Development part

To predict IMDb scores using data from a CSV file, you'll need to use Python and some libraries for data manipulation, analysis, and machine learning.

Here's a step-by-step guide on how to import the required libraries and work with a CSV file to predict IMDb scores:

pip install pandas scikit-learn

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from sklearn.metrics import r2_score.

Pandas:

This library is crucial for data manipulation and preprocessing.

import pandas as pd

Scikit-Learn (sklearn):

Scikit-Learn provides various machine learning algorithms and tools for model building and evaluation.

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

Matplotlib/Seaborn: These libraries are helpful for data visualization.

import matplotlib.pyplot as plt

import seaborn as sns

NumPy:

NumPy is used for numerical computations and array manipulations, often required for machine learning.

import numpy as np

Scipy:

If you need advanced statistical functions, you might use SciPy.

from scipy import stats

Statsmodels:

Useful for more detailed statistical analysis.

import statsmodels.api as sm

Feature Engineering and Selection Libraries:

Depending on the complexity of your project, you might need other libraries for feature engineering and selection, such as feature-engine or scikit-learn's feature selection tools.

You should also load your data using Pandas. Here's an example of how you might load a CSV file named data.csv:

data = pd.read_csv("data.csv")

Importing the data set:

You can perform one-hot encoding using various programming libraries such as Python's pandas or scikit-learn. Here's how you can do it using pandas:

import pandas as pd

data={'Genre':['comedy','Heist film/Thriller','Musical','comedy','documentary','drama',''romantic comedy','action comedy','thriller']}

In your Python script or Jupyter Notebook, import the necessary libraries and load the dataset using the Kaggle API.

import kaggle

import pandas as pd

Set your Kaggle API credentials

kaggle.api.authenticate(api_key="your_kaggle_api_key")

 # Download the dataset from Kaggle

dataset_name = "luiscorter/netflix-original-films-imdb-scores"

kaggle.api.dataset_download_files(dataset_name, path=".", unzip=True)

 # Load the dataset into a Pandas DataFrame

df = pd.read_csv("NetflixOriginals.csv") # Modify the filename if it's different

Once you have loaded the dataset into a Pandas DataFrame (in this example, I assumed the dataset file is named "NetflixOriginals.csv"), you can manipulate and create matrices as needed.

For example, if you want to create a matrix from specific columns, you can do something like

Extract relevant columns for your matrix

selected_columns = ["column1", "column2", "column3"] # Replace with the actual column names

 # Create a matrix from the selected columns

data_matrix = df[selected_columns].values

Replace "column1," "column2," and "column3" with the actual column names you want in your matrix.

Now you have imported the dataset and created a matrix from it in Python. Make sure to adapt the column names and data processing steps according to your specific requirements and analysis.

Import the necessary libraries:

import pandas as pd

from sklearn.impute import SimpleImputer

Load the dataset into a Pandas DataFrame:

df = pd.read_csv("NetflixOriginals.csv")

Identify the columns with missing data that you want to impute. For this example, let's assume

you want to handle missing values in the "column1" and "column2" of your dataset.

Initialize the SimpleImputer and specify the strategy for imputation.

The most common strategies are "mean," "median," "most_frequent," or you can even use a constant value for imputation.

imputer = SimpleImputer(strategy="mean")

Fit the imputer on your dataset, focusing on the columns with missing data:

columns_to_impute = ["column1", "column2"] # Replace with the actual column names

imputer.fit(df[columns_to_impute])

Transform the DataFrame to replace missing values with the imputed values:

df[columns_to_impute] = imputer.transform(df[columns_to_impute])

Encoding categorical data:

To encode categorical data using one-hot encoding in Python, you can use libraries like pandas and scikit-learn. Here's how you can perform one-hot on categorical columns in your dataset:

First, make sure you have already imported your dataset into a Pandas DataFrame, as mentioned in the previous responses.

Identify the categorical columns that you want to one-hot encode. For this example, let's assume you have a column named "Genre" that contains categorical data.

Use the pandas.get_dummies function to perform one-hot encoding on the categorical column:

Identify the categorical column to be one-hot encoded

categorical_column = "Genre

Perform one-hot encoding

df_encoded = pd.get_dummies(df, columns=[categorical_column])

The pd.get_dummies function will create new binary (0 or 1) columns for each unique category in the specified column, effectively one-hot encoding the data.

You can also specify other parameters in pd.get_dummies,

such as prefixing the new columns, handling missing values, and dropping one of the categories to avoid multicollinearity. Here's an example with some additional options:

Perform one-hot encoding with additional options

df_encoded = pd.get_dummies(df, columns=[categorical_column], prefix=categorical_column, dummy_na=True, drop_first=True)

prefix=categorical_column will add a prefix to the new columns for clarity.

dummy_na=True will create a separate column for missing values.

drop_first=True will drop the first category to avoid multicollinearity.

IN genre:

Comedy

Heist film/Thriller

Musical/Western/Fantasy

Comedy

Documentary

Drama

Romantic comedy

Action comedy

Thriller

In this it contain 9 columns

in comedy

1

0

0

1

0

0

1

1

0

In thriller:

0

1

0

0

0

0

0

0

1

Splitting the data set :

To split your dataset into a training set and a test set using the train_test_split function from scikit-learn, you can follow these steps:

Make sure you have imported your dataset and encoded it as needed (e.g., handling missing values and one-hot encoding).

Import the train_test_split function:

from sklearn.model_selection import train_test_split

Split your dataset into features (X) and the target variable (Y).

Replace "Target_Column_Name" with the actual name of your target variable, and remove it from the features (X):

X = df.drop("Target_Column_Name", axis=1) # X contains all columns except the target

Y = df["Target_Column_Name"] # Y is the target variable

Use train_test_split to split your data into a training set and a test set. Specify the test size and random seed as needed:

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

X_train: The features for the training set.

X_test: The features for the test set.

Y_train: The target variable for the training set.

Y_test: The target variable for the test set.

In this example, test_size is set to 0.2, which means 20% of the data will be used for testing, and the remaining 80% will be used for training.

You can adjust test_size to your preference.

The random_state parameter is used to ensure reproducibility.

If you want the split to be the same every time you run the code, use the same random_state value.

If you don't need that, you can omit the random_state parameter.

Feature scaling:

To perform feature scaling using StandardScaler from scikit-learn, you can follow these steps after splitting your dataset into training and test sets.

Feature scaling is important for many machine learning algorithms to ensure that features are on similar scales. Here's how to do it:

Import the necessary library and create an instance of StandardScaler:

```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
```

Fit and transform the scaler on the training set's feature data. This step computes the mean and standard deviation necessary for scaling:

```
X_train_scaled = scaler.fit_transform(X_train)
```

Transform the test set's feature data using the same scaler.

It's important to use the same scaler that you fitted on the training data to ensure consistency:

```
X_test_scaled = scaler.transform(X_test)
```

Now, you have standardized your features in both the training set (X_train_scaled) and the test set (X_test_scaled).

This scaling process will make sure that your features have a mean of 0 and a standard deviation of 1, which can be beneficial for many machine learning algorithms.

Remember that it's essential to perform feature scaling after splitting the data into training and test sets.

This ensures that information from the test set doesn't leak into the training set during scaling, which could lead to data leakage and inaccurate evaluation of your machine learning models.