Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
//Load Libraries
import pandas as pd # for data processing
import numpy as np # for numerical computation
import matplotlib.pyplot as plt  # for data visualization
import seaborn as sns # for visualization analysis
from datetime import datetime # for data time data

# Set Seaborn style
sns.set(style="whitegrid")

# Load the dataset
file_path = '/content/netflix.csv'
netflix_data = pd.read_csv(file_path)


# Display the first few rows to ensure data is loaded correctly
print(netflix_data.head())

# Display column data types
print(netflix_data.dtypes)

//Data Preprocessing

from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np

# Handling missing values (imputing with mean for numerical and most frequent for categorical)
imputer = SimpleImputer(strategy='mean')

# Handling the 'Premiere' date column and converting it to datetime
netflix_data['premiere'] = pd.to_datetime(netflix_data['premiere'], errors='coerce')

# Dropping rows where 'Premiere' cannot be parsed
netflix_data.dropna(subset=['premiere'], inplace=True)

# One-hot encoding categorical data (genre, language)
encoder = OneHotEncoder(sparse=False)
encoded_data = pd.DataFrame(encoder.fit_transform(netflix_data[['genre', 'language']]))
encoded_data.columns = encoder.get_feature_names_out(['genre', 'language'])

# Normalizing numerical data (imdb_score, runtime)
scaler = StandardScaler()

scaled_data = pd.DataFrame(scaler.fit_transform(netflix_data[['imdb_score', 'runtime']]),
columns=['imdb_score_scaled', 'runtime_scaled'])
```

Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
# Combine the encoded and scaled data back with the original dataset
netflix_data.reset_index(drop=True, inplace=True)
final_data = pd.concat([netflix_data[['title', 'premiere', 'year']], encoded_data, scaled_data], axis=1)

# Display the preprocessed data
print(final_data.head())


# Extract year, month, and day from Premiere date

final_data['Premiere_year'] = final_data['premiere'].dt.year
final_data['Premiere_month'] = final_data['premiere'].dt.month
final_data['Premiere_day'] = final_data['premiere'].dt.day

# Time since premiere (how many years since the release)
from datetime import datetime
current_year = datetime.now().year
final_data['years_since_release'] = current_year - final_data['year']

# Drop unnecessary columns (Premiere, year, etc.) from the final dataset
final_data.drop(columns=['premiere', 'year'], inplace=True)

# Display the new features
print(final_data.head())


# Display basic statistical analysis of numerical columns
print("Statistical Summary of Numerical Columns:")
print(netflix_data.describe())


# Display basic information about dataset (for data types and missing values)
print("\nDataset Information:")
print(netflix_data.info())


### Visualization 1: Distribution of IMDb Scores

plt.figure(figsize=(10, 6))
sns.histplot(netflix_data['imdb_score'], kde=True, bins=20, color='blue')
plt.title('Distribution of IMDb Scores')
plt.xlabel('IMDb Score')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
### Visualization 2: Distribution of Runtime (in minutes)
plt.figure(figsize=(10, 6))
sns.histplot(netflix_data['runtime'], kde=True, bins=20, color='green')
plt.title('Distribution of Movie/TV Show Runtime')
plt.xlabel('Runtime (minutes)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()


### Visualization 3: Count of Movies/TV Shows by Genre
plt.figure(figsize=(15, 10))
sns.countplot(y='genre', data=netflix_data, order=netflix_data['genre'].value_counts().index,
palette='viridis')
plt.title('Count of Movies/TV Shows by Genre')
plt.xlabel('Count')
plt.ylabel('Genre')
plt.grid(True)
plt.show()


### Visualization 4: Count of Movies/TV Shows by Language

plt.figure(figsize=(10, 6))
sns.countplot(y='language', data=netflix_data, order=netflix_data['language'].value_counts().index,
palette='coolwarm')
plt.title('Count of Movies/TV Shows by Language')
plt.xlabel('Count')
plt.ylabel('Language')
plt.grid(True)
plt.show()


### Visualization 5: IMDb Score vs Runtime

plt.figure(figsize=(15, 6))
sns.scatterplot(x='runtime', y='imdb_score', data=netflix_data, hue='genre', palette='deep', alpha=0.7)
plt.title('IMDb Score vs Runtime by Genre')
plt.xlabel('Runtime (minutes)')
plt.ylabel('IMDb Score')
plt.grid(True)
plt.show()
```

```
### Visualization 6: Number of Releases Per Year

plt.figure(figsize=(10, 6))
sns.histplot(final_data['Premiere_year'], bins=30, kde=False, color='purple')
plt.title('Number of Releases Per Year')
plt.xlabel('Premiere Year')
plt.ylabel('Number of Releases')
plt.grid(True)
plt.show()


### Visualization 7: Boxplot of IMDb Score by Genre

plt.figure(figsize=(12, 6))
sns.boxplot(x='imdb_score', y='genre', data=netflix_data, palette='pastel')
plt.title('IMDb Score Distribution by Genre')
plt.xlabel('IMDb Score')
plt.ylabel('Genre')
plt.grid(True)
plt.show()


### Visualization 8: Heatmap of Correlation Matrix (Numerical Columns)

plt.figure(figsize=(8, 6))
correlation_matrix = final_data[['imdb_score', 'runtime', 'Premiere_year']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Numerical Columns')
plt.show()


### Visualization 9: IMDb Score Trends Over Time (Yearly Average IMDb Score)
yearly_avg_imdb = final_data.groupby('Premiere_year')['imdb_score'].mean()
plt.figure(figsize=(10, 6))
sns.lineplot(x=yearly_avg_imdb.index, y=yearly_avg_imdb.values, color='red', marker='o')
plt.title('Yearly Average IMDb Score Over Time')
plt.xlabel('Premiere Year')
plt.ylabel('Average IMDb Score')
plt.grid(True)
plt.show()


### Visualization 10: Distribution of Movies/TV Shows by Premiere Month

plt.figure(figsize=(10, 6))
```

```
sns.countplot(x='Premiere_month', data=final_data, palette='plasma')
plt.title('Number of Movies/TV Shows Released by Month')
plt.xlabel('Premiere Month')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```

//Ridge Regression Model

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Define feature matrix X and target vector y
X = final_data.drop(columns=['imdb_score_scaled', 'title'])
y = final_data['imdb_score_scaled']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Ridge Regression model
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ridge_model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error for Ridge Regression: {mse}")
```

//Applying Ridge regression Model

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score


# Load the Netflix dataset

netflix_data = pd.read_csv('/content/netflix.csv')
```

Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
# Data preprocessing

# Convert Premiere date to datetime and extract year
netflix_data['premiere'] = pd.to_datetime(netflix_data['premiere'], errors='coerce')
netflix_data.dropna(subset=['premiere'], inplace=True)
netflix_data['premiere_year'] = netflix_data['premiere'].dt.year

# One-hot encode categorical features (genre, language)
encoder = OneHotEncoder(sparse=False)
encoded_features = encoder.fit_transform(netflix_data[['genre', 'language']])

# Scale numerical features (runtime, Premiere_year)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(netflix_data[['runtime', 'premiere_year']])

# Combine one-hot encoded and scaled features into a single matrix
import numpy as np
features = np.hstack((encoded_features, scaled_features))

# Define target variable (IMDb score)
target = netflix_data['imdb_score']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Apply Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Predict the IMDb score for the test set
y_pred = ridge_model.predict(X_test)

# Evaluate the model: Calculate mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Visualization: True vs Predicted IMDb Scores
def visualize_true_vs_predicted(y_test, y_pred):

    plt.figure(figsize=(10, 6))

    sns.scatterplot(x=y_test, y=y_pred, alpha=0.6, color='blue')

    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')  # Diagonal
line

    plt.title('True vs Predicted IMDb Scores (Ridge Regression)')
```

```
    plt.xlabel('True IMDb Scores')
    plt.ylabel('Predicted IMDb Scores')
    plt.grid(True)
    plt.show()

# Call the function to visualize True vs Predicted IMDb Scores
visualize_true_vs_predicted(y_test, y_pred)

# Visualization: Residual Plot (Errors between true and predicted IMDb scores)
def visualize_residuals(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(10, 6))
    sns.histplot(residuals, kde=True, color='purple')
    plt.title('Residuals (True - Predicted IMDb Scores)')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()

# Call the function to visualize Residuals
visualize_residuals(y_test, y_pred)

import statsmodels.api as sm

# Group by Premiere_year and calculate average IMDb score per year
yearly_trend = final_data.groupby('Premiere_year')['imdb_score_scaled'].mean()

# Apply ARIMA model to forecast future IMDb score trends
arima_model = sm.tsa.ARIMA(yearly_trend, order=(1, 1, 1))
arima_result = arima_model.fit()

# Forecast for the next 5 years
forecast = arima_result.forecast(steps=5)
print(f"Forecast for next 5 years: {forecast}")
```

Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
# Content-based filtering: Calculate cosine similarity between movies based on features
similarity_matrix = cosine_similarity(X)

# Function to recommend similar movies based on content similarity
def recommend_content_based(movie_title, top_n=10):

    movie_idx = final_data.index[final_data['title'] == movie_title].tolist()[0]

    similarity_scores = list(enumerate(similarity_matrix[movie_idx]))

    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)

    top_movies_idx = [i[0] for i in similarity_scores[1:top_n+1]]  # Exclude the movie itself

    return final_data.iloc[top_movies_idx][['title', 'imdb_score_scaled']]


# Example usage: Recommend similar movies to a given title
recommendations = recommend_content_based(movie_title='The Lovebirds')
print(recommendations)


from sklearn.metrics import precision_score, recall_score


# Ridge Regression MSE is already calculated


# Example evaluation for recommendation (precision and recall for relevance)

# Assuming y_true is the list of relevant items for a user, and y_pred are the recommended items
y_true = [1, 0, 1, 1, 0, 0]  # Example ground truth
y_pred = [1, 1, 1, 0, 0, 1]  # Example predictions

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)

print(f"Precision: {precision}, Recall: {recall}")


from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid for Ridge Regression
param_grid = {'alpha': [0.1, 1.0, 10.0, 100.0]}

# Initialize GridSearchCV
grid_search = GridSearchCV(Ridge(), param_grid, cv=5)

grid_search.fit(X_train, y_train)
```

Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
# Best hyperparameters
print(f"Best alpha: {grid_search.best_params_}")
```

```
pip install scikit-surprise
```

```
import pandas as pd
from surprise import Reader, Dataset, SVD
from surprise.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Load the Netflix dataset (assuming it contains 'user_id', 'movie_id', and 'rating')
netflix_data = pd.read_csv('/content/netflix.csv')

# For collaborative filtering, we need user interaction data (let's assume we have user ratings for movies)

# Sample dataset format: user_id, movie_id, rating

# For this code, we will simulate user ratings by assuming that 'imdb_score' is a proxy for user ratings (real-world datasets would contain explicit user ratings)

netflix_data['user_id'] = pd.factorize(netflix_data['title'])[0]  # Simulating user interactions
netflix_data['movie_id'] = pd.factorize(netflix_data['title'])[0]  # Assigning a unique movie_id
netflix_data['rating'] = netflix_data['imdb_score']  # Assuming imdb_score as user rating

# Prepare data for Surprise library
reader = Reader(rating_scale=(1, 10))  # Assuming IMDb score range is 1 to 10
data = Dataset.load_from_df(netflix_data[['user_id', 'movie_id', 'rating']], reader)

# Split the data into train and test sets
trainset, testset = train_test_split(data, test_size=0.2)

# Initialize the SVD model
model = SVD()

# Train the model on the trainset
model.fit(trainset)

# Predict the ratings for the test set
predictions = model.test(testset)
```

```
# Visualize predictions
def visualize_predictions(predictions):

   # Extract true and predicted ratings
   true_ratings = [pred.r_ui for pred in predictions]
   predicted_ratings = [pred.est for pred in predictions]

   # Plot the true vs predicted ratings
   plt.figure(figsize=(10, 6))
   sns.scatterplot(x=true_ratings, y=predicted_ratings, alpha=0.5)
   plt.title('True vs Predicted Ratings')
   plt.xlabel('True Ratings')
   plt.ylabel('Predicted Ratings')
  plt.grid(True)
   plt.show()


# Call the function to visualize true vs predicted ratings
visualize_predictions(predictions)

# Visualizing Recommendations

def recommend_movies_for_user(user_id, top_n=10):
   # Predict ratings for all movies for a given user
   movie_ids = netflix_data['movie_id'].unique()
   predictions = [model.predict(user_id, movie_id) for movie_id in movie_ids]

   # Sort movies by predicted rating
   sorted_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)

   # Get top N recommended movies
   top_recommendations = sorted_predictions[:top_n]

   # Print out the movie titles
   recommended_movie_ids = [pred.iid for pred in top_recommendations]
   recommended_movies = netflix_data[netflix_data['movie_id'].isin(recommended_movie_ids)]
   print("Top {} recommendations for User ID {}:".format(top_n, user_id))
   print(recommended_movies[['title', 'genre', 'imdb_score']].drop_duplicates())


# Example: Recommend top 10 movies for user_id = 1
recommend_movies_for_user(user_id=1, top_n=10)
```

Python code developed for MSc. Project on “Enhanced Prediction and Recommendation Systems using Netflix Viewing Data” by Khadiza Akter

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Load the Netflix dataset
netflix_data = pd.read_csv('/content/netflix.csv')

# Data preprocessing
# Convert Premiere date to datetime
netflix_data['premiere'] = pd.to_datetime(netflix_data['premiere'], errors='coerce')
netflix_data.dropna(subset=['premiere'], inplace=True)

# One-hot encode categorical features (genre, language)
encoder = OneHotEncoder(sparse=False)
encoded_features = encoder.fit_transform(netflix_data[['genre', 'language']])

# Scale numerical features (runtime, IMDb score)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(netflix_data[['runtime', 'imdb_score']])

# Combine one-hot encoded features and scaled features into a single matrix
import numpy as np
features = np.hstack((encoded_features, scaled_features))

# Cosine similarity between movies based on metadata
similarity_matrix = cosine_similarity(features)

# Function to get top N similar movies based on cosine similarity
def recommend_content_based(movie_title, top_n=10):

    # Get the index of the movie that matches the title
    movie_idx = netflix_data[netflix_data['title'] == movie_title].index[0]

    # Get similarity scores for this movie with all others
    similarity_scores = list(enumerate(similarity_matrix[movie_idx]))

    # Sort the movies based on similarity scores
    similarity_scores = sorted(similarity_scores, key=lambda x: x[1], reverse=True)

    # Get the indices of the top N most similar movies
    top_movie_indices = [i[0] for i in similarity_scores[1:top_n+1]]  # Exclude the movie itself

    # Return the titles and genres of the top N most similar movies
    return netflix_data.iloc[top_movie_indices][['title', 'genre', 'imdb_score']]
```

```
# Example usage: Recommend similar movies to a given movie title
recommended_movies = recommend_content_based(movie_title='The Lovebirds', top_n=5)
print(recommended_movies)

# Visualize the top recommended movies
def visualize_recommendations(recommendations):
    plt.figure(figsize=(10, 6))
    sns.barplot(x='imdb_score', y='title', data=recommendations, palette='viridis')
    plt.title('Top Recommended Movies Based on Content Similarity')
    plt.xlabel('IMDb Score')
    plt.ylabel('Movie Title')
    plt.grid(True)
    plt.show()


# Visualize the recommended movies
visualize_recommendations(recommended_movies)


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Load the Netflix dataset
netflix_data = pd.read_csv('/content/netflix.csv')

# Data preprocessing
# Convert Premiere date to datetime and extract year
netflix_data['premiere'] = pd.to_datetime(netflix_data['premiere'], errors='coerce')
netflix_data.dropna(subset=['premiere'], inplace=True)
netflix_data['Premiere_year'] = netflix_data['premiere'].dt.year

# One-hot encode categorical features (genre, language)
encoder = OneHotEncoder(sparse=False)
encoded_features = encoder.fit_transform(netflix_data[['genre', 'language']])

# Scale numerical features (runtime, Premiere_year)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(netflix_data[['runtime', 'Premiere_year']])

# Combine one-hot encoded and scaled features into a single matrix
import numpy as np
```

```
features = np.hstack((encoded_features, scaled_features))

# Define target variable (IMDb score)
target = netflix_data['imdb_score']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Apply Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Predict the IMDb score for the test set
y_pred = ridge_model.predict(X_test)

# Evaluate the model: Calculate mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Visualization: True vs Predicted IMDb Scores

def visualize_true_vs_predicted(y_test, y_pred):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=y_test, y=y_pred, alpha=0.6, color='blue')
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')  # Diagonal
line
    plt.title('True vs Predicted IMDb Scores (Ridge Regression)')
    plt.xlabel('True IMDb Scores')
    plt.ylabel('Predicted IMDb Scores')
    plt.grid(True)
    plt.show()


# Call the function to visualize True vs Predicted IMDb Scores
visualize_true_vs_predicted(y_test, y_pred)

# Visualization: Residual Plot (Errors between true and predicted IMDb scores)
def visualize_residuals(y_test, y_pred):
    residuals = y_test - y_pred
    plt.figure(figsize=(10, 6))
    sns.histplot(residuals, kde=True, color='purple')
    plt.title('Residuals (True - Predicted IMDb Scores)')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    plt.grid(True)
```

```
    plt.show()


# Call the function to visualize Residuals
visualize_residuals(y_test, y_pred)


#Model Training and Testing

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Split the dataset into training and testing sets
X = df[['runtime', 'premiere_month', 'title_length']]
y = df['imdb_score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Cross-validation to evaluate the model
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
print("Cross-Validation MSE: ", -cv_scores.mean())

# Predict on the test set
y_pred = rf_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Test MSE: ", mse)


 # Cross-Validation and Hyperparameter Tuning

 from sklearn.model_selection import GridSearchCV

 # Define the parameter grid
 param_grid = {
     'n_estimators': [100, 200, 300],
     'max_depth': [None, 10, 20, 30],
     'min_samples_split': [2, 5, 10]

 }


 # Perform Grid Search
 grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='neg_mean_squared_error')
```

```
 grid_search.fit(X_train, y_train)

 # Best parameters and score
 print("Best Parameters: ", grid_search.best_params_)
 print("Best Cross-Validation MSE: ", -grid_search.best_score_)



# Define the function to create the model

def create_model(input_shape):
   model = Sequential([
      Input(shape=(input_shape,)),  # Correctly define input shape using Input layer
      Dense(128, activation='relu'),
      Dropout(0.2),
      Dense(64, activation='relu'),
      Dropout(0.2),
      Dense(32, activation='relu'),
      Dense(1)
   ])

   model.compile(optimizer='adam', loss='mse', metrics=['mae'])
   return model


# Create the model with the correct input shape
model = create_model(X_train_transformed.shape[1])


# Fit the model and save the training history

history = model.fit(X_train_transformed, y_train, epochs=100, batch_size=32, verbose=1,
validation_data=(X_test_transformed, y_test))


# Evaluate the model on the test data

loss, mae = model.evaluate(X_test_transformed, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Mean Absolute Error: {mae:.4f}')


# Visualize Training History (Optional)
```

```
# Plot training & validation loss values

plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
plt.savefig('Model Loss.png', dpi=300)
plt.show()


//MultiFusion Model

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout, Input, concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
import matplotlib.pyplot as plt


# Load the dataset

file_path = '/content/netflix.csv'  # Update the path as needed
df = pd.read_csv(file_path)


# Convert 'premiere' to datetime and extract month and year
df['premiere'] = pd.to_datetime(df['premiere'])
df['premiere_month'] = df['premiere'].dt.month
df['premiere_year'] = df['premiere'].dt.year


# Handling missing values
df['imdb_score'].fillna(df['imdb_score'].median(), inplace=True)
df['runtime'].fillna(df['runtime'].mean(), inplace=True)
df['genre'].fillna('Unknown', inplace=True)
df['language'].fillna('Unknown', inplace=True)
```

```
# Define features and target
numerical_features = ['runtime', 'premiere_month', 'premiere_year']
categorical_features = ['genre', 'language']
target = 'imdb_score'


# Preprocess numerical and categorical features

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)


# Apply the preprocessing pipeline to the dataset
X_processed = preprocessor.fit_transform(df[numerical_features + categorical_features])

# Define the target variable
y = df[target].values


# Split the processed data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)


# Define the input shape based on the processed data
input_shape = X_train.shape[1]

# Define the input layer
inputs = Input(shape=(input_shape,))

# Define Dense layers for the fusion model
x = Dense(128, activation='relu')(inputs)
x = Dropout(0.2)(x)
x = Dense(64, activation='relu')(x)|
x = Dropout(0.2)(x)
x = Dense(32, activation='relu')(x)
output = Dense(1)(x)  # Single output for regression
```

```
# Create the model
model = Model(inputs=inputs, outputs=output)

# Compile the model
# model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

# Recompile the model using the full metric name
# model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mean_squared_error'])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error',
metrics=['mean_absolute_error'])

# Display the model summary
model.summary()


# Train the model with early stopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(X_train, y_train,
            epochs=50,
            batch_size=32,
            validation_data=(X_test, y_test),
            callbacks=[early_stopping])

# Evaluate the model on the test data
loss, mae = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss:.4f}')
print(f'Test Mean Absolute Error: {mae:.4f}')


# Plot training & validation loss values

plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
plt.savefig('Model Loss V2.png', dpi=300)
plt.show()
```