MovieLens 大作业

——基于观影数据集的数据分析与挖掘

2xxxxxxxxx

1. 数据集展示及环境配置

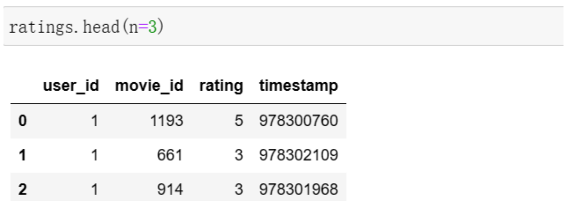
1.1 数据集展示

本次任务一共有 4 个数据集，分别为 ratings, users, movies, movies_info

表名	内容
ratings2.csv	user_id, movie_id, rating, timestamp
users.csv	user_id, gender, age_desc, occ_desc
movies.csv	movie_id, title, genres
info.csv	movie_id, name, genre, release_time, intro, directors, stars

每个数据集的 head(n=3)部分如下：

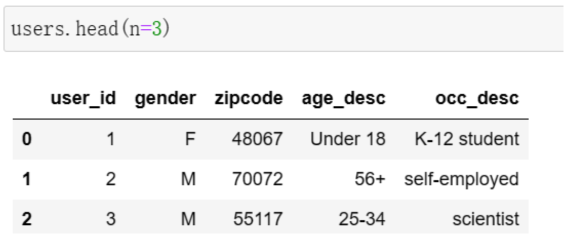
Ratings:



```
ratings.head(n=3)
```

	user_id	movie_id	rating	timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968

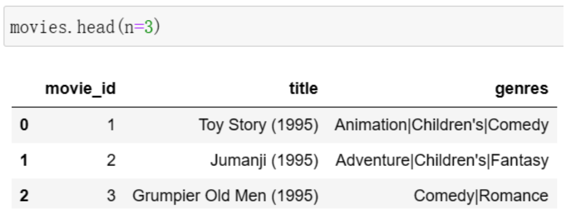
Users:



```
users.head(n=3)
```

	user_id	gender	zipcode	age_desc	occ_desc
0	1	F	48067	Under 18	K-12 student
1	2	M	70072	56+	self-employed
2	3	M	55117	25-34	scientist

Movies:



```
movies.head(n=3)
```

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance

Movies_info:

```
movies_info.head(n=3)
```

	movie_id	name	genres	release_time	intro	directors	stars
0	1	Toy Story (1995)	Animation|Adventure|Comedy	22 November 1995 (USA)	A cowboy doll is profoundly threatened and jea...	John Lasseter	Tom Hanks|Tim Allen|Don Rickles
1	2	Jumanji (1995)	Adventure|Comedy|Family	15 December 1995 (USA)	When two kids find and play a magical board ga...	Joe Johnston	Robin Williams|Kirsten Dunst|Bonnie Hunt
2	3	Grumpier Old Men (1995)	Comedy|Romance	22 December 1995 (USA)	John and Max resolve to save their beloved bai...	Howard Deutch	Walter Matthau|Jack Lemmon|Ann-Margret

1.2 环境配置

除了题目要求环境，还额外使用了 tqdm, lightgbm, xgboost, skmultilearn 的第三方库。

（1） 第一部分涉及第三方库：

```
import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
```

（2） 第二部分涉及第三方库：

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import numpy as np
import datetime
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
import lightgbm as lgb
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

（3） 第三部分涉及第三方库：

```
import os
import cv2
import numpy as np
import pandas as pd
from PIL import Image
import torch
from img2vec_pytorch import Img2Vec
from tqdm import tqdm
from sklearn.cluster import KMeans
```

```
import torch
import torchvision
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from skmultilearn.problem_transform import LabelPowerset
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from xgboost import XGBClassifier
```

2. 传统偏好发现（30 分）

本次从职业、年龄中选择年龄，分析用户偏好。

2.1 设计指标并筛选偏好前十的电影

（1）指标设计（10 分）

已知年龄的类别为 Under 18, 18-24, 25-34, 35-44, 45-49, 50-55, 56+7 种类别，由于我们不能先入为主地认为年龄大小与偏好存在线性关系，比如我们不能通过 Under18 的偏好>45-49 的偏好从而推断出 45-49 的偏好>56+的偏好，所以我们认为在设计指标的时候需要针对每一个类别计算（i 类型的平均评分-除去 i 类型之外的平均评分）得到 i 类型的偏好。

但是由于 7 个类别设置 7 个指标过于繁杂没有统一性，我们将年龄再次分组为 0-24(youth), 25-44(adult), 45+(old)三个类别，设置三个指标：少年组比其他组别更偏好程度、青年组比其他组别更偏好程度、中老年组比其他组别更偏好程度。

又由于观影人数多能增加偏好程度的可信度，所以在差值上乘以 Ln(观影人数)作为权重得到最终的偏好分值。涉及的公式如下：

youth_score=(youth_avr_rate-other_avr_rate)*ln(观影人数)

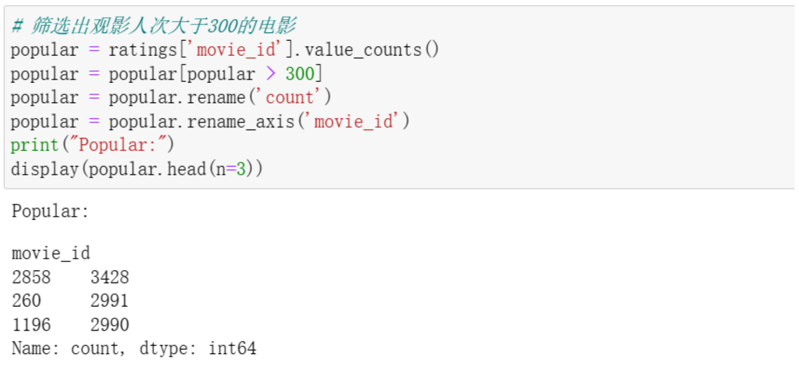
adult_score =(adult_avr_rate-other_avr_rate)*ln(观影人数)

old_score =(old_avr_rate-other_avr_rate)*ln(观影人数)

（2）展示不同类别偏好前十的电影（10 分）

以下为程序过程：

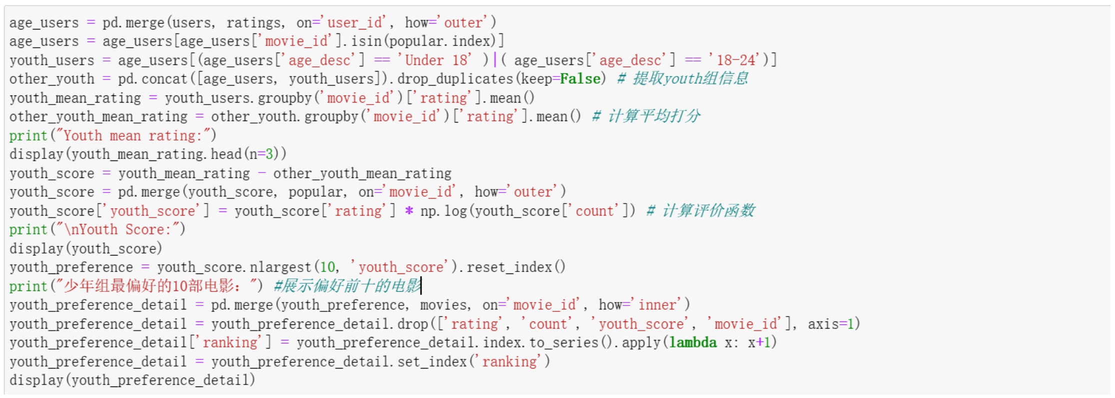
首先筛选出观影人次大于 300 的电影：



```
# 筛选出观影人次大于300的电影
popular = ratings['movie_id'].value_counts()
popular = popular[popular > 300]
popular = popular.rename('count')
popular = popular.rename_axis('movie_id')
print("Popular:")
display(popular.head(n=3))
```

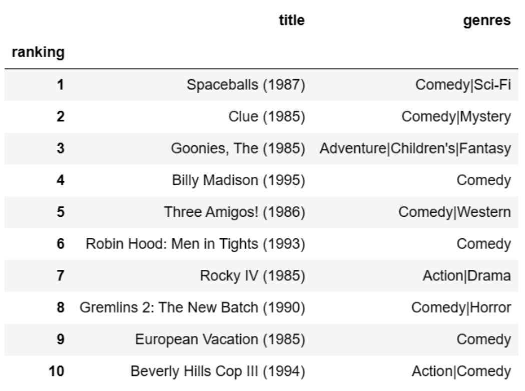
```
Popular:

movie_id
2858    3428
260     2991
1196    2990
Name: count, dtype: int64
```

其次提取各个年龄段的数据，再计算平均打分和评价函数，并展示出前 10 个不同类型用户喜欢的电影（以下只给出 youth 组的代码，并附上三组的结果）

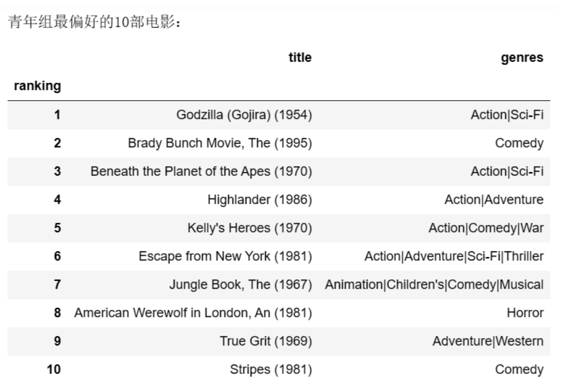


```
age_users = pd.merge(users, ratings, on='user_id', how='outer')
age_users = age_users[age_users['movie_id'].isin(popular.index)]
youth_users = age_users[(age_users['age_desc'] == 'Under 18' )|( age_users['age_desc'] == '18-24')]
other_youth = pd.concat([age_users, youth_users]).drop_duplicates(keep=False) # 提取youth组信息
youth_mean_rating = youth_users.groupby('movie_id')['rating'].mean()
other_youth_mean_rating = other_youth.groupby('movie_id')['rating'].mean() # 计算平均打分
print("Youth mean rating:")
display(youth_mean_rating.head(n=3))
youth_score = youth_mean_rating - other_youth_mean_rating
youth_score = pd.merge(youth_score, popular, on='movie_id', how='outer')
youth_score['youth_score'] = youth_score['rating'] * np.log(youth_score['count']) # 计算评价函数
print("\nYouth Score:")
display(youth_score)
youth_preference = youth_score.nlargest(10, 'youth_score').reset_index()
print("少年组最偏好的10部电影：") #展示偏好前十的电影
youth_preference_detail = pd.merge(youth_preference, movies, on='movie_id', how='inner')
youth_preference_detail = youth_preference_detail.drop(['rating', 'count', 'youth_score', 'movie_id'], axis=1)
youth_preference_detail['ranking'] = youth_preference_detail.index.to_series().apply(lambda x: x+1)
youth_preference_detail = youth_preference_detail.set_index('ranking')
display(youth_preference_detail)
```

少年组最偏好的10部电影：

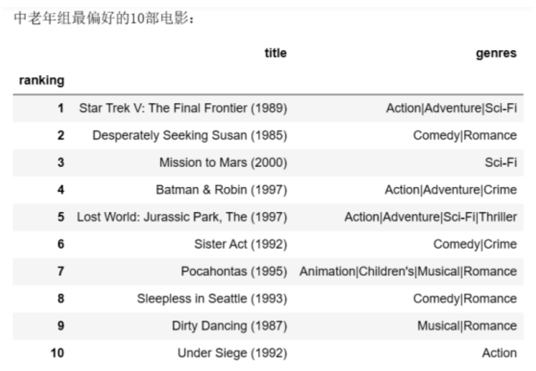


ranking	title	genres
1	Spaceballs (1987)	Comedy|Sci-Fi
2	Clue (1985)	Comedy|Mystery
3	Goonies, The (1985)	Adventure|Children's|Fantasy
4	Billy Madison (1995)	Comedy
5	Three Amigos! (1986)	Comedy|Western
6	Robin Hood: Men in Tights (1993)	Comedy
7	Rocky IV (1985)	Action|Drama
8	Gremlins 2: The New Batch (1990)	Comedy|Horror
9	European Vacation (1985)	Comedy
10	Beverly Hills Cop III (1994)	Action|Comedy



青年组最偏好的10部电影：

ranking	title	genres
1	Godzilla (Gojira) (1954)	Action|Sci-Fi
2	Brady Bunch Movie, The (1995)	Comedy
3	Beneath the Planet of the Apes (1970)	Action|Sci-Fi
4	Highlander (1986)	Action|Adventure
5	Kelly's Heroes (1970)	Action|Comedy|War
6	Escape from New York (1981)	Action|Adventure|Sci-Fi|Thriller
7	Jungle Book, The (1967)	Animation|Children's|Comedy|Musical
8	American Werewolf in London, An (1981)	Horror
9	True Grit (1969)	Adventure|Western
10	Stripes (1981)	Comedy



中老年组最偏好的10部电影：

ranking	title	genres
1	Star Trek V: The Final Frontier (1989)	Action|Adventure|Sci-Fi
2	Desperately Seeking Susan (1985)	Comedy|Romance
3	Mission to Mars (2000)	Sci-Fi
4	Batman & Robin (1997)	Action|Adventure|Crime
5	Lost World: Jurassic Park, The (1997)	Action|Adventure|Sci-Fi|Thriller
6	Sister Act (1992)	Comedy|Crime
7	Pocahontas (1995)	Animation|Children's|Musical|Romance
8	Sleepless in Seattle (1993)	Comedy|Romance
9	Dirty Dancing (1987)	Musical|Romance
10	Under Siege (1992)	Action

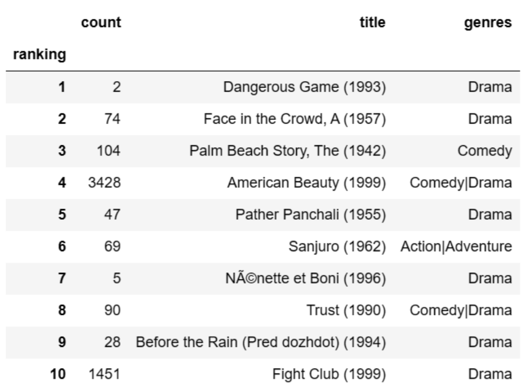
（3）基于其他流行统计量$Ra=WR+(1-W)R0$ 得到的结果

基于新的流行统计量得到新的 popular：

```
R=ratings.groupby('movie_id')['rating'].mean()
N=ratings.groupby('movie_id')['rating'].count()
popular=pd.DataFrame({'rating':R,'count':N})
W=popular['count'].apply(lambda x: max(0.5*x/popular['count'].mean(),1))
popular['rating']=W*popular['rating']+(1-W)*popular['rating'].mean()
popular=popular.sort_values(by='rating',ascending=False)
popular_percentile=popular['rating'].quantile(0.9)
popular=popular[popular['rating']>=popular_percentile]
print(popular)
```

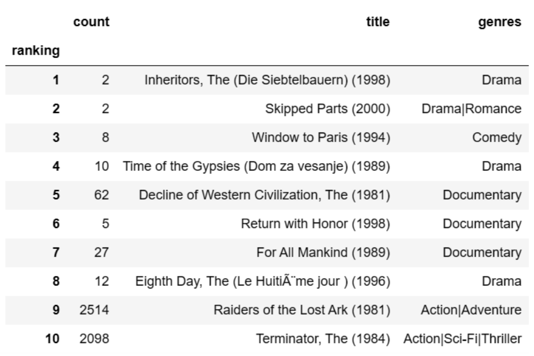
其余方法和（1）相同，得到：

少年组：



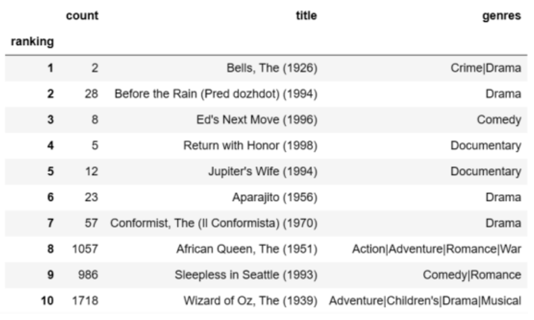
ranking	count	title	genres
1	2	Dangerous Game (1993)	Drama
2	74	Face in the Crowd, A (1957)	Drama
3	104	Palm Beach Story, The (1942)	Comedy
4	3428	American Beauty (1999)	Comedy|Drama
5	47	Pather Panchali (1955)	Drama
6	69	Sanjuro (1962)	Action|Adventure
7	5	NÃ©nette et Boni (1996)	Drama
8	90	Trust (1990)	Comedy|Drama
9	28	Before the Rain (Pred dozhdot) (1994)	Drama
10	1451	Fight Club (1999)	Drama

青年组：



ranking	count	title	genres
1	2	Inheritors, The (Die Siebtelbauern) (1998)	Drama
2	2	Skipped Parts (2000)	Drama|Romance
3	8	Window to Paris (1994)	Comedy
4	10	Time of the Gypsies (Dom za vesanje) (1989)	Drama
5	62	Decline of Western Civilization, The (1981)	Documentary
6	5	Return with Honor (1998)	Documentary
7	27	For All Mankind (1989)	Documentary
8	12	Eighth Day, The (Le HuitiÃ¨me jour) (1996)	Drama
9	2514	Raiders of the Lost Ark (1981)	Action|Adventure
10	2098	Terminator, The (1984)	Action|Sci-Fi|Thriller

中老年组：



ranking	count	title	genres
1	2	Bells, The (1926)	Crime|Drama
2	28	Before the Rain (Pred dozhdot) (1994)	Drama
3	8	Ed's Next Move (1996)	Comedy
4	5	Return with Honor (1998)	Documentary
5	12	Jupiter's Wife (1994)	Documentary
6	23	Aparajito (1956)	Drama
7	57	Conformist, The (Il Conformista) (1970)	Drama
8	1057	African Queen, The (1951)	Action|Adventure|Romance|War
9	986	Sleepless in Seattle (1993)	Comedy|Romance
10	1718	Wizard of Oz, The (1939)	Adventure|Children's|Drama|Musical

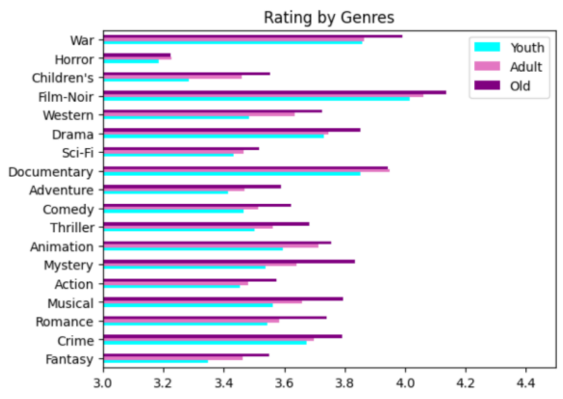
2.2 基于电影风格的可视化（10 分）

在可视化之前的准备工作有：

根据 genres 字段进行分割，对 movies 数据集进行扩展，填充每部电影的风格标记（0-1 矩阵），将 users, ratings 和扩展后的 movies 合并，遍历每一种风格，对不同年龄组的评分数据进行收集，计算计算均值、标准差，同时将评分数据归一化。

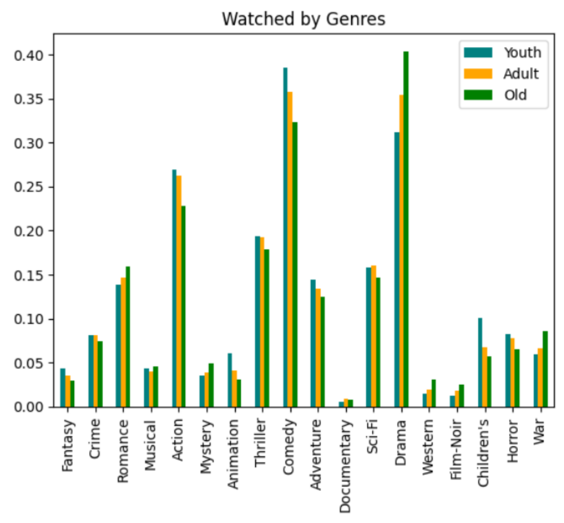
（1） 不同年龄组在不同风格电影中平均均值对比

计算在某一风格的电影下，不同年龄组对于电影评分的均值



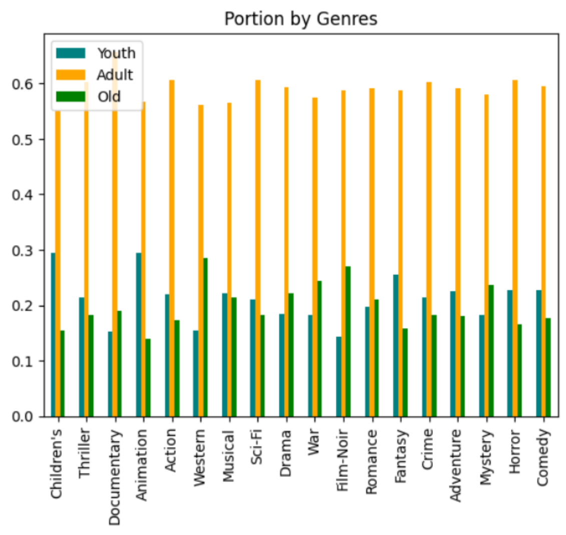
（2） 不同年龄组观看不同电影风格的数量对比

计算在某一电影风格下，不同年龄组观看该风格电影占本组总观影人次的比例



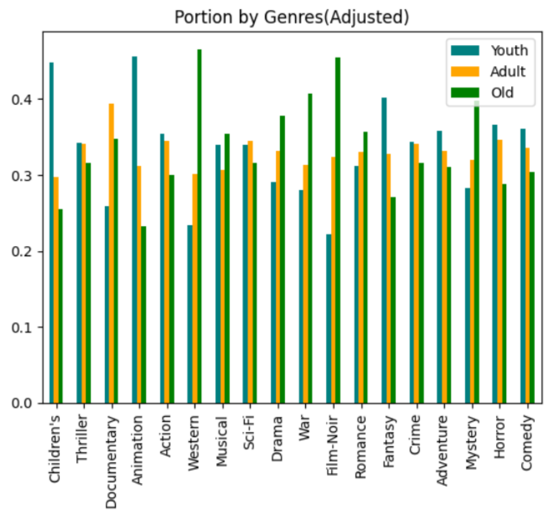
（3） 不同年龄组在不同风格电影中的比例对比

计算在某一电影风格下，不同年龄组观看该风格电影占观看该风格电影总观影人次的比例



（4） 不同年龄组在不同风格电影中的比例对比（消除人数基数影响）

将所有年龄组的人数进行归一化，假设所有年龄组的人数基数相同的情况下，计算在某一电影风格下，不同年龄组观看该风格电影占观看该风格电影总观影人次的比例



3.　用户评分预测（40 分+5）

3.1 基础预测（仅基于性别、年龄、职业、电影类型的特征）（30 分）

（1）特征工程：对性别、年龄、职业、电影类型特征进行 One-Hot 编码（8 分），并降维（2 分）

首先将 ratings, users, movies, movies_info 四张表合起来，从中选取 gender, age_desc, occ_desc, genres 作为特征，并进行编码。由于 genres 特征中一个电影可能具有多种 genre，不适合使用 one-hot 编码，所以我们对 genres 延用上一题中在 movies dataframe 中拆出的每个电影针对 18 种电影风格的 0-1 矩阵，每个电影可能对应多个 1。然而，在本次数据集中，gender, age_desc, occ_desc 不存在此类问题，每个电影针对每种字段仅有一个类型，可以使用 one-hot 编码，得到降维前需要的特征。此后使用 PCA 对提取出来的特征进行降维。

```
merged_data = pd.merge(ratings, users, on='user_id', how='left')
merged_data = pd.merge(merged_data, movies, on='movie_id', how='left')
merged_data = pd.merge(merged_data, movies_info, on='movie_id', how='left')
merged_data['year']=merged_data['release_time']
```

```
merged_data.columns
```

```
Index(['user_id', 'movie_id', 'rating', 'timestamp', 'gender', 'zipcode',
       'age_desc', 'occ_desc', 'title', 'genres_x', 'Drama', 'Horror',
       'Documentary', 'Crime', 'Fantasy', 'Musical', 'Thriller', 'War',
       'Action', 'Adventure', 'Comedy', 'Western', 'Children's', 'Animation',
       'Romance', 'Film-Noir', 'Mystery', 'Sci-Fi', 'name', 'genres_y',
       'release_time', 'intro', 'directors', 'stars', 'year'],
      dtype='object')
```

```
features = merged_data[['gender', 'age_desc', 'occ_desc', 'Fantasy', 'Crime',
       'Romance', 'Musical', 'Action', 'Mystery', 'Animation', 'Thriller',
       'Comedy', 'Adventure', 'Documentary', 'Sci-Fi', 'Drama', 'Western',
       'Film-Noir', "Children's", 'Horror', 'War']]
column_transformer = ColumnTransformer(
    [('one_hot_encoder', OneHotEncoder(), ['gender', 'age_desc', 'occ_desc'])],
    remainder='passthrough')
one_hot_features = column_transformer.fit_transform(features)
print(one_hot_features.shape)
pca=PCA(n_components=0.95)
reduced_features=pca.fit_transform(one_hot_features)
```

```
(1000209, 48)
```

（2）模型训练：划分训练集与测试集（2 分），使用机器学习模型进行预测（8 分）

从数据集中划取 20%作为测试集，基于梯度提升使用 LightGBM 模型对数据进行训练和预测。

```
import lightgbm as lgb
target=merged_data['rating']
X_train, X_test, y_train, y_test = train_test_split(reduced_features, target, test_size=0.2, random_state=42)
model = lgb.LGBMClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.068349 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 8415
[LightGBM] [Info] Number of data points in the train set: 800167, number of used features: 33
[LightGBM] [Info] Start training from score -2.883215
[LightGBM] [Info] Start training from score -2.228220
[LightGBM] [Info] Start training from score -1.342099
[LightGBM] [Info] Start training from score -1.052452
[LightGBM] [Info] Start training from score -1.487451
```

（3）预测与评估：使用 numpy 写一个 MSE 函数（2 分），在上述训练中实现 MSE<2 的要求（8 分）

```
def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

mse = mean_squared_error(y_test, y_pred)
print("MSE:", mse)
```

```
MSE: 1.3952769918317154
```

计算得到上述过程的 MSE 为 1.39<2，满足要求。

3.2 高级预测（10 分）

经过以下特征调整和模型训练，得到最小 MSE 为 1.19，不能满足 MSE<1 的要求，但做了以下尝试：

（1） 将上述基础预测步骤中的 PCA 降维更换为截断 SVD 降维，没能有效减少 MSE

```
mse_svd = mean_squared_error(y_test_svd, y_pred_svd)
print("MSE_SVD:", mse_svd)
```

```
MSE_SVD: 1.403350296437748
```

（2） 添加 movie_id, year, country, movie_intro 作为特征加入特征工程，并结合截断 SVD 做降维：

由于只尝试其他的降维方式没有作用，我们希望添加新的特征和选用新的模型。

首先尝试加入 movie_id 和 user_id，企图通过电影本身的质量和用户的打分偏好来增加预测的准确度，但经过后期尝试发现其 MSE 大于只添加 movie_id 的 MSE，所以在提取 features 的步骤我们只添加 movie_id。

其次我们针对 year 这一列使用正则表达式提取出每个电影的年份和国家，并对国家进行 one-hot 编码，也加入特征工程当中。

```
import re
def extract_info(date_str):
    year = re.findall(r'\b\d{4}\b', date_str)[0]  # 提取四位数字作为年份
    country_match = re.search(r'\((.*?)\)', date_str)  # 在括号中提取国家信息
    country = country_match.group(1) if country_match else None
    return year, country

# 应用提取函数到datastr列
merged_data[['Year', 'Country']] = merged_data['year'].apply(lambda x: pd.Series(extract_info(x)))
```

...

```
merged_data['Year'] = merged_data['Year'].astype(int)
```

```
features = merged_data[['gender', 'Year','Country','age_desc', 'occ_desc', 'movie_id','Fantasy', 'Crime',
       'Romance', 'Musical', 'Action', 'Mystery', 'Animation', 'Thriller',
       'Comedy', 'Adventure', 'Documentary', 'Sci-Fi', 'Drama', 'Western',
       'Film-Noir', "Children's", 'Horror', 'War']]
column_transformer = ColumnTransformer(
    [('one_hot_encoder', OneHotEncoder(), ['gender', 'age_desc', 'occ_desc','Country'])],
    remainder='passthrough')
one_hot_features = column_transformer.fit_transform(features)
one_hot_features.shape
```

```
(981855, 94)
```

最后，我们对 movie_intro 的文本计算 TF-IDF 矩阵，使用截断 SVD 对 TF-IDF 做降维（类似 LSA 模型），将降维后的文本特征矩阵和上述 feature 矩阵结合，再做一次截断 SVD 降维得到新的特征矩阵。

```
movie_intro=merged_data['intro'].tolist()
```

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(movie_intro)
tfidf_matrix.shape
```

```
(981855, 10328)
```

```
lsa_model = TruncatedSVD(n_components=200)
lsa_matrix = lsa_model.fit_transform(tfidf_matrix)
lsa_matrix.shape
```

```
(981855, 200)
```

```
lsa_df=pd.DataFrame(lsa_matrix)
feature_df=pd.DataFrame(one_hot_features.toarray())
lsa_df.shape,feature_df.shape
```

```
((981855, 200), (981855, 94))
```

```
combined_features = pd.concat([feature_df, lsa_df], axis=1)
combined_features.shape
```

```
(981855, 294)
```

```
svd = TruncatedSVD(n_components=100)
reduced_features_svd = svd.fit_transform(combined_features)
```

（3） 尝试除了 LightGBM 之外的模型，使用 LogisticRegression, Linear Discriminant Analysis, XGBoost 模型来预测

原本我希望尝试 SVM 和 Random Forest，但特征矩阵过于庞大，这两种模型运算时间过长，所以我们选择了 LightGBM, LogisticRegression, Linear Discriminant Analysis, XGBoost 模型针对上述新特征矩阵进行新的训练，得到 MSE 如下：

```
clfs={
    'Logistic Regression': LogisticRegression(),
    'LightGBM': lgb.LGBMClassifier(verbose=-1),
    'XGBoost': XGBClassifier(),
    #'SVM': SVC(),
    'Linear Discriminant Analysis': LinearDiscriminantAnalysis(),
    #'Random Forest': RandomForestClassifier(),
}
```

```
for clf_name,clf in clfs.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(clf_name," MSE:",mse)
```

```
Logistic Regression  MSE: 1.4231072816250872
LightGBM  MSE: 1.1987717127274393
XGBoost  MSE: 1.1902368475996965
Linear Discriminant Analysis  MSE: 1.420825885695953
```

得到最小 MSE 来自于 XGBoost 为 1.19。

4.　海报按内容聚类（30 分）

4.1　图像特征提取与降维：从海报中提取颜色直方图、灰度直方图等特征并拼接（5 分），使用 PCA 和其他方法降维（3+2 分）

首先使用 opencv 定义两个提取颜色直方图和灰度直方图的函数，如下：

```
def extract_color_histogram(image):
    hist = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist

def extract_grayscale_histogram(image):
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist
```

再将这两个函数得到的直方图拼接在一起，但是由后续有监督聚类的分析检验，这样的特征矩阵得到的准确度较低。所以本次我们选择使用颜色直方图、灰度直方图和 Img2Vec 提取的特征拼接起来得到我们降维前的特征。

```
folder_path = "./poster"
features = []
img2vec_model = Img2Vec()

for filename in tqdm(os.listdir(folder_path)):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        image_path = os.path.join(folder_path, filename)
        image = cv2.imread(image_path)
        color_hist = extract_color_histogram(image)
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray_hist = extract_grayscale_histogram(gray_image)
        image0 = Image.open(image_path)
        if image0.mode != 'RGB':
            image0 = image0.convert('RGB')
        vector = img2vec_model.get_vec(image0)
        feature_vector = np.concatenate((color_hist, gray_hist, vector))
        features.append([filename.split('.')[0], feature_vector])


columns = ['movie_id', 'features']
df = pd.DataFrame(features, columns=columns)

print("DataFrame with Movie ID and Features:")
print(df.head())
```

对该特征矩阵分别进行 PCA 和截断 SVD 降维：

```
from sklearn.decomposition import PCA

X = np.array(df['features'].tolist())
pca = PCA(n_components=0.9)
X_pca = pca.fit_transform(X)
pca_features_list = X_pca.tolist()
df_result = pd.DataFrame({'movie_id': df['movie_id'], 'pca_features': pca_features_list})

print("DataFrame with Movie ID and PCA Features:")
print(df_result.head())
```

```
DataFrame with Movie ID and PCA Features:
  movie_id                                       pca_features
0        1  [0.39958012104034424, 6.453545570373535, -1.67...
1       10  [-3.4519400596618652, -1.992896318435669, -3.3...
2      100  [-1.1995089054107666, -7.333141803741455, 4.98...
3     1003  [-1.0824620723724365, -7.392695903778076, -0.1...
4     1004  [-0.4197857081890106, -6.109785079956055, -4.9...
```

```
from sklearn.decomposition import TruncatedSVD

n_components = 50
svd = TruncatedSVD(n_components=n_components)
X_svd = svd.fit_transform(X)
svd_features_list = X_svd.tolist()
df_result_svd = pd.DataFrame({'movie_id': df['movie_id'], 'svd_features': svd_features_list})

print("DataFrame with Movie ID and Truncated SVD Features:")
print(df_result_svd.head())
```

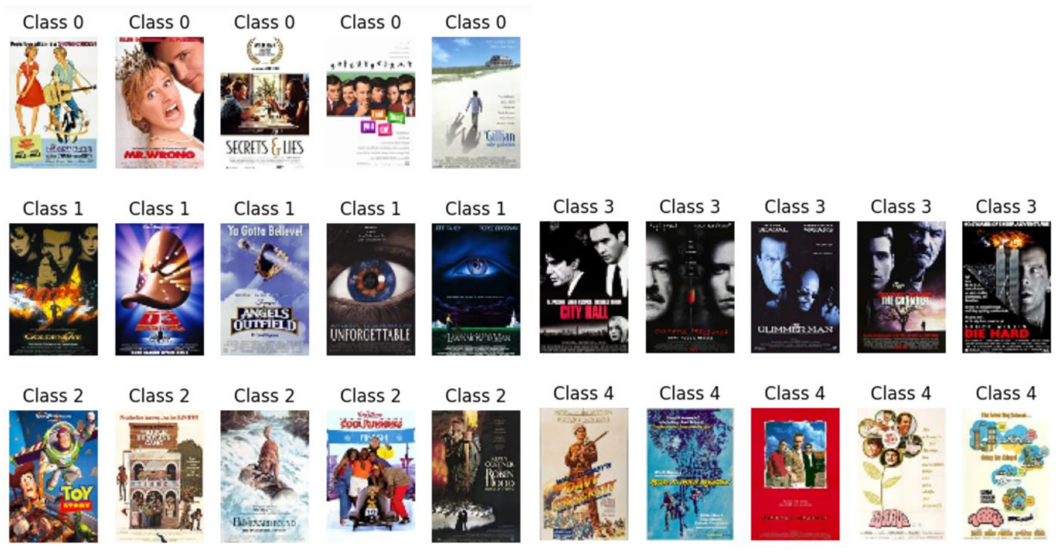
```
DataFrame with Movie ID and Truncated SVD Features:
  movie_id                                      svd_features
0        1  [24.60973358154297, 1.4586468935012817, 6.8198...
1       10  [23.475494384765625, -3.1120924949645996, -2.0...
2      100  [21.993040084838867, -0.9967103004455566, -7.3...
3     1003  [21.57632064819336, -1.0070329904556274, -7.46...
4     1004  [22.849201202392578, 0.02555108070373535, -6.0...
```

4.2 无监督聚类：K-means（5 分）

将电影海报聚成 5 类：

```
from sklearn.cluster import KMeans
n_clusters = 5
kmeans = KMeans(n_clusters=n_clusters)

kmeans.fit(X)
labels = kmeans.labels_
cluster_centers = kmeans.cluster_centers_
df['cluster_label'] = labels

for i in range(n_clusters):
    representative_images_indices = [df[df['cluster_label'] == i].index[j] for j in range(min(5, len(df[df['cluster_label'] == i])))]
    for j, index in enumerate(representative_images_indices):
        representative_image_path = os.path.join(folder_path, os.listdir(folder_path)[index])
        representative_image = cv2.imread(representative_image_path)
        representative_image = cv2.resize(representative_image, (60, 90))
        plt.subplot(1, 5, j+1)
        plt.imshow(cv2.cvtColor(representative_image, cv2.COLOR_BGR2RGB))
        plt.title("Class {}".format(i))
        plt.axis("off")
    plt.show()
```

得到结果：



4.3 有监督聚类：以电影风格作为 y，划分数据集（1 分），进行训练（2 分），准确度>0.2（2 分）

首先提取 genre_vector 得到有监督学习的 y：

```
movies_df = pd.read_csv("./movies.csv")
genres = movies_df['genres']
genre_dict = {}
for genre_string in genres:
    genre_list = genre_string.split("|")
    for genre in genre_list:
        if genre not in genre_dict:
            genre_dict[genre] = len(genre_dict)
print("Genre Dictionary:")
print(genre_dict)
genre_vectors = []
for genre_string in genres:
    genre_list = genre_string.split("|")
    genre_vector = [0] * len(genre_dict)
    for genre in genre_list:
        genre_vector[genre_dict[genre]] = 1
    genre_vectors.append(genre_vector)
genre_lists = [list(genre_vector) for genre_vector in genre_vectors]
movies_df['genre_vector'] = genre_lists
print("\nMovies DataFrame with Genre Vector:")
print(movies_df.head())
```

```
Movies DataFrame with Genre Vector:
   Unnamed: 0  movie_id                           title  \
0           0         1                Toy Story (1995)
1           1         2                  Jumanji (1995)
2           2         3         Grumpier Old Men (1995)
3           3         4        Waiting to Exhale (1995)
4           4         5  Father of the Bride Part II (1995)

                        genres  \
0   Animation|Children's|Comedy
1  Adventure|Children's|Fantasy
2               Comedy|Romance
3                 Comedy|Drama
4                       Comedy

                                        genre_vector
0  [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1  [0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2  [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
3  [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
4  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
```

使用上述 PCA 降维后的特征矩阵作为 X，genre_vector 作为 y 进行训练。由于本训练属于多标签训练，一般有基于问题转化的方法和基于算法适用的方法两种思路，为了简单起见，我们这里选择使用问题转化方法，分别使用 Binary Relevance，Classifier Chain，Label Powerset 来训练多标签问题，他们分别是思路是：将多标签独立为多个单标签来训练，使用链式结构来训练多个分类器，将多标签转化为多分类问题。

针对每一种问题转化方法，我们尝试使用的分类器有 Logistic Regression, Random Forest, SVM, MLP, KNN, Naive Bayes, Decision Tree, Linear Discriminant Analysis, XGBoost， 最后我们得到：

（1） LabelPowerset 方法下 Random Forest 和 SVM 的训练效果较好，准确度>0.2

使用 PCA：

```
Classifier: Random Forest
Accuracy: 0.21428571428571427

Classifier: SVM
Accuracy: 0.23469387755102042
```

使用截断 SVD：

```
Classifier: Random Forest
Accuracy: 0.22959183673469388

Classifier: SVM
Accuracy: 0.23299319727891157
```

（2） Classifier Chain 方法下，Logistic Regression 和 Linear Discriminant Analysis 效果较好，准确率>0.2

使用截断 SVD：

```
Classifier: Logistic Regression
Accuracy: 0.21428571428571427
```

```
Classifier: Linear Discriminant Analysis
Accuracy: 0.21768707482993196
```

综上，最高的准确度为 0.23

4.4 高级聚类分析（10 分）

经过多种特征工程调整的尝试，准确率仍然没能达到 0.3 以上，但我的尝试如下：

1） 将 directors, stars, posters, year, country, intro 全部加入特征工程，每个特征的处理方式为：directors 和 stars 处理方式和 genres 一致，生成一个 vector 后降维；year, country 的处理方式和第二部分相同（year 不处理，country 使用 one-hot 编码）；posters 是将直方图降维；intro 和第二部分相同，使用 TF-IDF 后降维；但最终预测准确率没有较高的提升

2） 只将 posters, intro, title 加入特征工程，intro 和 title 的处理方式均是 TF-IDF 后降维，将三个特征的矩阵拼接起来；最终预测准确率也没有较大提升。

这两种尝试下最高准确率仍然在 0.23-0.24 之间，没能满足题目要求。