import pandas as pd

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import seaborn as sns

# 加载鸢尾花数据集

iris = load\_iris()

data = iris.data

target = iris.target

target\_names = iris.target\_names

# 将数据转换为DataFrame

df = pd.DataFrame(data, columns=iris.feature\_names)

df['target'] = target

# 数据标准化

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

# 使用K-means算法进行聚类

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(data\_scaled)

clusters = kmeans.labels\_

# 将聚类结果加入到DataFrame

df['cluster'] = clusters

# 可视化聚类结果

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

sns.scatterplot(data=df, x='sepal length (cm)', y='sepal width (cm)', hue='cluster', palette='viridis', style='target')

plt.title('K-means Clustering of Iris Data (Sepal Length vs Sepal Width)')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.legend(title='Cluster')

plt.show()

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

sns.scatterplot(data=df, x='petal length (cm)', y='petal width (cm)', hue='cluster', palette='viridis', style='target')

plt.title('K-means Clustering of Iris Data (Petal Length vs Petal Width)')

plt.xlabel('Petal Length (cm)')

plt.ylabel('Petal Width (cm)')

plt.legend(title='Cluster')

plt.show()

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# 读取CSV文件

data = pd.read\_csv('info.csv')

# 清洗数据，去除多余的空格或特殊字符

data.columns = data.columns.str.strip()

data['Height'] = data['Height'].astype(float)

data['Weight'] = data['Weight'].astype(float)

data['Type'] = data['Type'].str.strip()

# 提取特征和标签

X = data[['Height', 'Weight']].values

y = data['Type'].values

# 输入个人信息

height = float(input("请输入您的身高(cm): "))

weight = float(input("请输入您的体重(kg): "))

# 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)

# 逻辑回归

logistic\_model = LogisticRegression()

logistic\_model.fit(X\_train, y\_train)

# 决策树

tree\_model = DecisionTreeClassifier()

tree\_model.fit(X\_train, y\_train)

# KNN

knn\_model = KNeighborsClassifier(n\_neighbors=5)

knn\_model.fit(X\_train, y\_train)

# 将个人信息转换为模型输入格式

user\_data = np.array([[height, weight]])

# 使用不同的模型进行预测

logistic\_pred = logistic\_model.predict(user\_data)[0]

tree\_pred = tree\_model.predict(user\_data)[0]

knn\_pred = knn\_model.predict(user\_data)[0]

# 输出预测结果

print(f"逻辑回归预测结果: {logistic\_pred}")

print(f"决策树预测结果: {tree\_pred}")

print(f"KNN预测结果: {knn\_pred}")

# 预测测试集

logistic\_test\_pred = logistic\_model.predict(X\_test)

tree\_test\_pred = tree\_model.predict(X\_test)

knn\_test\_pred = knn\_model.predict(X\_test)

# 计算准确率

logistic\_accuracy = accuracy\_score(y\_test, logistic\_test\_pred)

tree\_accuracy = accuracy\_score(y\_test, tree\_test\_pred)

knn\_accuracy = accuracy\_score(y\_test, knn\_test\_pred)

print(f"逻辑回归准确率: {logistic\_accuracy:.2f}")

print(f"决策树准确率: {tree\_accuracy:.2f}")

print(f"KNN准确率: {knn\_accuracy:.2f}")

# 显示结果

fig, ax = plt.subplots()

algorithms = ['Logistic Regression', 'Decision Tree', 'KNN']

predictions = [logistic\_pred, tree\_pred, knn\_pred]

accuracies = [logistic\_accuracy, tree\_accuracy, knn\_accuracy]

# 绘制预测结果

ax.barh(algorithms, accuracies, color=['blue', 'green', 'orange'])

ax.set\_xlabel('Accuracy')

ax.set\_title('Accuracy of Different Algorithms')

# 显示准确率

for i, v in enumerate(accuracies):

ax.text(v + 0.01, i, f'{v:.2f}', color='black', va='center')

plt.show()