案例1：银行客户流失预测

# 数据来源：匿名处理后的国外银行真实数据，共有14个特征，保存在Churn-Modelling-new.csv中

样本特征说明

RowNumber：行号 ×

CustomerID：用户编号 ×

Surname：用户姓名 ×

CreditScore：信用分数

Geography：用户所在国家/地区 ×

Gender：用户性别

Age：年龄

Tenure：当了本银行多少年用户

Balance：存贷款情况

NumOfProducts：使用产品数量

HasCrCard：是否有本行信用卡

IsActiveMember：是否活跃用户

EstimatedSalary：估计收入

Exited：是否已流失，这将作为我们的标签数据

具体要求：

（1）非特征值处理：Geography与Gender两列为非数值类型特征，需将其转换为数值型特征

（2）连续性变量离散化处理：决策树算法需要处理离散化的数据，需要将连续型变量先转化为离散型变量，

具体为信用分数（小于584为0，584到718之间为1，大于718为2）、

年龄（20以下为0，20到40之间1，大于40为2）、

存贷款情况（小于48000为0，48000到97198之间为1，大于97198为2）、

估计收入（小于51002为0，51002到149388之间为1，大于149388为2），等进行转换

（3）数据筛选：由于原数据集中训练数据类别不均衡，为了达到较好的模型效果，可以采用简单的欠采样方法，将多余的类别数据删掉。（也可使用其他方法解决类别不平衡问题）

（4）数据预处理：第1列为编号，对决策树模型无意义，去除，将剩余列作为特征项，其中标签数据为 Exited列

（5）划分训练集和测试集，将数据集按4:1分为训练集和测试集

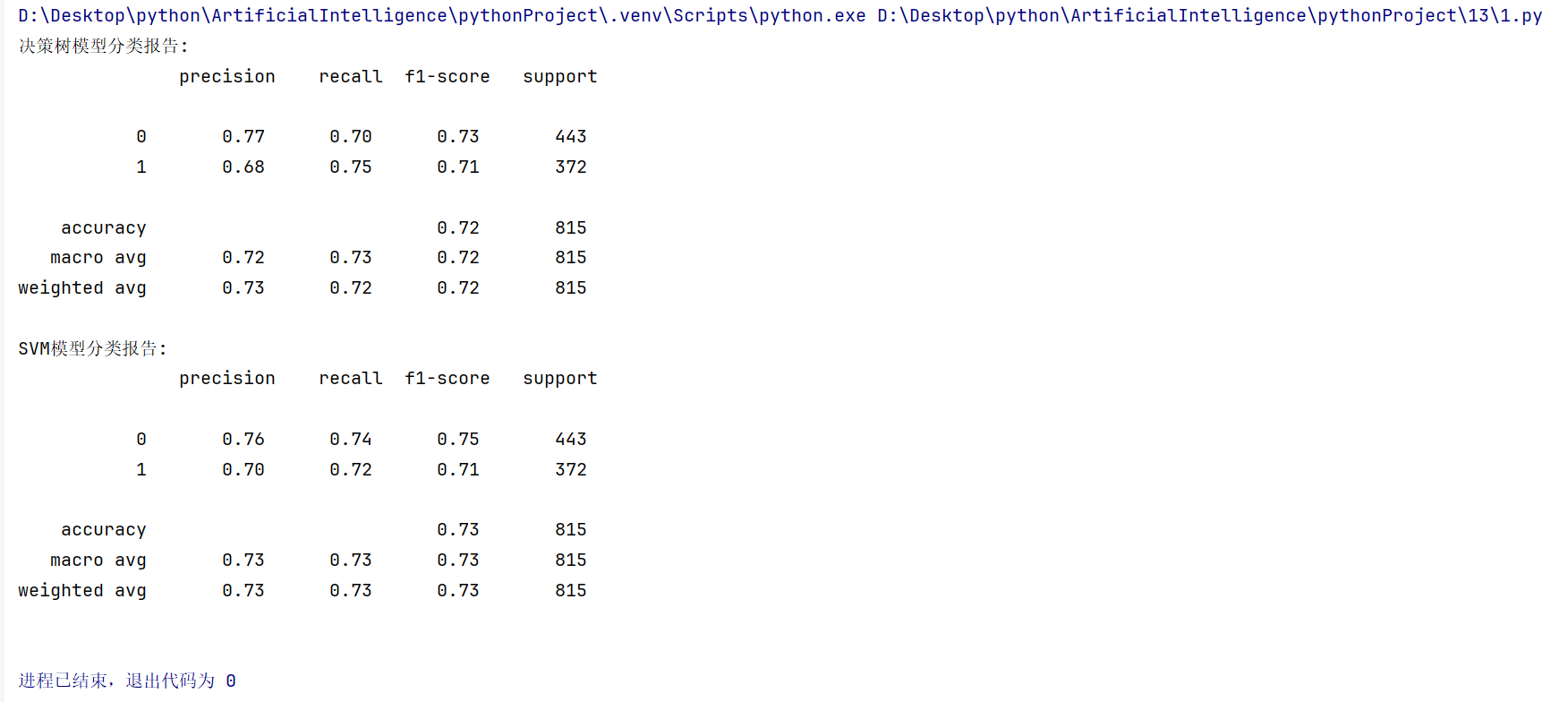
（6）定义决策树模型，训练数据，输出模型得分，并进行模型优化。

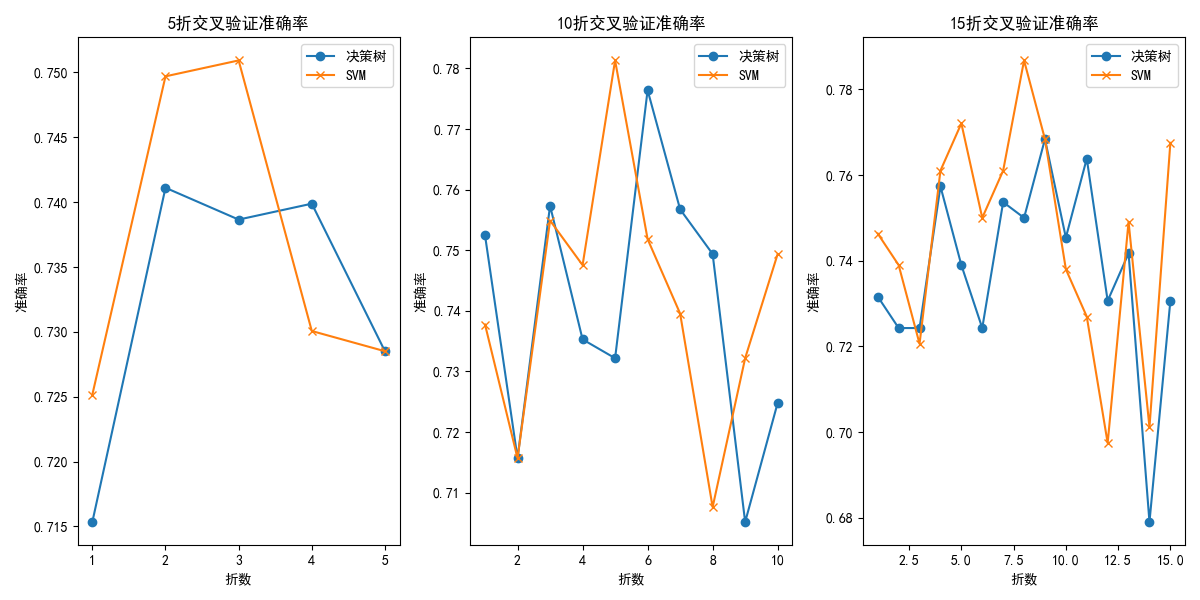
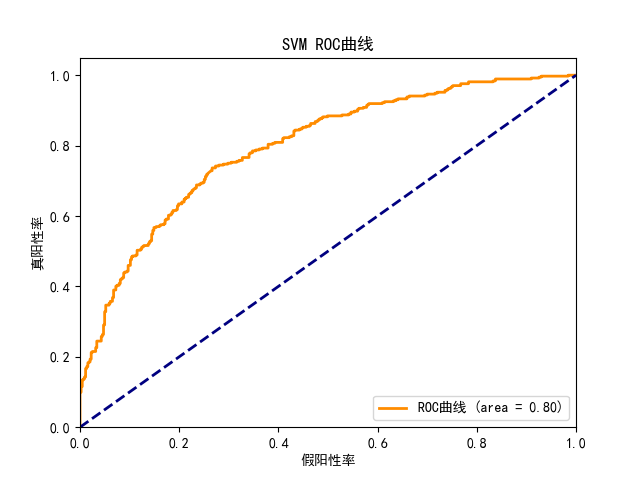
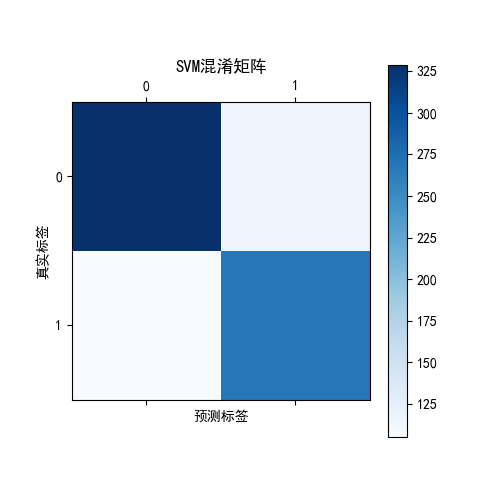
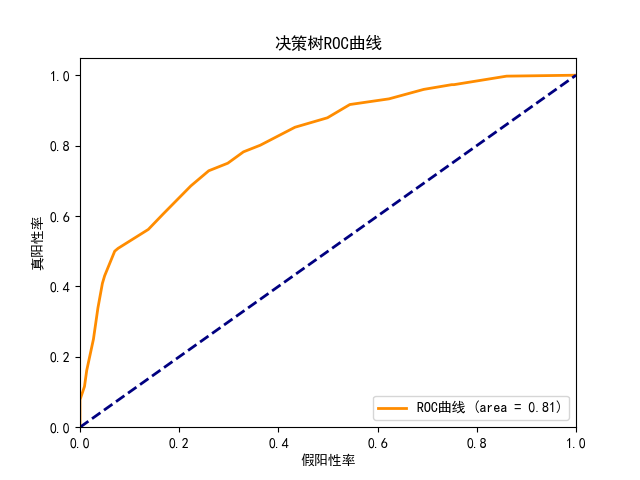
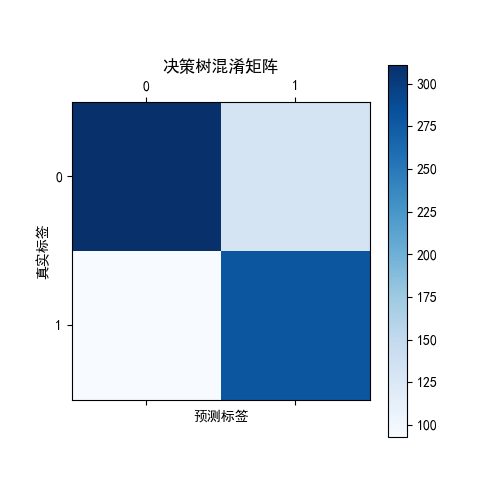
（7）模型评估，画混淆矩阵图、ROC曲线图，输出精确率、召回率和正确率

（8）定义SVM模型，训练数据，画混淆矩阵，ROC曲线，输出精确率、召回率和正确率，并进行模型优化。

（9）比较两种模型的性能，选择一种模型进行K折交叉验证，将5折、10折、15折交叉验证的正确率(精确率)使用折线图（不同子图）显示

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import LabelEncoder  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, roc\_curve, auc, classification\_report  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.model\_selection import cross\_val\_score  
import matplotlib.font\_manager as fm  
  
# 设置字体为 SimHei，以支持中文  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签  
plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号  
# 导入数据  
data = pd.read\_csv('Churn-Modelling-new.csv',sep=',')  
# print(data.info)  
  
data.dropna(inplace=True)  
  
# print("缺失值检查:\n", data.isna().sum())  
# 删除无意义列  
data.drop(['RowNumber', 'CustomerId', 'Surname',], axis=1, inplace=True)  
  
# 使用LabelEncoder将非数值特征转化为数值特征  
le\_geography = LabelEncoder()  
le\_gender = LabelEncoder()  
  
data['Geography'] = le\_geography.fit\_transform(data['Geography'])  
data['Gender'] = le\_gender.fit\_transform(data['Gender'])  
  
# 信用分数离散化  
data['CreditScore'] = pd.cut(data['CreditScore'], bins=[0, 584, 718, float('inf')], labels=[0, 1, 2])  
  
# 年龄离散化  
data['Age'] = pd.cut(data['Age'], bins=[0, 20, 40, float('inf')], labels=[0, 1, 2])  
  
# 存贷款情况离散化  
# print((data['Balance'] == 0).sum())  
# print(data['Balance'].describe())  
data['Balance'] = pd.cut(data['Balance'], bins=[-1, 48000, 97198, float('inf')], labels=[0, 1, 2])  
# print(data['Balance'].unique())  
# 估计收入离散化  
data['EstimatedSalary'] = pd.cut(data['EstimatedSalary'], bins=[0, 51002, 149388, float('inf')], labels=[0, 1, 2])  
  
# 分离特征与标签  
X = data.drop('Exited', axis=1)  
y = data['Exited']  
  
# 使用欠采样方法处理类别不平衡问题  
class\_counts = data['Exited'].value\_counts()  
# print("每个类别的样本数量:\n", class\_counts)  
  
# 假设标签0的数量更多，即 class\_0 是多数类，class\_1 是少数类  
if class\_counts[0] > class\_counts[1]:  
 majority\_class = 0  
 minority\_class = 1  
else:  
 majority\_class = 1  
 minority\_class = 0  
  
# 提取不同类别的样本  
class\_majority = data[data['Exited'] == majority\_class]  
class\_minority = data[data['Exited'] == minority\_class]  
  
# 下采样  
class\_majority\_under = class\_majority.sample(len(class\_minority))  
data\_under = pd.concat([class\_majority\_under, class\_minority], axis=0)  
  
# 分离特征和标签  
X = data\_under.drop('Exited', axis=1)  
y = data\_under['Exited']  
  
  
# 划分训练集和测试集  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# 决策树模型  
dt\_model = DecisionTreeClassifier(criterion="gini",max\_depth=6,min\_samples\_split=200,random\_state=42)  
dt\_model.fit(X\_train, y\_train)  
y\_pred\_dt = dt\_model.predict(X\_test)  
  
conf\_matrix\_dt = confusion\_matrix(y\_test, y\_pred\_dt)  
fpr\_dt, tpr\_dt, \_ = roc\_curve(y\_test, dt\_model.predict\_proba(X\_test)[:, 1])  
roc\_auc\_dt = auc(fpr\_dt, tpr\_dt)  
  
print("决策树模型分类报告:\n", classification\_report(y\_test, y\_pred\_dt, zero\_division=1))  
  
# 绘制决策树混淆矩阵  
plt.figure()  
plt.matshow(conf\_matrix\_dt, cmap=plt.cm.Blues)  
plt.title('决策树混淆矩阵')  
plt.colorbar()  
plt.ylabel('真实标签')  
plt.xlabel('预测标签')  
plt.show()  
  
# 绘制决策树ROC曲线  
plt.figure()  
plt.plot(fpr\_dt, tpr\_dt, color='darkorange', lw=2, label='ROC曲线 (area = %0.2f)' % roc\_auc\_dt)  
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('假阳性率')  
plt.ylabel('真阳性率')  
plt.title('决策树ROC曲线')  
plt.legend(loc="lower right")  
plt.show()  
  
# print(X.isna().sum())  
# SVM模型  
  
# print(X.shape)  
svm\_model = SVC(C=10,gamma=0.1,probability=True, random\_state=42)  
svm\_model.fit(X\_train, y\_train)  
y\_pred\_svm = svm\_model.predict(X\_test)  
  
conf\_matrix\_svm = confusion\_matrix(y\_test, y\_pred\_svm)  
fpr\_svm, tpr\_svm, \_ = roc\_curve(y\_test, svm\_model.predict\_proba(X\_test)[:, 1])  
roc\_auc\_svm = auc(fpr\_svm, tpr\_svm)  
  
print("SVM模型分类报告:\n", classification\_report(y\_test, y\_pred\_svm, zero\_division=1))  
  
# 绘制SVM混淆矩阵  
plt.figure()  
plt.matshow(conf\_matrix\_svm, cmap=plt.cm.Blues)  
plt.title('SVM混淆矩阵')  
plt.colorbar()  
plt.ylabel('真实标签')  
plt.xlabel('预测标签')  
plt.show()  
  
# 绘制SVM ROC曲线  
plt.figure()  
plt.plot(fpr\_svm, tpr\_svm, color='darkorange', lw=2, label='ROC曲线 (area = %0.2f)' % roc\_auc\_svm)  
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('假阳性率')  
plt.ylabel('真阳性率')  
plt.title('SVM ROC曲线')  
plt.legend(loc="lower right")  
plt.show()  
  
# 决策树交叉验证  
dt\_scores = cross\_val\_score(dt\_model, X, y, cv=5)  
dt\_scores\_10 = cross\_val\_score(dt\_model, X, y, cv=10)  
dt\_scores\_15 = cross\_val\_score(dt\_model, X, y, cv=15)  
  
# SVM交叉验证  
svm\_scores = cross\_val\_score(svm\_model, X, y, cv=5)  
svm\_scores\_10 = cross\_val\_score(svm\_model, X, y, cv=10)  
svm\_scores\_15 = cross\_val\_score(svm\_model, X, y, cv=15)  
  
# 绘制交叉验证结果  
plt.figure(figsize=(12, 6))  
  
plt.subplot(1, 3, 1)  
plt.plot(range(1, 6), dt\_scores, marker='o', label='决策树')  
plt.plot(range(1, 6), svm\_scores, marker='x', label='SVM')  
plt.xlabel('折数')  
plt.ylabel('准确率')  
plt.title('5折交叉验证准确率')  
plt.legend()  
  
plt.subplot(1, 3, 2)  
plt.plot(range(1, 11), dt\_scores\_10, marker='o', label='决策树')  
plt.plot(range(1, 11), svm\_scores\_10, marker='x', label='SVM')  
plt.xlabel('折数')  
plt.ylabel('准确率')  
plt.title('10折交叉验证准确率')  
plt.legend()  
  
plt.subplot(1, 3, 3)  
plt.plot(range(1, 16), dt\_scores\_15, marker='o', label='决策树')  
plt.plot(range(1, 16), svm\_scores\_15, marker='x', label='SVM')  
plt.xlabel('折数')  
plt.ylabel('准确率')  
plt.title('15折交叉验证准确率')  
plt.legend()  
  
plt.tight\_layout()  
plt.show()





案例2：电影推荐算法

具体要求：

（1）读取数据(json格式转换为字典格式/excel格式数据合并)

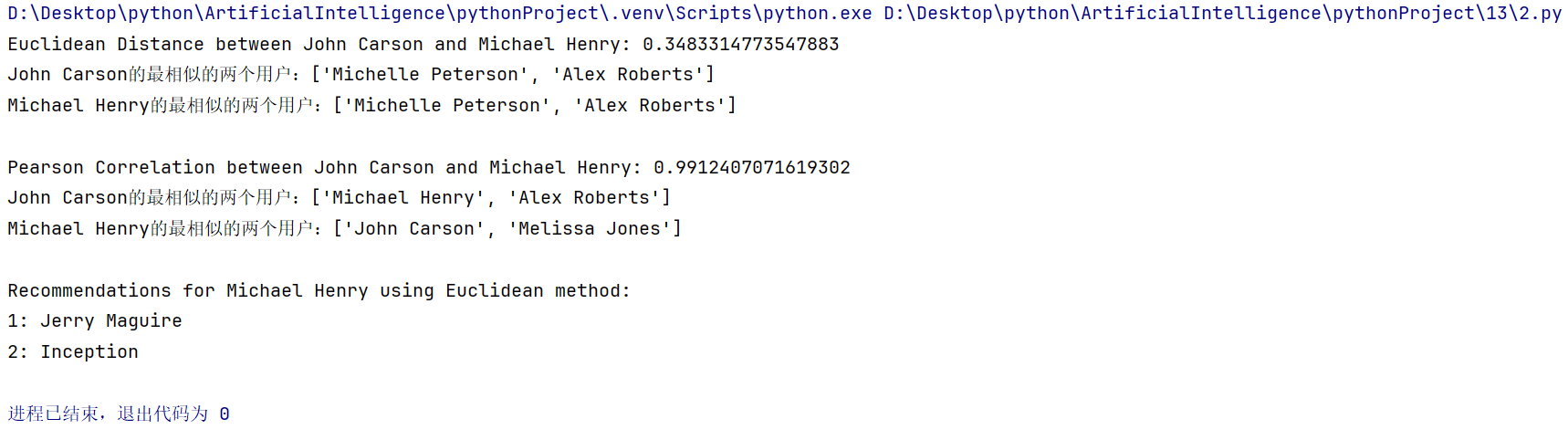
（2）选择两个用户，判断两个用户是否有共同评过分的电影，如果有，计算欧式距离，寻找数据集中的相似用户

（3）选择两个用户，判断两个用户是否有共同评过分的电影，如果有，计算皮尔孙相关系数，寻找数据集中的相似用户

（4）给定其中一个用户，任选一种方法为给定用户生成2部电影推荐。

## json格式

import json  
import math  
  
import numpy as np  
from scipy.stats import pearsonr  
  
# def load\_data(json\_data):  
# return json.loads(json\_data)  
import os  
  
  
def load\_data(filepath):  
 if not os.path.exists(filepath):  
 raise FileNotFoundError(f"No such file: '{filepath}'")  
  
 with open(filepath, 'r', encoding='utf-8') as file:  
 json\_data = file.read()  
  
 if not json\_data.strip(): # 检查文件是否为空  
 raise ValueError("File is empty")  
  
 return json.loads(json\_data)  
  
  
import math  
from scipy.stats import pearsonr  
  
def sort\_movie\_ratings\_desc(data):  
 sorted\_data = {user: dict(sorted(ratings.items(), key=lambda x: x[1], reverse=True))  
 for user, ratings in data.items()}  
 return sorted\_data  
  
import numpy as np  
  
def euclidean\_distance(user1\_ratings, user2\_ratings):  
 common\_movies = set(user1\_ratings.keys()).intersection(set(user2\_ratings.keys()))  
 if not common\_movies:  
 return float('inf') # No common movies, distance is infinite  
  
 squared\_differences = []  
 for movie in common\_movies:  
 squared\_differences.append(np.square(user1\_ratings[movie] - user2\_ratings[movie]))  
 return 1 / (1 + np.sqrt(np.sum(squared\_differences)))  
  
  
  
def pearson\_correlation(user1\_ratings, user2\_ratings):  
 # 找到共同评分的项目  
 common\_movies = set(user1\_ratings.keys()).intersection(set(user2\_ratings.keys()))  
 if len(common\_movies) < 2:  
 return 0 # 如果共同评分项少于2，返回0表示无效相关性  
  
 # 提取共同评分项的评分  
 user1\_scores = [user1\_ratings[movie] for movie in common\_movies]  
 user2\_scores = [user2\_ratings[movie] for movie in common\_movies]  
  
 # 检查是否为常量数组  
 if np.all(np.array(user1\_scores) == user1\_scores[0]) or np.all(np.array(user2\_scores) == user2\_scores[0]):  
 return 0 # 如果数组是常量数组，返回0表示无效相关性  
  
 # 计算并返回 Pearson 相关系数  
 correlation, \_ = pearsonr(user1\_scores, user2\_scores)  
 return correlation  
  
def find\_most\_similar\_users(data, target\_user, k=1, method='euclidean'):  
 distances = []  
 for user in data:  
 if user != target\_user:  
 if method == 'euclidean':  
 distance = euclidean\_distance(data[target\_user], data[user])  
 elif method == 'pearson':  
 distance = pearson\_correlation(data[target\_user], data[user])  
 distances.append((user, distance))  
  
 if method == 'euclidean':  
 # For Euclidean distance, the smaller the distance, the more similar the users  
 distances.sort(key=lambda x: x[1])  
 elif method == 'pearson':  
 # For Pearson correlation, the larger the correlation, the more similar the users  
 distances.sort(key=lambda x: x[1], reverse=True)  
  
 # Return the top k most similar users  
 return distances[:k]  
  
  
def recommend\_movies(data, target\_user, k, method='euclidean', num\_recommendations=2):  
 # print(k)  
 similar\_users = find\_most\_similar\_users(data, target\_user, k, method)  
 # print(similar\_users)  
 similar\_user\_names = [user for user, \_ in similar\_users]  
  
 movie\_recommendations = {}  
 for user in similar\_user\_names:  
 for movie, rating in data[user].items():  
 if movie not in data[target\_user]:  
 if movie not in movie\_recommendations:  
 movie\_recommendations[movie] = []  
 movie\_recommendations[movie].append(rating)  
  
 for movie in movie\_recommendations:  
 movie\_recommendations[movie] = np.mean(movie\_recommendations[movie])  
  
 recommended\_movies = sorted(movie\_recommendations.items(), key=lambda x: x[1], reverse=True)  
  
 return [name for name,\_ in recommended\_movies[:num\_recommendations]]  
  
  
def print\_similiar\_users(data, target\_user, k=2,method='euclidean'):  
 similar\_users = find\_most\_similar\_users(data, target\_user, k,method)  
 # 取出前 k 个最相似用户的名称  
 top\_k\_similar\_users = [user for user, \_ in similar\_users[:k]]  
 print(f"{target\_user}的最相似的两个用户：{top\_k\_similar\_users}")  
  
  
# Example usage  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 测试加载 JSON 数据  
 try:  
 data = load\_data('movie\_ratings.json')  
 # print(data)  
 except Exception as e:  
 print(f"An error occurred: {e}")  
 data=sort\_movie\_ratings\_desc(data)  
 # print(data)  
 user1 = 'John Carson'  
 user2 = 'Michael Henry'  
  
 # Calculate Euclidean distance  
 euclidean\_dist = euclidean\_distance(data[user1], data[user2])  
 if euclidean\_dist != float('inf'):  
 print(f"Euclidean Distance between {user1} and {user2}: {euclidean\_dist}")  
 print\_similiar\_users(data, user1)  
 print\_similiar\_users(data, user2)  
 print()  
 # Calculate Pearson correlation  
 pearson\_corr = pearson\_correlation(data[user1], data[user2])  
 if pearson\_corr != 0:  
 print(f"Pearson Correlation between {user1} and {user2}: {pearson\_corr}")  
 print\_similiar\_users(data, user1,method='pearson')  
 print\_similiar\_users(data, user2,method='pearson')  
 print()  
  
 # Generate movie recommendations  
 target\_user = user2  
 recommendations = recommend\_movies(data, target\_user,k=len(data),method='euclidean')  
 print(f"Recommendations for {target\_user} using Euclidean method:")  
 for i, recommendation in enumerate(recommendations, start=1):  
 print(f"{i}: {recommendation}")



## excel格式

# import pandas as pd  
#  
import os  
  
import numpy as np  
import pandas as pd  
import json  
from scipy.stats import pearsonr  
  
  
# 读取 CSV 数据  
def load\_data(csv\_file):  
 df = pd.read\_csv(csv\_file)  
 return df  
  
  
from scipy.stats import pearsonr  
  
  
def sort\_movie\_ratings\_desc(data):  
 sorted\_data = {user: dict(sorted(ratings.items(), key=lambda x: x[1], reverse=True))  
 for user, ratings in data.items()}  
 return sorted\_data  
  
  
  
def euclidean\_distance(user1\_ratings, user2\_ratings):  
 common\_movies = set(user1\_ratings.keys()).intersection(set(user2\_ratings.keys()))  
 if not common\_movies:  
 return float('inf') # No common movies, distance is infinite  
  
 squared\_differences = []  
 for movie in common\_movies:  
 squared\_differences.append(np.square(user1\_ratings[movie] - user2\_ratings[movie]))  
 return 1 / (1 + np.sqrt(np.sum(squared\_differences)))  
  
  
def pearson\_correlation(user1\_ratings, user2\_ratings):  
 # 找到共同评分的项目  
 common\_movies = set(user1\_ratings.keys()).intersection(set(user2\_ratings.keys()))  
 if len(common\_movies) < 2:  
 return 0 # 如果共同评分项少于2，返回0表示无效相关性  
  
 # 提取共同评分项的评分  
 user1\_scores = [user1\_ratings[movie] for movie in common\_movies]  
 user2\_scores = [user2\_ratings[movie] for movie in common\_movies]  
  
 # 检查是否为常量数组  
 if np.all(np.array(user1\_scores) == user1\_scores[0]) or np.all(np.array(user2\_scores) == user2\_scores[0]):  
 return 0 # 如果数组是常量数组，返回0表示无效相关性  
  
 # 计算并返回 Pearson 相关系数  
 correlation, \_ = pearsonr(user1\_scores, user2\_scores)  
 return correlation  
  
  
def find\_most\_similar\_users(data, target\_user, k=1, method='euclidean'):  
 distances = []  
 for user in data:  
 if user != target\_user:  
 if method == 'euclidean':  
 distance = euclidean\_distance(data[target\_user], data[user])  
 elif method == 'pearson':  
 distance = pearson\_correlation(data[target\_user], data[user])  
 distances.append((user, distance))  
  
 if method == 'euclidean':  
 # For Euclidean distance, the smaller the distance, the more similar the users  
 distances.sort(key=lambda x: x[1])  
 elif method == 'pearson':  
 # For Pearson correlation, the larger the correlation, the more similar the users  
 distances.sort(key=lambda x: x[1], reverse=True)  
  
 # Return the top k most similar users  
 return distances[:k]  
  
  
def recommend\_movies(data, target\_user, k, method='euclidean', num\_recommendations=2):  
 # print(k)  
 similar\_users = find\_most\_similar\_users(data, target\_user, k, method)  
 # print(similar\_users)  
 similar\_user\_names = [user for user, \_ in similar\_users]  
  
 movie\_recommendations = {}  
 for user in similar\_user\_names:  
 for movie, rating in data[user].items():  
 if movie not in data[target\_user]:  
 if movie not in movie\_recommendations:  
 movie\_recommendations[movie] = []  
 movie\_recommendations[movie].append(rating)  
  
 for movie in movie\_recommendations:  
 movie\_recommendations[movie] = np.mean(movie\_recommendations[movie])  
  
 recommended\_movies = sorted(movie\_recommendations.items(), key=lambda x: x[1], reverse=True)  
  
 return [name for name, \_ in recommended\_movies[:num\_recommendations]]  
  
  
def print\_similiar\_users(data, target\_user, k=2, method='euclidean'):  
 similar\_users = find\_most\_similar\_users(data, target\_user, k, method)  
 # 取出前 k 个最相似用户的名称  
 top\_k\_similar\_users = [user for user, \_ in similar\_users[:k]]  
 print(f"{target\_user}的最相似的两个用户：{top\_k\_similar\_users}")  
def load\_data\_json(filepath):  
 if not os.path.exists(filepath):  
 raise FileNotFoundError(f"No such file: '{filepath}'")  
  
 with open(filepath, 'r', encoding='utf-8') as file:  
 json\_data = file.read()  
  
 if not json\_data.strip(): # 检查文件是否为空  
 raise ValueError("File is empty")  
  
 return json.loads(json\_data)  
  
  
# 将 CSV 数据转换为用户-电影评分的字典格式  
def convert\_to\_dict(df):  
 user\_ratings = {}  
 for user\_id, group in df.groupby('userId'):  
 user\_ratings[user\_id] = group.set\_index('title')['rating'].to\_dict()  
 return user\_ratings  
  
  
# 示例代码  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 读取movies.csv和ratings.csv文件  
 movies\_df = pd.read\_csv('movies.csv')  
 ratings\_df = pd.read\_csv('ratings.csv')  
  
 # 合并两个DataFrame，使用movieId作为关联键  
 merged\_df = pd.merge(ratings\_df, movies\_df, on='movieId')  
  
 # 将合并后的DataFrame保存为新的CSV文件  
 merged\_df.to\_csv('merged\_movies\_ratings.csv', index=False)  
 csv\_file = 'merged\_movies\_ratings.csv'  
 data = load\_data(csv\_file)  
 user\_ratings = convert\_to\_dict(data)  
 # 将字典数据写入JSON文件  
 filename = 'data.json'  
 with open(filename, 'w', encoding='utf-8') as file:  
 json.dump(user\_ratings, file, ensure\_ascii=False, indent=4)  
 print(user\_ratings)  
  
 data=load\_data\_json('data.json')  
 data = sort\_movie\_ratings\_desc(data)  
 with open(filename, 'w', encoding='utf-8') as file:  
 json.dump(data, file, ensure\_ascii=False, indent=4)  
 data = load\_data\_json('data.json')  
   
 user1 = "1"  
 user2 = "7"  
  
 euclidean\_dist = euclidean\_distance(data[user1], data[user2])  
 if euclidean\_dist != float('inf'):  
 print(f"Euclidean Distance between {user1} and {user2}: {euclidean\_dist}")  
 print\_similiar\_users(data, user1)  
 print\_similiar\_users(data, user2)  
 print()  
 # Calculate Pearson correlation  
 pearson\_corr = pearson\_correlation(data[user1], data[user2])  
 if pearson\_corr != 0:  
 print(f"Pearson Correlation between {user1} and {user2}: {pearson\_corr}")  
 print\_similiar\_users(data, user1, method='pearson')  
 print\_similiar\_users(data, user2, method='pearson')  
 print()  
  
 # Generate movie recommendations  
 target\_user = user2  
 recommendations = recommend\_movies(data, target\_user, k=12, method='euclidean',num\_recommendations=2)  
 print(f"Recommendations for {target\_user} using Euclidean method:")  
 for i, recommendation in enumerate(recommendations, start=1):  
 print(f"{i}: {recommendation}")

