实验二：鸢尾花数据集分析实验报告

实验目的

通过鸢尾花数据集的分析，掌握机器学习中分类与聚类算法的应用。实验目标包括：

1. 使用两种聚类算法（K-means 和层次聚类）分析数据，并评价聚类效果。
2. 使用两种分类算法（逻辑回归和决策树）进行分类，并评估分类性能。
3. 比较不同算法的优劣，并分析参数选择与性能之间的关系。
4. 利用可视化方法展示实验结果，得出结论。

数据集描述

鸢尾花（Iris）数据集包含 150 个样本，分别来自三种鸢尾属植物：

- **Iris Setosa**
- **Iris Versicolour**
- **Iris Virginica**

每个样本包含以下四个特征（单位：cm）：

1. 花萼长度（sepal length）
2. 花萼宽度（sepal width）
3. 花瓣长度（petal length）
4. 花瓣宽度（petal width）

目标是根据特征对样本进行聚类和分类分析。

实验方法

数据预处理

1. 数据标准化：对特征进行标准化处理，确保均值为 0，标准差为 1。

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2. 数据划分：按 7:3 比例将数据集划分为训练集和测试集，用于分类任务。

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, random_state=42)
```

聚类分析

1. **K-means 聚类**
   - 参数设置：簇数 $k = 3$。
   - 使用欧几里得距离进行聚类。
   - 评估指标：轮廓系数（Silhouette Score）。

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)
print("K-means Silhouette Score: ", silhouette_score(X_scaled,
kmeans_labels))
```

2. **层次聚类**
   - 方法：Ward 连续法。
   - 使用层次聚类树状图（Dendrogram）展示分层关系。
   - 评估指标：轮廓系数。

```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
agglo = AgglomerativeClustering(n_clusters=3)
agglo_labels = agglo.fit_predict(X_scaled)
print("Agglomerative Clustering Silhouette Score: ",
silhouette_score(X_scaled, agglo_labels))

linked = linkage(X_scaled, 'ward')
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)
plt.title('Dendrogram of Hierarchical Clustering')
plt.show()
```

分类分析

1. **逻辑回归分类**
   - 最大迭代次数：200。
   - 通过交叉验证评估模型的泛化性能。

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
lr = LogisticRegression(max_iter=200)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
lr_cv_score = cross_val_score(lr, X_train, y_train, cv=5).mean()
print("Logistic Regression Accuracy: ", accuracy_score(y_test, y_pred_lr))
print("Logistic Regression CV Score: ", lr_cv_score)
```

2. **决策树分类**
   - 最大深度：3。
   - 通过交叉验证评估模型的泛化性能。

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
dt_cv_score = cross_val_score(dt, X_train, y_train, cv=5).mean()
print("Decision Tree Accuracy: ", accuracy_score(y_test, y_pred_dt))
print("Decision Tree CV Score: ", dt_cv_score)
```

可视化方法

1. PCA 降维后，绘制聚类结果的二维散点图。

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis',
edgecolor='k')
plt.title("K-means Clustering")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```

2. 绘制分类模型的混淆矩阵，分析预测结果。

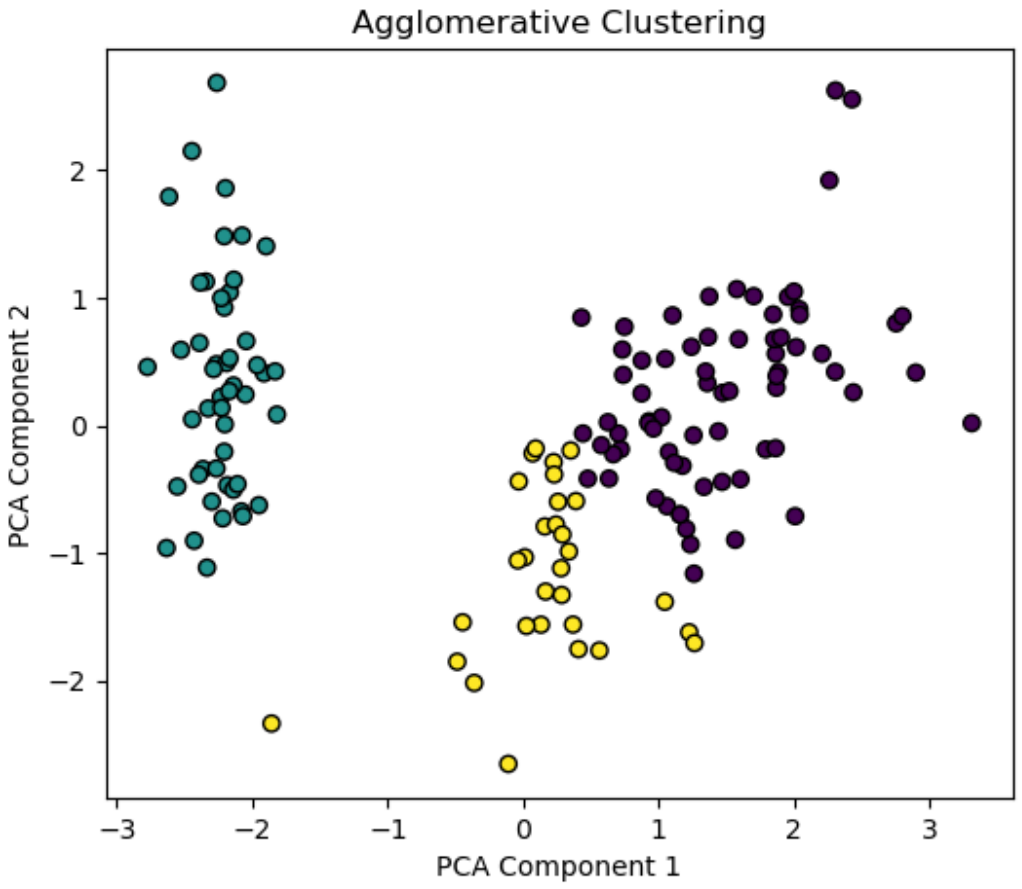
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)
    plt.title(title)
```

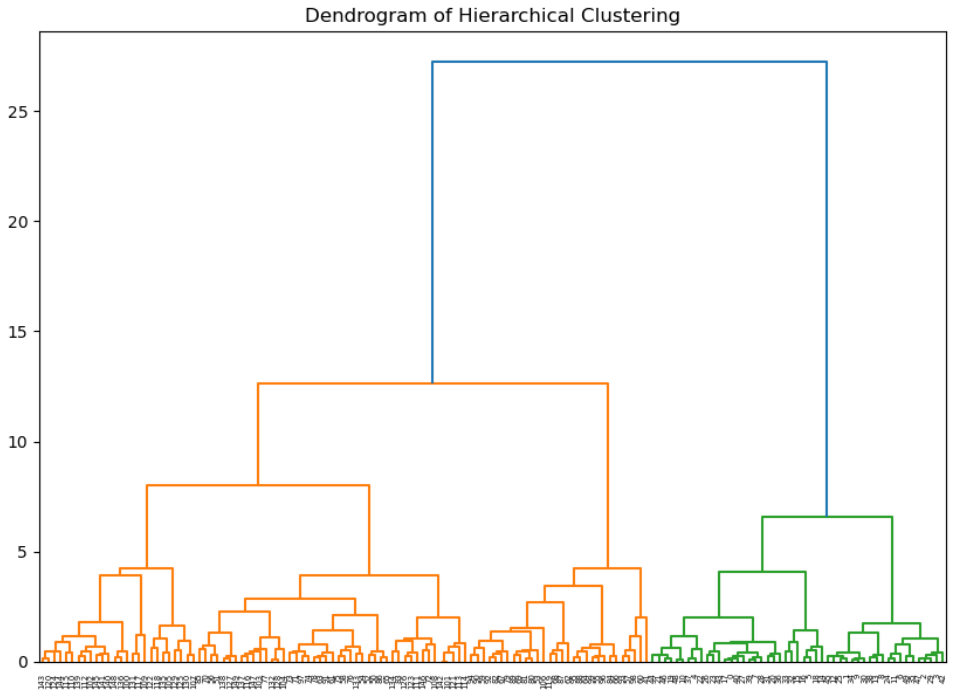
```
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

plot_confusion_matrix(y_test, y_pred_lr, 'Confusion Matrix - Logistic
Regression')
plot_confusion_matrix(y_test, y_pred_dt, 'Confusion Matrix - Decision Tree')
```

实验结果

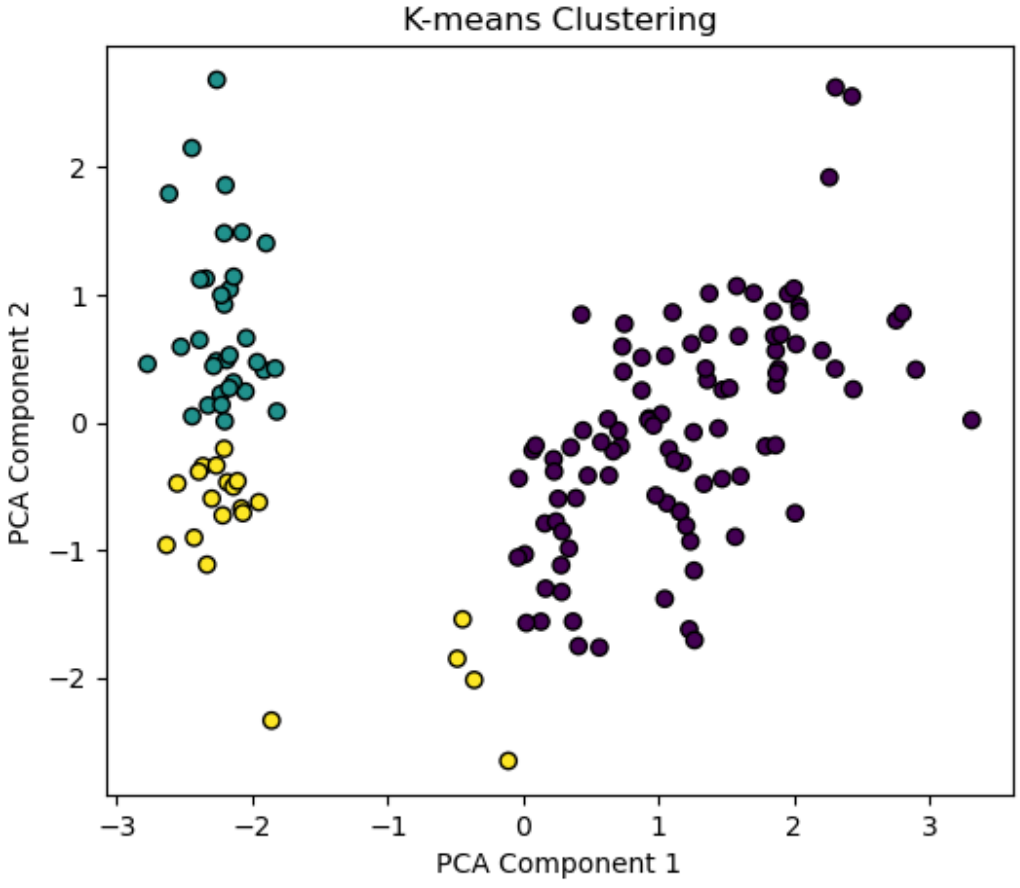
聚类结果

1. **层次聚类**
   - 轮廓系数： 0.4466890410285909





2. **K-means聚类**
   - 轮廓系数：0.4798814508199817

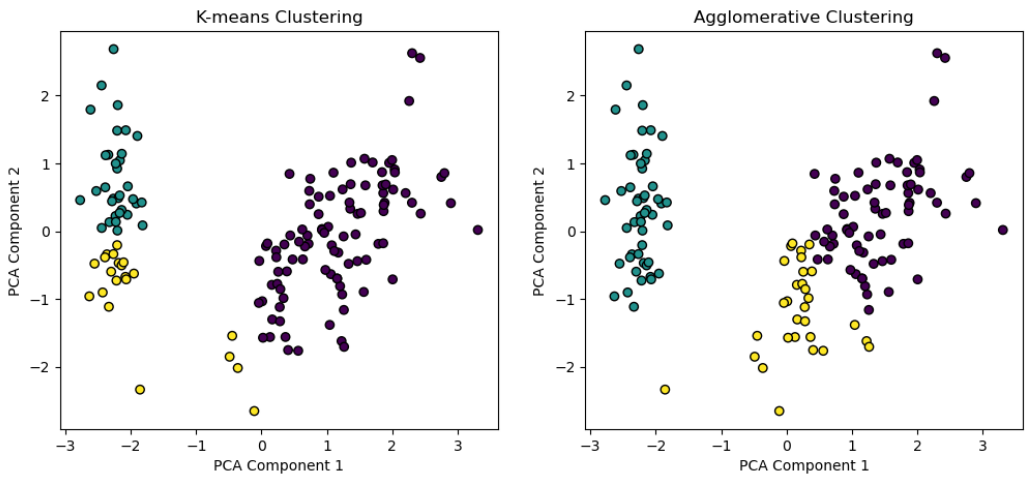


分类结果

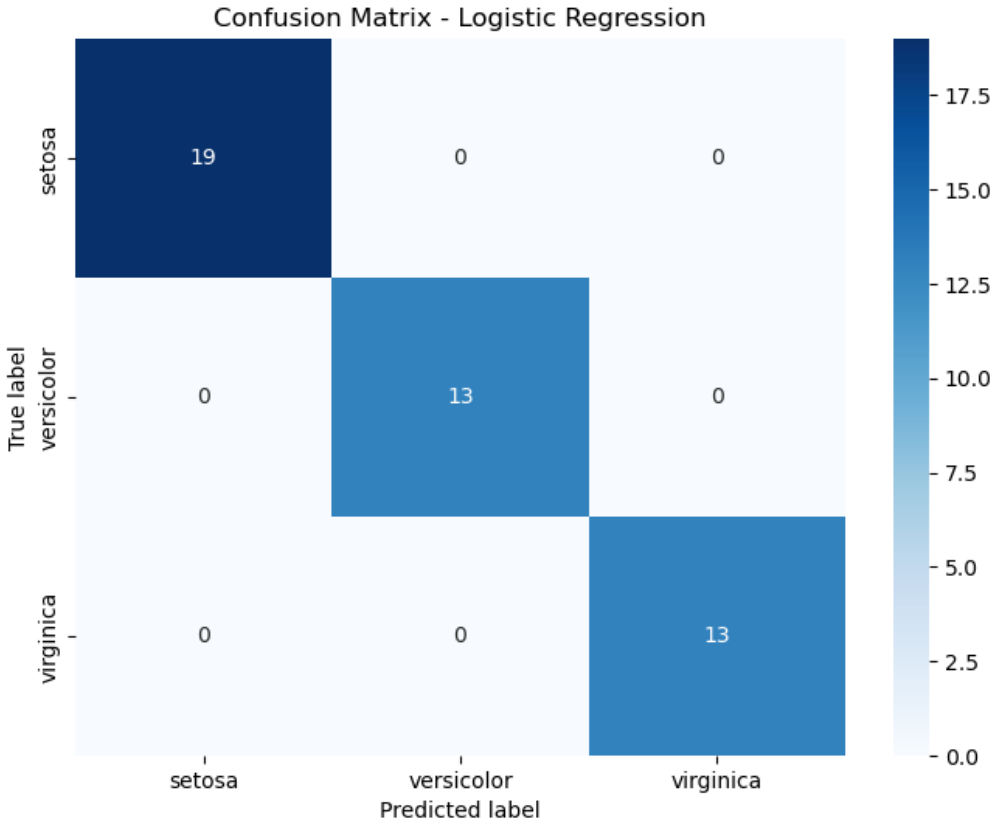
1. **逻辑回归分类**
   - 测试集准确率：100%
   - 交叉验证得分：94.28571428571428%
   - 混淆矩阵显示分类效果良好，完全正确
2. **决策树分类**
   - 测试集准确率：100%
   - 交叉验证得分：93.3333333333333%
   - 决策树模型易于解释，但略逊于逻辑回归。

可视化展示

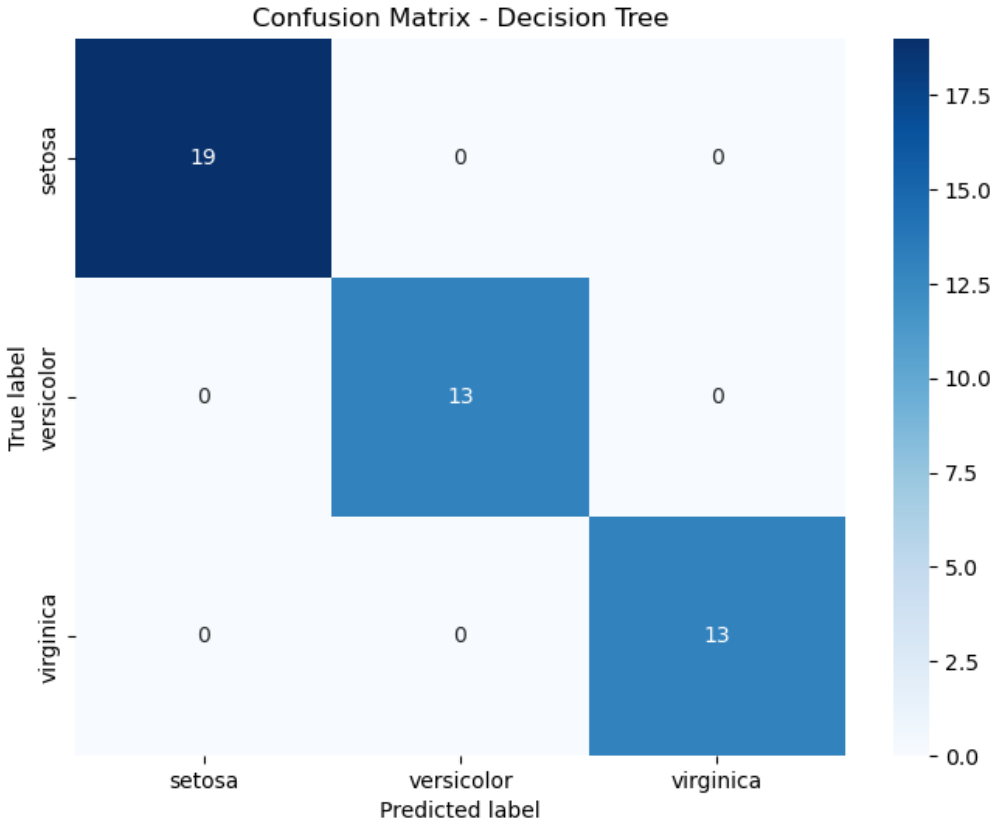
1. **聚类散点图**



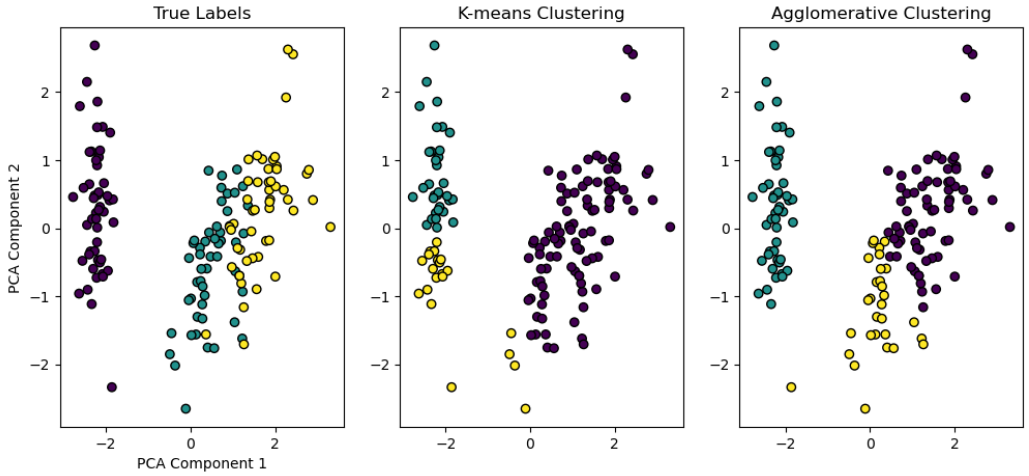
2. **逻辑回归混淆矩阵**



3. **决策树混沌矩阵**



4. **真实标签与聚类结果对比**



实验分析与讨论

1. **聚类算法**
   - K-means 和层次聚类在轮廓系数上的表现相近，说明两者对鸢尾花数据集的聚类效果相当。
   - K-means 聚类速度较快，适用于大规模数据；层次聚类通过树状图提供了分层信息，更适合小规模数据的分析。
2. **分类算法**
   - 逻辑回归的性能优于决策树，特别是在泛化能力上，交叉验证得分更高。
   - 决策树具有可解释性，但深度较小可能限制了其表现。
3. **参数选择**
   - K-means 聚类对簇数 k 敏感，应结合肘部法则确定最佳值。
   - 决策树深度的选择直接影响分类性能，应在防止过拟合和欠拟合之间找到平衡。

结论与展望

1. **结论**
   - 鸢尾花数据集因其特征清晰分离，使得聚类和分类算法均能取得较高性能。
   - K-means 聚类适合快速聚类任务，而层次聚类提供了更丰富的层次信息。
   - 逻辑回归因其简单高效，是分类任务的首选；决策树则提供了可解释性。
2. **展望**
   - 在更复杂的数据集上，可尝试引入其他算法（如随机森林、支持向量机）。
   - 进一步优化参数选择方法，如网格搜索或贝叶斯优化。
   - 增加特征工程步骤，探索更多数据特征的潜力。

附录

1. **主要代码片段**

```
# 导入必要的库

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import datasets

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.metrics import silhouette_score, accuracy_score,
classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.decomposition import PCA

from scipy.cluster.hierarchy import dendrogram, linkage



# 加载鸢尾花数据集

iris = datasets.load_iris()

X = iris.data  # 特征

y = iris.target  # 标签



# 数据标准化

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)
```

```
# 数据划分为训练集和测试集

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.3, random_state=42)



# PCA降维（用于可视化）

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)



# 聚类分析函数

def perform_clustering(X, X_pca):

    # K-means聚类

    kmeans = KMeans(n_clusters=3, random_state=42)

    kmeans_labels = kmeans.fit_predict(X)

    # 层次聚类

    agglomerative = AgglomerativeClustering(n_clusters=3)

    agglo_labels = agglomerative.fit_predict(X)



    # 评估聚类效果

    print("K-means Silhouette Score: ", silhouette_score(X, kmeans_labels))

    print("Agglomerative Clustering Silhouette Score: ", silhouette_score(X,
agglo_labels))



    # 可视化聚类结果

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
```

```
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis',
edgecolor='k')

    plt.title("K-means Clustering")

    plt.xlabel("PCA Component 1")

    plt.ylabel("PCA Component 2")

    plt.subplot(1, 2, 2)

    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=agglo_labels, cmap='viridis',
edgecolor='k')

    plt.title("Agglomerative Clustering")

    plt.xlabel("PCA Component 1")

    plt.ylabel("PCA Component 2")

    plt.show()



    return kmeans_labels, agglo_labels



kmeans_labels, agglo_labels = perform_clustering(X_scaled, X_pca)



# 单独展示 K-means 聚类结果

plt.figure(figsize=(6, 5))

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis',
edgecolor='k')

plt.title("K-means Clustering")

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.show()



# 单独展示层次聚类结果
```

```
plt.figure(figsize=(6, 5))

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=agglo_labels, cmap='viridis',
edgecolor='k')

plt.title("Agglomerative Clustering")

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")

plt.show()



# 层次聚类树状图

linked = linkage(X_scaled, 'ward')

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)

plt.title('Dendrogram of Hierarchical Clustering')

plt.show()



# 分类分析函数

def perform_classification(X_train, X_test, y_train, y_test):

    # 逻辑回归分类

    lr = LogisticRegression(max_iter=200)

    lr.fit(X_train, y_train)

    y_pred_lr = lr.predict(X_test)

    # 决策树分类

    dt = DecisionTreeClassifier(random_state=42, max_depth=3)

    dt.fit(X_train, y_train)
```

```
    y_pred_dt = dt.predict(X_test)



    # 交叉验证

    lr_cv_score = cross_val_score(lr, X_train, y_train, cv=5).mean()

    dt_cv_score = cross_val_score(dt, X_train, y_train, cv=5).mean()



    # 评估分类效果

    print("Logistic Regression Accuracy: ", accuracy_score(y_test,
y_pred_lr))

    print("Decision Tree Accuracy: ", accuracy_score(y_test, y_pred_dt))

    print("Logistic Regression CV Score: ", lr_cv_score)

    print("Decision Tree CV Score: ", dt_cv_score)

    # 打印详细的分类报告

    print("\nLogistic Regression Classification Report:\n",
classification_report(y_test, y_pred_lr))

    print("Decision Tree Classification Report:\n",
classification_report(y_test, y_pred_dt))



    return y_pred_lr, y_pred_dt



y_pred_lr, y_pred_dt = perform_classification(X_train, X_test, y_train,
y_test)



# 可视化混淆矩阵

def plot_confusion_matrix(y_true, y_pred, title):

    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8,6))
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=iris.target_names, yticklabels=iris.target_names)

    plt.title(title)

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

    plt.show()



plot_confusion_matrix(y_test, y_pred_lr, 'Confusion Matrix - Logistic
Regression')

plot_confusion_matrix(y_test, y_pred_dt, 'Confusion Matrix - Decision Tree')



# 聚类与真实标签的比较

plt.figure(figsize=(12, 5))

plt.subplot(1, 3, 1)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k')

plt.title("True Labels")

plt.xlabel("PCA Component 1")

plt.ylabel("PCA Component 2")



plt.subplot(1, 3, 2)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis',
edgecolor='k')

plt.title("K-means Clustering")



plt.subplot(1, 3, 3)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=agglo_labels, cmap='viridis',
edgecolor='k')
```

```
plt.title("Agglomerative Clustering")

plt.show()
```

3. **参考文献**
    - Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems.
    - UCI Machine Learning Repository: Iris Data Set.
    - Python 官方文档与相关库文档（sklearn, numpy, matplotlib）。