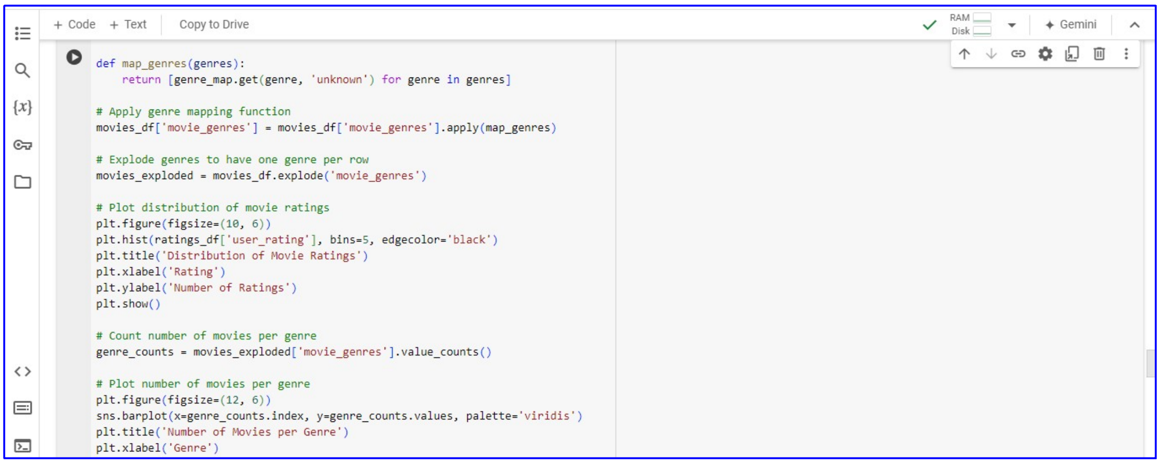
RECOMMENDATION SYSTEM 3 : ENHANCED NEURAL COLLABORATIVE FILTERING (NCF) MODEL

```
[ ]  Start coding or generate with AI.
```

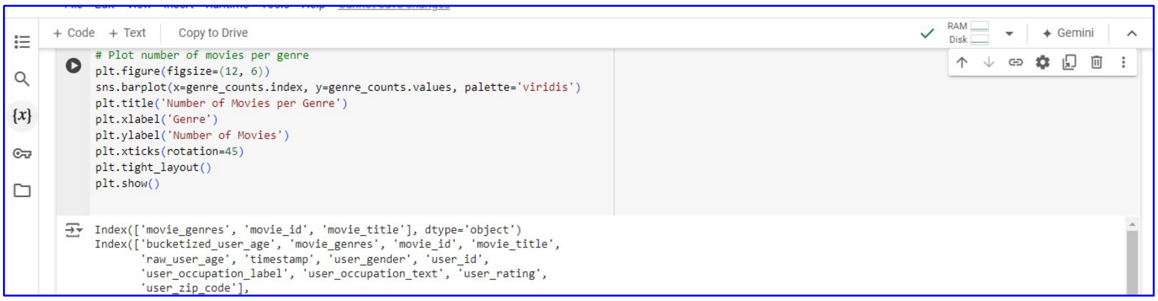
```
!pip install -q matplotlib pandas tensorflow-datasets seaborn

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow_datasets as tfds

# Load the movie metadata
movies_data = tfds.load('movielens/100k-movies', split="train")
ratings_data = tfds.load('movielens/100k-ratings', split="train")

movies_df = tfds.as_dataframe(movies_data)
ratings_df = tfds.as_dataframe(ratings_data)

# Inspect the column names
print(movies_df.columns)
print(ratings_df.columns)

# Output:
```



```
# Inspect the column names
print(movies_df.columns)
print(ratings_df.columns)

# Output:
# Index(['movie_genres', 'movie_id', 'movie_title'], dtype='object')
# Index(['bucketized_user_age', 'movie_genres', 'movie_id', 'movie_title',
#        'raw_user_age', 'timestamp', 'user_gender', 'user_id',
#        'user_occupation_label', 'user_occupation_text', 'user_rating',
#        'user_zip_code'],
#       dtype='object')

# Select relevant columns from movies and ratings dataframes
movies_df = movies_df[['movie_id', 'movie_title', 'movie_genres']]
ratings_df = ratings_df[['movie_id', 'user_id', 'user_rating']]

# Decode bytes to string where applicable
movies_df['movie_title'] = movies_df['movie_title'].apply(lambda x: x.decode('utf-8') if isinstance(x, bytes) else x)

# Map genre IDs to genre names
genre_map = {
    0: 'unknown', 1: 'Action', 2: 'Adventure', 3: 'Animation', 4: "Children's",
    5: 'Comedy', 6: 'Crime', 7: 'Documentary', 8: 'Drama', 9: 'Fantasy',
    10: 'Film-Noir', 11: 'Horror', 12: 'Musical', 13: 'Mystery', 14: 'Romance',
    15: 'Sci-Fi', 16: 'Thriller', 17: 'War', 18: 'Western'
}
```

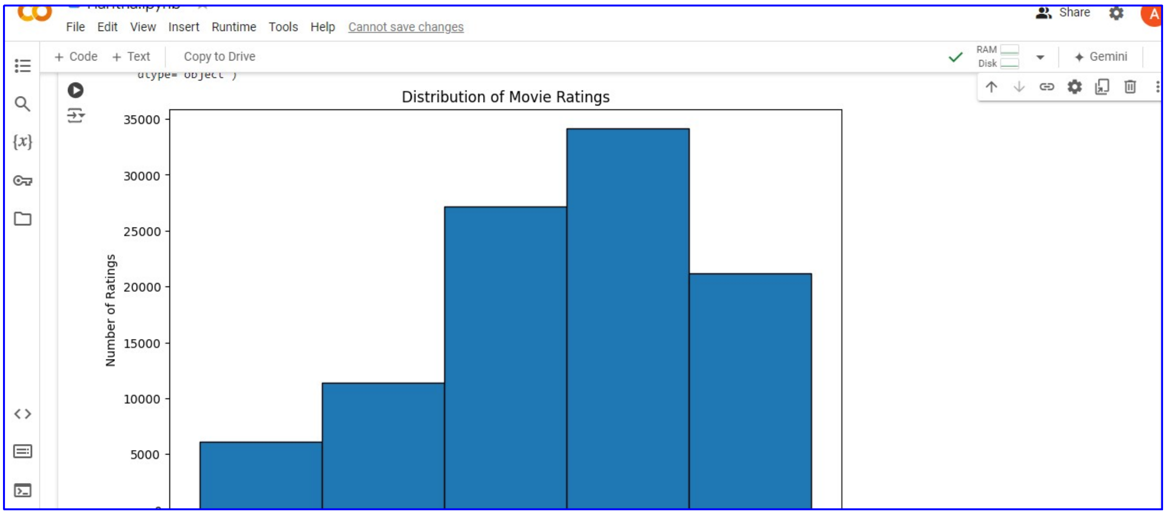


```
def map_genres(genres):
    return [genre_map.get(genre, 'unknown') for genre in genres]

# Apply genre mapping function
movies_df['movie_genres'] = movies_df['movie_genres'].apply(map_genres)

# Explode genres to have one genre per row
movies_exploded = movies_df.explode('movie_genres')

# Plot distribution of movie ratings
plt.figure(figsize=(10, 6))
plt.hist(ratings_df['user_rating'], bins=5, edgecolor='black')
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Number of Ratings')
plt.show()

# Count number of movies per genre
genre_counts = movies_exploded['movie_genres'].value_counts()

# Plot number of movies per genre
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_counts.index, y=genre_counts.values, palette='viridis')
plt.title('Number of Movies per Genre')
plt.xlabel('Genre')
```

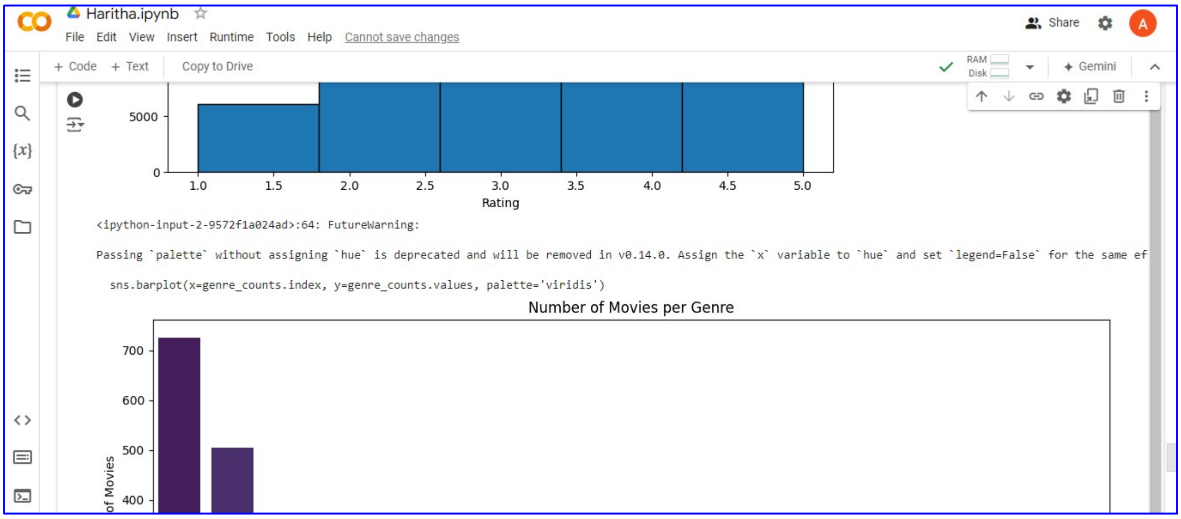


```
# Plot number of movies per genre
plt.figure(figsize=(12, 6))
sns.barplot(x=genre_counts.index, y=genre_counts.values, palette='viridis')
plt.title('Number of Movies per Genre')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
Index(['movie_genres', 'movie_id', 'movie_title'], dtype='object')
Index(['bucketized_user_age', 'movie_genres', 'movie_id', 'movie_title',
       'raw_user_age', 'timestamp', 'user_gender', 'user_id',
       'user_occupation_label', 'user_occupation_text', 'user_rating',
       'user_zip_code'],
```

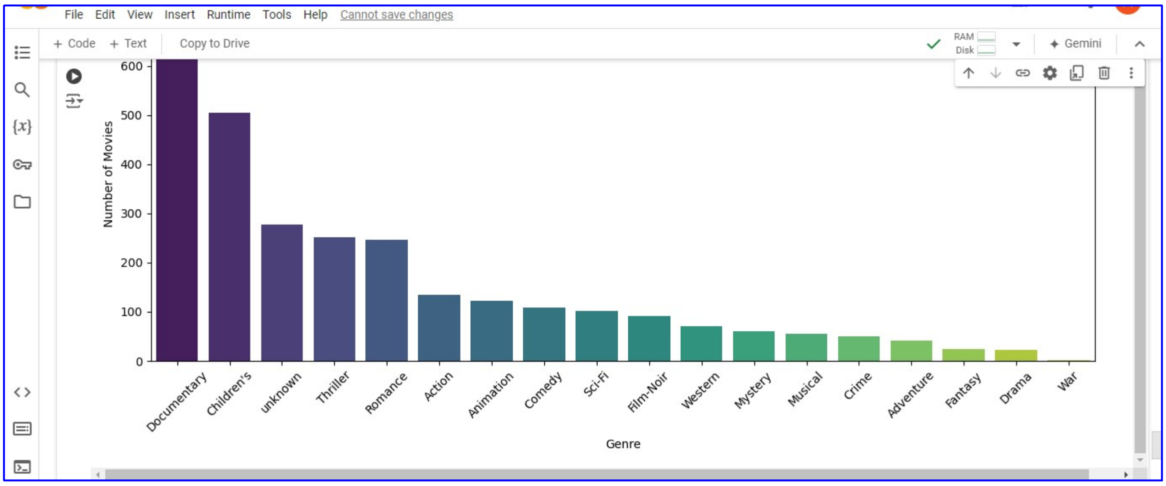

Haritha.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Gemini
Distribution of Movie Ratings
Number of Ratings
35000
30000
25000
20000
15000
10000
5000

```
dtype= object )
```


Haritha.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Gemini
5000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Rating

```
<ipython-input-2-9572f1a024ad>:64: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same ef

  sns.barplot(x=genre_counts.index, y=genre_counts.values, palette='viridis')
```

Number of Movies per Genre
of Movies
700
600
500
400


File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
RAM Disk
Gemini
Number of Movies
600
500
400
300
200
100
0
Documentary
Children's
unknown
Thriller
Romance
Action
Animation
Comedy
Sci-Fi
Film-Noir
Western
Mystery
Musical
Crime
Adventure
Fantasy
Drama
War
Genre