Assignment 3

July 9, 2023

1 Assesment 3

22975276 Lachlan Bassi

[1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import AgglomerativeClustering, KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.model_selection import GridSearchCV
from scipy.cluster.hierarchy import cut_tree
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import fcluster
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
```

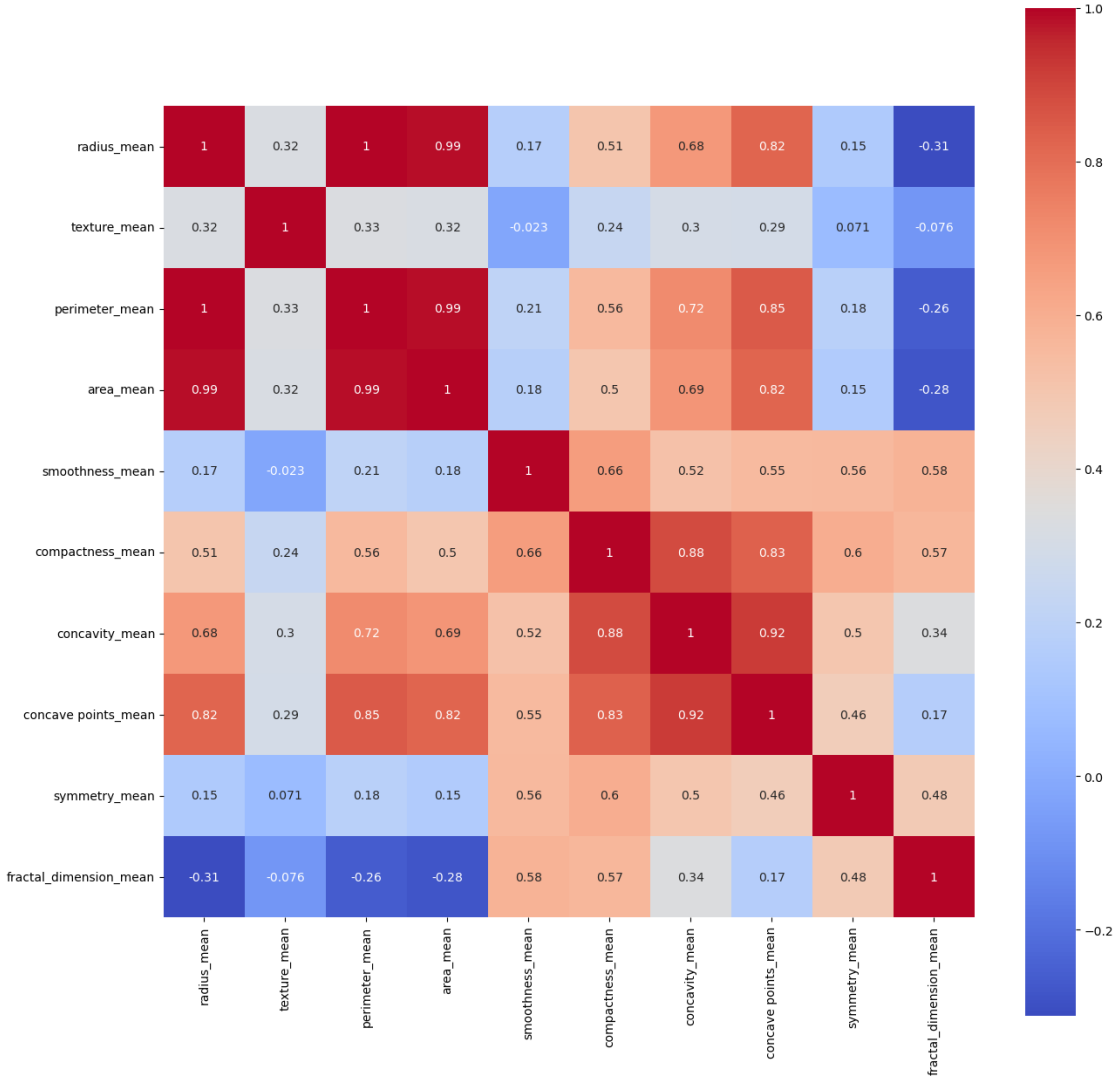
1.1 Part 1: A model for diagnosing cancer (60 marks)

1.1.1 Task 0:

Data Cleaning

[2]:
```
# Load data
data = pd.read_csv('breast-cancer.csv')
X = data.drop(columns=['id', 'diagnosis'])
y = data['diagnosis']
```

[3]:
```
# Compute the correlation matrix
df = pd.DataFrame(X)
corr_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(15, 15))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", square=True)
plt.show()
```



[4]:
```
threshold = 0.9

to_drop = []
while True:
    dropped = False
```

```
    for idx_i, col_i in enumerate(corr_matrix.columns):
        for idx_j, col_j in enumerate(corr_matrix.columns):
            if idx_i != idx_j and np.abs(corr_matrix.loc[col_i, col_j]) >=␣
 ↪threshold:
                to_drop.append(col_i)
                df = df.drop(col_i, axis=1)
                corr_matrix = df.corr()
                dropped = True
                break
        if dropped:
            break
    if not dropped:
        break

to_drop = list(set(to_drop))
print("Features to be dropped:", to_drop)

df.head()
X = df
```

```
Features to be dropped: ['perimeter_mean', 'concavity_mean', 'radius_mean']
```

[5]:
```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=123)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Highly correlated features were removed due to their overlapping information. Basically, these variables shared a significant portion of data that other features also depicted, which meant they were not contributing substantial novel insights. A threshold of overlap of 90% was set and the following features were chosen for exclusion:

1. Concavity mean
2. Perimeter mean
3. Radius mean

Next the data was split into training and testing sets and scaled using StandardScaler().

1.1.2 Task 1:

Build a logistic regression and a decision tree model to predict the tumour status. The presentation should include a comparison of the two models and a recommendation regarding which would be more appropriate in a clinical setting. Note: You should use the fundamental steps of a machine learning project (e.g. hyperparameters fine-tuning, cross-validation, etc.).

Task 1.1: Fitting a base Logistic and Decision Tree Model with no hyperparameters

[6]:
```
# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_pred_lr = log_reg.predict(X_test_scaled)
print('Logistic Regression Accuracy evaluated on test data: ',␣
 ↪accuracy_score(y_test, y_pred_lr))


# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_pred_lr = log_reg.predict(X_train_scaled)
print('Logistic Regression Accuracy evaluated on train data: ',␣
 ↪accuracy_score(y_train, y_pred_lr))

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_test_scaled)
print('Decision Tree Accuracy: ', accuracy_score(y_test, y_pred_dt))

# Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train_scaled, y_train)
y_pred_dt = dt.predict(X_train_scaled)
print('Decision Tree Accuracy evaluated on train data: ',␣
 ↪accuracy_score(y_train, y_pred_dt))
```

```
Logistic Regression Accuracy evaluated on test data:  0.9649122807017544
Logistic Regression Accuracy evaluated on train data:  0.9384615384615385
Decision Tree Accuracy:  0.9122807017543859
Decision Tree Accuracy evaluated on train data:  1.0
```

Comparing the performance of both models with no hyper parameters it appears the logistic regression has a higher accuracy on the testing set. Comparing the accuracy of the training and testing sets used on the logistic model it appears as though it underfits the training data which results in a higher accuracy on the testing set. Whereas the opposite appears to be true for the decision tree where the accuracy on the training data is exactly 1.0 and on the testing data it is much lower at 0.912 a clear sign of overfitting. Because of this without adding hyper-parameters logistic regression appears to be a better model due to its better performance on unseen data.

Task 1.2: Hyperparameters for logistic regression

Hyperparameters are settings that can be tuned to control the behavior of a machine learning algorithm. The choice of hyperparameters can have a significant impact on the performance of the model.

For Logistic Regression, the main hyperparameters include: **C:** This is the inverse of the regularisation strength. Smaller values specify stronger regularisation.

penalty: This is the type of regularisation applied to the model. Options are 'l1', 'l2', 'elasticnet', or 'none'.

solver: Algorithm to use in the optimisation problem. Options are 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'.

There were warnings indicating that the logistic regression model failed to train on some combinations of hyperparameters. Specifically, the 'newton-cg' and 'lbfgs' solvers only support 'l2' or 'none' penalties, but they were also attempted with the 'l1' penalty, which resulted in an error.

The solvers 'newton-cg', 'lbfgs', 'sag', and 'saga' handle 'l2' or no penalty, whereas 'liblinear' and 'saga' handle 'l1'. Therefore, the parameter grid was adjusted to only try the compatible combinations of penalties and solvers resulting in two different grid searches.

Next the best performing paramters were compared.

[7]:
```
# Define hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'],
    'max_iter': [10000]
}

# Include l2 penalty with 'newton-cg' and 'lbfgs' solvers
param_grid_l2 = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l2'],
    'solver': ['newton-cg', 'lbfgs', 'saga'],
    'max_iter': [10000]
}

# Initialise GridSearchCV for l1 and l2 penalties with 'liblinear' and 'saga'␣
 ↪solvers
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=3)
grid_search.fit(X_train_scaled, y_train)
print("Best parameters: ", grid_search.best_params_)

best_params = grid_search.best_params_

# Initialise GridSearchCV for l2 penalty with 'newton-cg' and 'lbfgs' solvers
grid_search_l2 = GridSearchCV(LogisticRegression(), param_grid_l2, cv=3)
grid_search_l2.fit(X_train_scaled, y_train)
print("Best parameters: ", grid_search_l2.best_params_)

best_params_l2 = grid_search_l2.best_params_
```

```
Best parameters:  {'C': 0.1, 'max_iter': 10000, 'penalty': 'l2', 'solver':
'liblinear'}
Best parameters:  {'C': 1, 'max_iter': 10000, 'penalty': 'l2', 'solver':
```

```
'newton-cg'}
```

[8]:
```
# Train logistic regression model with best parameters for liblinear solver
log_reg_liblinear = LogisticRegression(C=best_params['C'],␣
 ↪max_iter=best_params['max_iter'], penalty=best_params['penalty'],␣
 ↪solver=best_params['solver'])
log_reg_liblinear.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred_liblinear = log_reg_liblinear.predict(X_test_scaled)

#Print accuracy
print('Logistic Regression liblinear Accuracy: ', accuracy_score(y_test,␣
 ↪y_pred_liblinear))

# Print classification report
print("Classification Report for Logistic Regression (liblinear):")
print(classification_report(y_test, y_pred_liblinear))

# Predict on the test set
y_pred_liblinear = log_reg_liblinear.predict(X_train_scaled)

#Print accuracy
print('Logistic Regression liblinear Accuracy on training data: ',␣
 ↪accuracy_score(y_train, y_pred_liblinear))

# Train logistic regression model with best parameters for newton-cg solver
log_reg_newton = LogisticRegression(C=best_params_l2['C'],␣
 ↪max_iter=best_params_l2['max_iter'], penalty=best_params_l2['penalty'],␣
 ↪solver=best_params_l2['solver'])
log_reg_newton.fit(X_train_scaled, y_train)

# Predict on the test set
y_pred_newton = log_reg_newton.predict(X_test_scaled)

#Print accuracy
print('Logistic Regression Newton Accuracy: ', accuracy_score(y_test,␣
 ↪y_pred_newton))

# Print classification report
print("Classification Report for Logistic Regression (newton-cg):")
print(classification_report(y_test, y_pred_newton))

# Predict on the test set
y_pred_newton = log_reg_newton.predict(X_train_scaled)

#Print accuracy
```

```
print('Logistic Regression Newton Accuracy on training data: ',␣
 ↪accuracy_score(y_train, y_pred_newton))
```

```
Logistic Regression liblinear Accuracy:  0.9649122807017544
Classification Report for Logistic Regression (liblinear):
              precision    recall  f1-score   support

           B       0.95      1.00      0.97        73
           M       1.00      0.90      0.95        41

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114

Logistic Regression liblinear Accuracy on training data:  0.9384615384615385
Logistic Regression Newton Accuracy:  0.9649122807017544
Classification Report for Logistic Regression (newton-cg):
              precision    recall  f1-score   support

           B       0.95      1.00      0.97        73
           M       1.00      0.90      0.95        41

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114

Logistic Regression Newton Accuracy on training data:  0.9384615384615385
```

What was unexpected is that both models, even with different hyper parameters, had identical summary statistics with no clear seperator in any metric. Therefore the one selected was the liblinear model as it tends to do better on smaller datasets such as the airplane dataset used in this experiment. The reason these two had identical results could be explained by one of the following reasons:

1. Stability of Logistic Regression: Logistic regression is a robust and stable model. As a result, minor differences in hyperparameters often do not lead to substantial variations in the model performance.
2. Model Complexity: The dataset might not have been prone to overfitting, or the best model was relatively simple, leading to similar performance metrics with different C values.

The selected hyperparametes were: {'C': 0.1, 'max_iter': 10000, 'penalty': 'l2', 'solver': 'liblinear'} resulting in an accuracy of 0.9649122807017544 on the testing data.

Task 1.3: Hyperparameter Tuning for Decision Tree

Decision Trees, hyperparameters include: **max_depth:** The maximum depth of the tree.

min_samples_split: The minimum number of samples required to split an internal node.

min_samples_leaf: The minimum number of samples required to be at a leaf node.

max_features: The number of features to consider when looking for the best split.

[9]:
```
# Define hyperparameter grid
param_grid = {
    'max_depth':  [None] + list(range(1, 21)),
    'min_samples_split': list(range(1, 10)),
    'min_samples_leaf': list(range(1, 10)),
    'max_features': [None, 'sqrt', 'log2']
}

# Initialise GridSearchCV
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=123),␣
 ↪param_grid, cv=3)

# Fit and find best hyperparameters
grid_search.fit(X_train_scaled, y_train)

# Output best parameters
print("Best parameters: ", grid_search.best_params_)

# Use the best estimator to make predictions
dt_best = grid_search.best_estimator_
y_pred_dt_best = dt_best.predict(X_test_scaled)
print('Decision Tree Accuracy (with best hyperparameters): ',␣
 ↪accuracy_score(y_test, y_pred_dt_best))
```

```
Best parameters:  {'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf':
5, 'min_samples_split': 1}
Decision Tree Accuracy (with best hyperparameters):  0.9298245614035088
```

The best parameters for the decision tree model were: {'max_depth': 6, 'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 1} Which resulted in an accuracy of: 0.9298245614035088

[10]:
```
# Generate predictions for the decision tree model
y_pred_dt_best = dt_best.predict(X_test_scaled)

# Generate predictions for the logistic regression model
y_pred_liblinear = log_reg_liblinear.predict(X_test_scaled)

# Print classification report for the decision tree model
print("Classification Report for Decision Tree (with best hyperparameters):")
print(classification_report(y_test, y_pred_dt_best))

# Print classification report for the logistic regression model
print("Classification Report for Logistic Regression (liblinear):")
print(classification_report(y_test, y_pred_liblinear))
```

```
Classification Report for Decision Tree (with best hyperparameters):
```

```
              precision    recall  f1-score   support

           B       0.92      0.97      0.95        73
           M       0.95      0.85      0.90        41

    accuracy                           0.93       114
   macro avg       0.93      0.91      0.92       114
weighted avg       0.93      0.93      0.93       114

Classification Report for Logistic Regression (liblinear):
              precision    recall  f1-score   support

           B       0.95      1.00      0.97        73
           M       1.00      0.90      0.95        41

    accuracy                           0.96       114
   macro avg       0.97      0.95      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Task 1.4: Conclusion

When looking at the classification reports for the Decision Tree and Logistic Regression models, several observations can be made:

Overall Accuracy: The Logistic Regression model has a higher overall accuracy (96%) compared to the Decision Tree model (93%). This means that the Logistic Regression model correctly predicted more instances overall.

Precision: For class B, the precision is slightly higher for the Logistic Regression model, indicating that when it predicts class B, it is more often correct. For class M, the Logistic Regression model has a perfect precision score, meaning that whenever it predicted class M, it was always correct. This is better than the Decision Tree model, which has a slightly lower precision for both classes.

Recall: The recall for class B is perfect for the Logistic Regression model, while it is slightly less for the Decision Tree model. For class M, the recall is higher for the Decision Tree model, which indicates that it was able to identify more actual M cases.

F1-Score: The F1-score, which is a balance of precision and recall, is higher for the Logistic Regression model for both classes, indicating that it provides a better balance of precision and recall compared to the Decision Tree model.

Overall, the Logistic Regression model appears to perform better across most metrics in this comparison, with a higher overall accuracy, precision, and F1-score. However, it's also important to note where the Decision Tree model performs better: namely, it has a higher recall for class M, meaning it is better at identifying all actual M cases. But overall the logistic regression model should be used for clinical applications

Task 2: Describe the features that have a higher chance of impacting the prediction of the tumour status according to each of the two models. Discuss their similarities/differences.

[11]:
```
print('Logistic Regression coefficients: ', log_reg_newton.coef_)
print('Decision Tree feature importances: ', dt_best.feature_importances_)
print(df.columns)
```

```
Logistic Regression coefficients:  [[ 1.38413819  2.59937226  0.65324665
0.30860881  2.17249696  0.57451663
  -0.40154889]]
Decision Tree feature importances:  [0.11519609 0.07862293 0.00958916 0.00853315
0.78308072 0.00189528
 0.00308269]
Index(['texture_mean', 'area_mean', 'smoothness_mean', 'compactness_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean'],
      dtype='object')
```

Logistic Regression: One common approach to evaluate the features with the most impact on the model is to look at the coefficients of the logistic regression model. The higher the absolute value of the coefficient, the more important the feature is.

Logistic Regression coefficients ranked:

1. area_mean: It has the highest coefficient (2.59937226), indicating a strong positive impact on the tumor status prediction.
2. concave points_mean: It has a coefficient of 2.17249696, suggesting a significant positive influence.
3. texture_mean: It has a coefficient of 1.38413819, implying a moderately positive impact.
4. smoothness_mean: It has a coefficient of 0.65324665, indicating a weaker positive influence.
5. symmetry_mean: It has a coefficient of 0.57451663, suggesting a relatively weaker positive impact.
6. compactness_mean: It has a coefficient of 0.30860881, implying a minor positive influence.
7. fractal_dimension_mean: It has a coefficient of -0.40154889, indicating a negative impact (inverse relationship) on the tumor status prediction.

Decision Trees: Sklearn's DecisionTreeClassifier has a .feature_importances_ attribute after the model has been trained. This attribute directly gives the importance of each feature in making decisions in the tree.

Decision Tree feature importances ranked:

1. concave points_mean: It has the highest importance (0.78308072), indicating the most significant impact on the tumor status prediction.
2. texture_mean: It has an importance of 0.11519609, suggesting a moderately influential role.
3. area_mean: It has an importance of 0.07862293, implying a relatively weaker influence.
4. smoothness_mean: It has an importance of 0.00958916, indicating a minor impact.
5. compactness_mean: It has an importance of 0.00853315, suggesting a relatively weaker influence.
6. fractal_dimension_mean: It has an importance of 0.00308269, implying a minor impact.
7. symmetry_mean: It has an importance of 0.00189528, indicating a relatively weaker influence.

Summarising the similarities and differences between the two models in terms of feature importance:

Similarities:

Both models identify concave points_mean as the an important feature for predicting tumor status. Features smoothness_mean, compactness_mean, and fractal_dimension_mean have relatively lower importances/coefficients in both models.

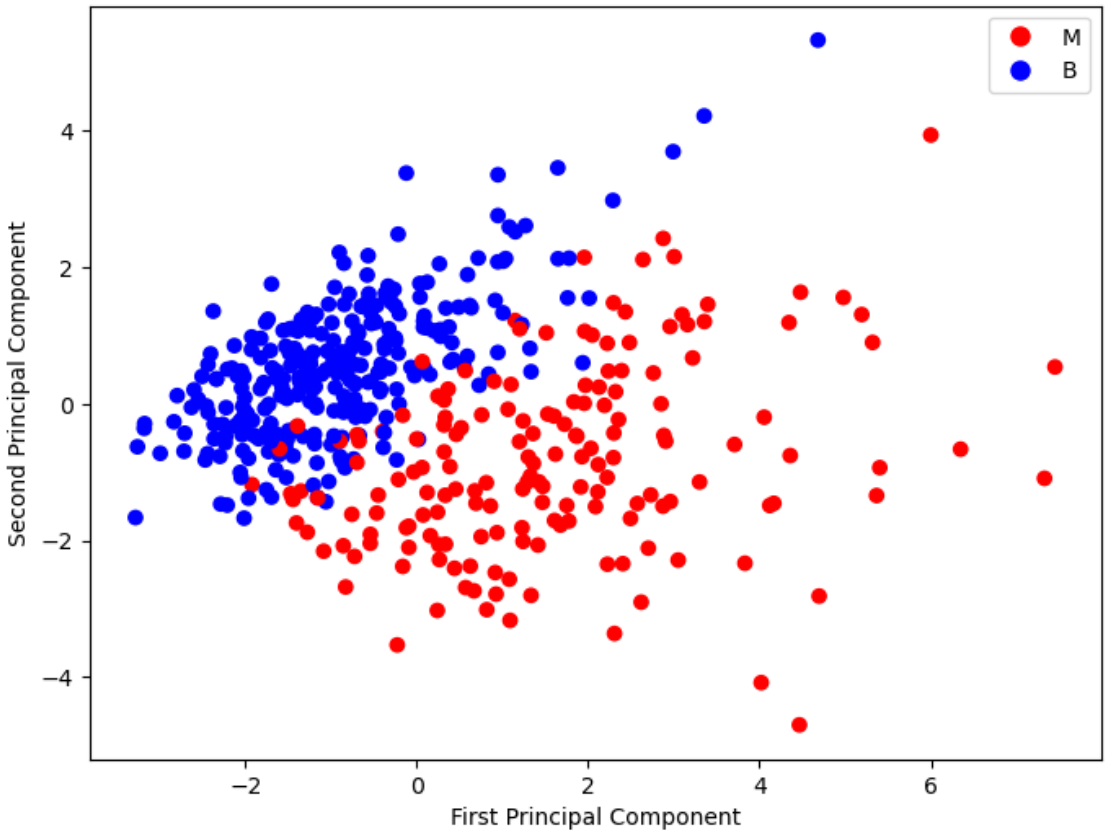
Differences:

The Logistic Regression model emphasises area_mean as the most important feature, while the Decision Tree model assigns higher importance to texture_mean. The Logistic Regression model assigns positive coefficients to all features except fractal_dimension_mean, while the Decision Tree model does not directly provide information on the direction of influence.

1.1.3 Task 3:

Using PCA, present the scatter plot of the data on the first two principal components. Add to your scatter plot different colours to represent the two classes in the data. What proportion of data variance is explained using the first two principal components?

[12]:
```
pca = PCA(n_components=2, random_state=123)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Map categories to colors
color_map = {'M': 'red', 'B': 'blue'}
colors = y_train.map(color_map)

# Plotting the first two principal components
plt.figure(figsize=(8,6))
plt.scatter(X_train_pca[:,0], X_train_pca[:,1], c=colors)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')

# Create legend
legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='M',␣
 ↪markerfacecolor='red', markersize=10),
                   plt.Line2D([0], [0], marker='o', color='w', label='B',␣
 ↪markerfacecolor='blue', markersize=10)]
plt.legend(handles=legend_elements)

plt.show()
```



[13]:
```
# Print the explained variance
explained_variance = pca.explained_variance_ratio_
total_explained_variance = sum(explained_variance)

print(f"The first principal component explains {explained_variance[0]*100:.2f}%␣
  ↪of the variance.")
print(f"The second principal component explains {explained_variance[1]*100:.
  ↪2f}% of the variance.")
print(f"Together, the first two principal components explain␣
  ↪{total_explained_variance*100:.2f}% of the variance.")
```

```
The first principal component explains 49.50% of the variance.
The second principal component explains 25.59% of the variance.
Together, the first two principal components explain 75.09% of the variance.
```

The analysis revealed that the first principal component explained 49.50% of the variance, while the second principal component explained 25.59% of the variance. When considering these two principal components together, they accounted for a cumulative variance of 75.09%.

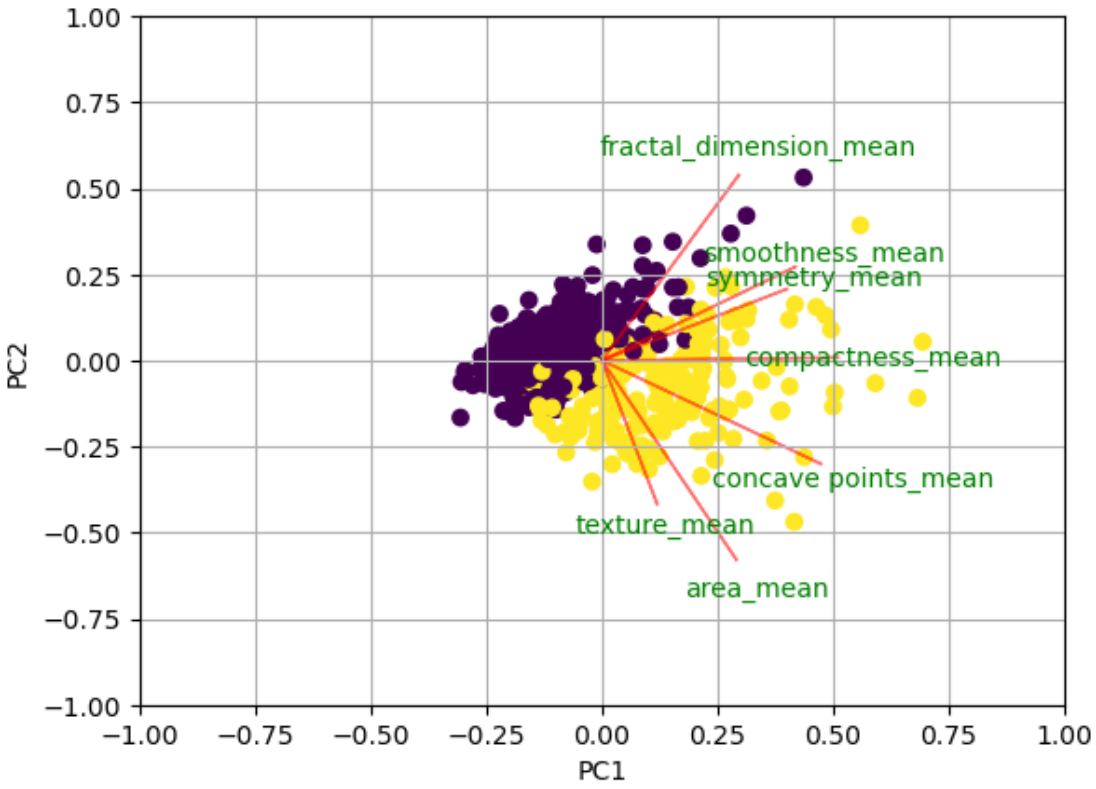
This indicates that a significant portion of the variability in the dataset can be captured by the first two principal components. The high cumulative variance suggests that these components retain a substantial amount of information and can potentially be used effectively for dimensionality

reduction and data visualisation.

1.1.4 Task 4:

Considering the first two principal components from Task 3, present the biplot with the variables vectors and the observed data projected on the first two principal components (with the colours for the two categories). Give your interpretation of the results.

[14]:
```
def biplot(score, coeff , y):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex, ys * scaley, c = y)
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color='r', alpha=0.5)
        plt.text(coeff[i,0]*1.15, coeff[i,1] *1.15, feature_names[i],␣
 ↪color='g', ha='center', va='center')
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train)

# Call the function
feature_names = ['texture_mean', 'area_mean', 'smoothness_mean',␣
 ↪'compactness_mean', 'concave points_mean', 'symmetry_mean',␣
 ↪'fractal_dimension_mean']
biplot(X_train_pca[:,0:2], np.transpose(pca.components_[0:2, :]),␣
 ↪y_train_encoded)
```



Conclusions 1. 'fractal_dimension_mean': This variable has the longest line pointing towards the top right quadrant. It suggests a strong positive correlation with PC1 and a positive correlation with PC2. 'fractal_dimension_mean' is an important variable for differentiating between the two categories. 2. 'area_mean': The line for 'area_mean' points towars the bottom right quandrant. It implies a positive correlation with PC1 and a negative correlation with PC2. 'area_mean' contributes significantly to the separation of the two categories. 3. 'compactness_mean': 'compactness_mean' has a long line pointing directly to the right. It suggests a positive correlation with PC1 and a weaker correlation with PC2. This variable plays a significant role in the direction of PC1 and has less influence on PC2. 4. 'smoothness_mean' and 'symmetry_mean': These two variables are positioned between the middle and the top right quadrant. They have similar moderate lengths, indicating that they have comparable positive correlation to PC1 and PC2. They have a moderate contribution in distinguishing between the two categories. 5. 'concave points_mean': 'concave points_mean' has a relatively long line, similar to 'area_mean' and 'compactness_mean.' It points between the middle and the bottom right quadrant, indicating a positive correlation with PC1 and a negative correlation with PC2. This variable also plays a significant role in distinguishing between the two categories. 6. 'texture_mean': 'texture_mean' has the shortest line, pointing almost straight down but slightly to the right. It suggests a weaker correlation with PC1 and a negative correlation with PC2. 'texture_mean' contributes less to the separation of the two categories compared to the other variables.

1.1.5 Task 5:

Using the plot of Task 4, which variables are more related to the tumour status? Justify your answer. Compare the results obtained with the results obtained in Task 2.

Variables with the most significant impact on tumor status:

1. 'fractal_dimension_mean': This variable shows a strong positive correlation with both PC1 and PC2, indicating its importance in differentiating between the two tumor status categories.
2. 'area_mean': It exhibits a positive correlation with PC1 and a negative correlation with PC2, contributing significantly to the separation of the tumor status categories.
3. 'concave points_mean': This variable demonstrates a positive correlation with PC1 and a negative correlation with PC2, playing a significant role in distinguishing between the two categories.

Variables with moderate impact on tumor status:

1. 'compactness_mean': It shows a positive correlation with PC1 and a weaker correlation with PC2, contributing to the direction of PC1 but having less influence on PC2.
2. 'smoothness_mean' and 'symmetry_mean': These variables exhibit moderate positive correlations with both PC1 and PC2 and contribute moderately to the differentiation between tumor status categories.

Variables with weaker impact on tumor status:

1. 'texture_mean': This variable demonstrates a weaker correlation with both PC1 and PC2 compared to the other variables. It contributes less to the separation of the tumor status categories.

Similarities:

- Both Task 2 and Task 4 identify 'area_mean' as an important variable related to tumor status. It has a strong positive impact according to the Logistic Regression model and contributes significantly to the separation of the two categories based on the biplot in Task 4.
- Both tasks indicate 'concave points_mean' as an influential variable for predicting tumor status. It has a significant positive influence in Task 2 and plays a crucial role in distinguishing between the categories according to the biplot in Task 4.
- Task 2 and Task 4 both identify 'compactness_mean', 'smoothness_mean', and 'symmetry_mean' as variables with moderate influence over tumour status.

Differences:

- Task 2's logistic model ranks 'texture_mean' as the third most important variable and the decision tree model ranks it second, however it is positioned lower in the biplot from Task 4, indicating a weaker correlation with tumor status.
- Task 4 identifies 'fractal_dimension_mean' as one of the most important variables, and yet, both models in task 2 rank this as a variable with little importance.

These differences can be attributed to variations in the modeling techniques, algorithms, and interpretations of feature importance between Logistic Regression and PCA with biplot analysis. While Task 2 focuses on coefficients and importance rankings derived from the Logistic Regression model,

Task 4 leverages PCA and biplot visualisation to analyse the relationships between variables and principal components.

1.1.6 Task 6:

Using PCA, determine the number of components to retain 95% of the explained variance. Use as new features the resulting principal components scores and repeat task 1 on these new features. You can choose one of the models (logistic regression or the decision tree). What is the dimension of the new (projected) data set? Comment on the performance resulting from using the original and principal components features.

[15]:
```
pca = PCA()
pca.fit(X_train_scaled)

cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
component_number = np.argmax(cumulative_variance_ratio >= 0.95) + 1 # Adding 1␣
 ↪as numpy array index starts from 0

print("Number of components that explain at least 95% of the variance: ",␣
 ↪component_number)
```

```
Number of components that explain at least 95% of the variance:  5
```

[16]:
```
# Perform PCA
pca = PCA(n_components=component_number)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Create a logistic regression model
log_reg = LogisticRegression(C=0.1, max_iter=10000, penalty='l2',␣
 ↪solver='liblinear')

# Train the model
log_reg.fit(X_train_pca, y_train)

# Predict the test set results
y_pred_pca = log_reg.predict(X_test_pca)

# Calculate accuracy
accuracy_pca = accuracy_score(y_test, y_pred_pca)
print("Accuracy with PCA features: ", accuracy_pca)


cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)
print("Variance explained by 5 variables ", cumulative_variance_ratio[4])
```

```
Accuracy with PCA features:  0.9649122807017544
Variance explained by 5 variables  0.9816364286704534
```

[17]:
```
# Calculate the dimension of the original data and new data
original_dim = X_train.shape[1]
new_dim = X_train_pca.shape[1]

print(f"Original data dimension: {original_dim}, New data dimension: {new_dim}")
```

```
Original data dimension: 7, New data dimension: 5
```

Using a Logistic Regression Model, retaining 95% of the explained variance can be done using 5 of the 7 variables. Explained variance using these 5 variables is 98.2% which is quite significant.

When evaluating the accuracy of the model using these 5 variables, it is observed that the accuracy remains exactly the same (0.965) as compared to the accuracy achieved with the original set of variables. This consistency in accuracy can be attributed to the fact that the retained variance is still quite high, indicating that the variables that were removed did not contribute significantly to the model's predictive performance.

2 Part 2: A clustering analysis on airlines safety records (50 marks)

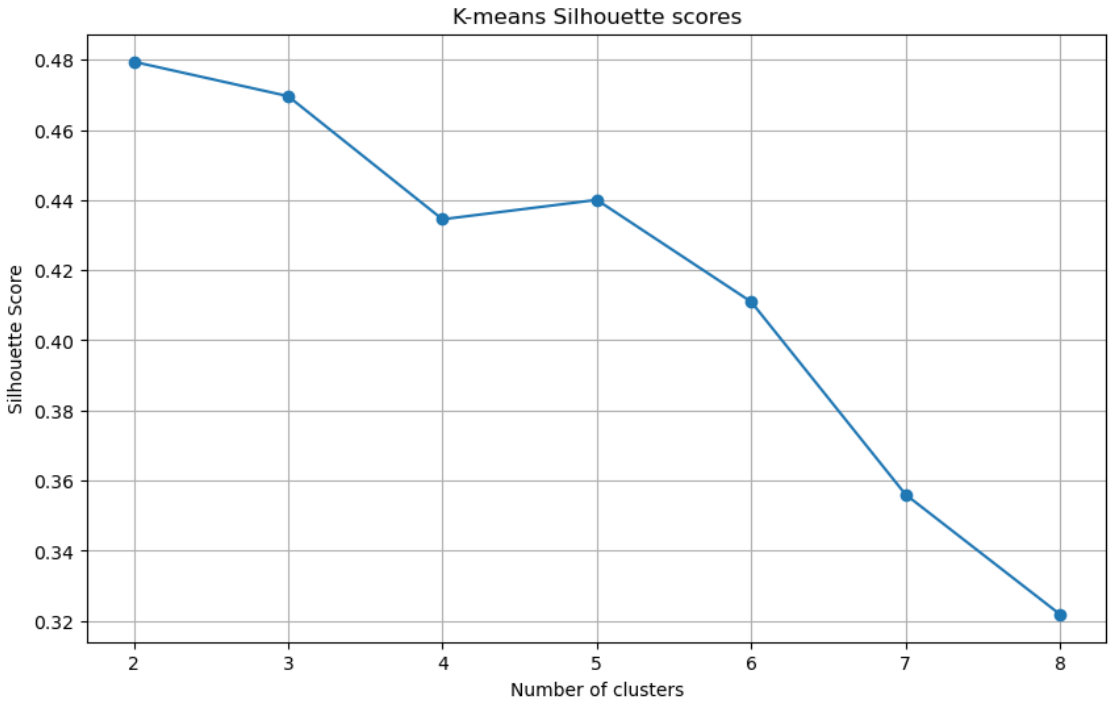
[18]:
```
data = pd.read_csv('airline-safety.csv')
```

2.0.1 Task 1

Considering the K-means clustering, plot the silhouette score for values of K varying from 2 to 8. Discuss the results and comment on what would be a good choice(s) for K. For the K-means clustering, you should use the Euclidean distance and set random state to "5508".

[19]:
```
# Select the relevant columns
data_selected = data.iloc[:, 2:]  # Exclude airline names

# Standardise the features to have mean=0 and variance=1
scaler = StandardScaler().fit(data_selected)
data_standardised = scaler.transform(data_selected)
silhouette_scores = []

# Run K-means for K from 2 to 8
for i in range(2, 9):
    kmeans = KMeans(n_clusters=i, n_init=10, random_state=5508)
    kmeans.fit(data_standardised)
    score = silhouette_score(data_standardised, kmeans.labels_)
    silhouette_scores.append(score)

# Plot silhouette scores
plt.figure(figsize=(10,6))
plt.plot(range(2, 9), silhouette_scores, marker='o')
plt.title('K-means Silhouette scores')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()

print("2 clusters has a silhouette score of ", silhouette_scores[0])
```




```
2 clusters has a silhouette score of  0.4794517912991833
```

2 Clusters has the highest Silhouette Score, 0.479, meaning it exhibits a relatively better clustering structure compared to other values of K.

As the number of clusters increases from 2 to 8, the Silhouette Scores generally decreases. This suggests that the clusters become less distinct and more overlapping as the number of clusters increases. While higher values of K may capture more detailed patterns in the data, they result in smaller and less interpretable clusters.

2.0.2 Task 2:

Apply K-means clustering with the value of K obtained in Task 1. Describe the main characteristic of each group, that is, provide the interpretation of the groups in terms of safety records. For the K-means clustering, you should use the Euclidean distance and set random state to "5508".

[20]:
```
optimal_clusters = 2
data_standardised_df = pd.DataFrame(data_standardised, columns=data_selected.
↪columns, index=data_selected.index)
```

```python
# Apply K-means clustering
kmeans = KMeans(n_clusters=optimal_clusters, n_init=10, random_state=5508).
 ↪fit(data_standardised)
data_standardised_df['cluster'] = kmeans.labels_

# Descriptive statistics for each cluster
print("Descriptive Statistics for Each Cluster:")
for i in range(optimal_clusters):
    print(f"\nCluster {i}:")
    print(data_standardised_df[data_standardised_df['cluster'] == i].describe())
```

```
Descriptive Statistics for Each Cluster:

Cluster 0:
       incidents_85_99  fatal_accidents_85_99  fatalities_85_99  \
count        43.000000              43.000000         43.000000
mean         -0.320404              -0.374652         -0.190151
std           0.255304               0.456671          0.937664
min          -0.656376              -0.768345         -0.773244
25%          -0.473505              -0.768345         -0.773244
50%          -0.382069              -0.415662         -0.745729
75%          -0.153481              -0.239321          0.017811
max           0.440849               0.995070          2.803701

       incidents_00_14  fatal_accidents_00_14  fatalities_00_14  cluster
count        43.000000              43.000000         43.000000     43.0
mean         -0.327213              -0.448475         -0.315377      0.0
std           0.558479               0.591771          0.511993      0.0
min          -0.915809              -0.776414         -0.503179      0.0
25%          -0.693795              -0.776414         -0.503179      0.0
50%          -0.471780              -0.776414         -0.503179      0.0
75%          -0.027752              -0.188857         -0.498647      0.0
max           1.526348               1.573811          2.061755      0.0

Cluster 1:
       incidents_85_99  fatal_accidents_85_99  fatalities_85_99  \
count        13.000000              13.000000         13.000000
mean          1.059797               1.239235          0.628961
std           1.689352               1.330450          1.015528
min          -0.382069              -0.415662         -0.539367
25%           0.075108               0.289704         -0.099128
50%           0.623720               0.995070          0.375505
75%           1.263768               1.700435          1.015228
max           6.292717               4.169215          2.906882

       incidents_00_14  fatal_accidents_00_14  fatalities_00_14  cluster
count        13.000000              13.000000         13.000000     13.0
mean          1.082320               1.483418          1.043172      1.0
```

std	1.386479	0.579975	1.487914	0.0
min	-0.471780	0.398699	-0.303785	1.0
25%	0.194263	1.573811	-0.040947	1.0
50%	0.638291	1.573811	0.330651	1.0
75%	1.526348	1.573811	1.536079	1.0
max	4.412534	2.748924	4.363851	1.0

Cluster 0:

- This cluster consists of 43 airlines.
- In this cluster, the variables related to incidents, fatal accidents, and fatalities for both the 1985-1999 and 2000-2014 periods have relatively lower values compared to Cluster 1.
- The mean values for incidents, fatal accidents, and fatalities in this cluster are negative, indicating lower values than the overall dataset mean.
- The standard deviations in this cluster are relatively smaller compared to Cluster 1, suggesting less variability in the safety records.
- Airlines in Cluster 0 have fewer incidents, fatal accidents, and fatalities over the given time periods compared to Cluster 1.

Cluster 1:

- This cluster consists of 13 airlines.
- In this cluster, the variables related to incidents, fatal accidents, and fatalities for both the 1985-1999 and 2000-2014 periods have relatively higher values compared to Cluster 0.
- The mean values for incidents, fatal accidents, and fatalities in this cluster are positive, indicating higher values than the overall dataset mean.
- The standard deviations in this cluster are relatively larger compared to Cluster 0, suggesting more variability in the safety records.
- Airlines in Cluster 1 have a higher number of incidents, fatal accidents, and fatalities over the given time periods compared to Cluster 0.
- Based on these observations, we can conclude that Cluster 0 represents airlines with relatively better safety records, characterised by lower incident rates, fatal accidents, and fatalities. On the other hand, Cluster 1 represents airlines with relatively poorer safety records, characterised by higher incident rates, fatal accidents, and fatalities.

2.0.3 Task 3:

Explain your decision about scaling or not the data before running K-means (on Tasks 1 and 2), and explain your decision about using or not all variables in the analysis.

Reason why the data was scaled

Variables such as "fatalities_85_99" and "fatalities_00_14" measure the counts of fatalities, representing the number of people who died in plane incidents. These variables are on a different scale compared to variables like "fatal_accidents_85_99," "fatal_accidents_00_14," "incidents_85_99," and "incidents_00_14," which measure the counts of fatal accidents and overall incidents. Since fatalities involve the number of people and can have significantly larger values, they have the potential to skew the clustering analysis.

By scaling the data, all variables are on a similar scale, making them comparable and ensuring that no single variable dominates the clustering process solely due to its scale or magnitude. This helps

in achieving a more balanced and accurate clustering analysis.

Reasons why "airline" and "avail seat km per week" were not included:

"Airline":

The focus of the analysis is solely on safety records, and the airline variable might not directly contribute to understanding safety performance. Including this categorical variable could introduce noise or unnecessary complexity to the clustering analysis if the objective is solely to identify patterns in safety records.

"Avail seat km per week":

The variable measures the capacity or size of the airline's operations and might not have a direct relationship with safety records. Including this variable could lead to misleading interpretations or associations between operational scale and safety performance if such a relationship does not exist.

Conclusion: Based on the objectives of the task, investigating safety records, "airline" and "avail seat km per week" variables were excluded. The focus of the analysis is on safety records, and including these variables might introduce unnecessary complexity or irrelevant information. By excluding them, the analysis can more directly focus on patterns and characteristics related to safety incidents, accidents, and fatalities.

Task 4: Perform a K-means cluster analysis, considering the value of K from Task 1, and: (a) the three variables from the years 1985-1999; (b) the three variables from the years 2000-2014. Did the clusters change? Explain the results. For the K-means clustering, you should use the Euclidean distance and set random state to "5508".

[21]:
```
# Split the standardised data based on the years
data_85_99_standardised_df = data_standardised_df[['incidents_85_99',␣
 ↪'fatal_accidents_85_99', 'fatalities_85_99']]
data_00_14_standardised_df = data_standardised_df[['incidents_00_14',␣
 ↪'fatal_accidents_00_14', 'fatalities_00_14']]

# Apply K-means clustering
kmeans_85_99 = KMeans(n_clusters=optimal_clusters, n_init=10,␣
 ↪random_state=5508).fit(data_85_99_standardised_df)
kmeans_00_14 = KMeans(n_clusters=optimal_clusters, n_init=10,␣
 ↪random_state=5508).fit(data_00_14_standardised_df)

data_standardised_df['cluster_85_99'] = kmeans_85_99.labels_
data_standardised_df['cluster_00_14'] = kmeans_00_14.labels_

# Descriptive statistics for each cluster
print("Group 85/99")
for i in range(optimal_clusters):
    print(f"\nCluster {i}:")
    print(data_standardised_df[data_standardised_df['cluster_85_99'] ==␣
 ↪i][['incidents_85_99', 'fatal_accidents_85_99', 'fatalities_85_99']].
 ↪describe())
```

```
print("\nGroup 00/14")
for i in range(optimal_clusters):
    print(f"\nCluster {i}:")
    print(data_standardised_df[data_standardised_df['cluster_00_14'] ==␣
 ↪i][['incidents_00_14', 'fatal_accidents_00_14', 'fatalities_00_14']].
 ↪describe())
```

```
Group 85/99

Cluster 0:
       incidents_85_99  fatal_accidents_85_99  fatalities_85_99
count        41.000000              41.000000         41.000000
mean         -0.326316              -0.415662         -0.479807
std           0.271147               0.409780          0.496235
min          -0.656376              -0.768345         -0.773244
25%          -0.473505              -0.768345         -0.773244
50%          -0.382069              -0.415662         -0.752608
75%          -0.199199              -0.415662         -0.333005
max           0.623720               0.642387          1.489861

Cluster 1:
       incidents_85_99  fatal_accidents_85_99  fatalities_85_99
count        15.000000              15.000000         15.000000
mean          0.891931               1.136143          1.311473
std           1.620412               1.277193          0.882846
min          -0.382069              -0.415662         -0.078491
25%           0.075108               0.289704          0.801987
50%           0.440849               0.995070          1.345408
75%           1.172333               1.524094          1.737496
max           6.292717               4.169215          2.906882

Group 00/14

Cluster 0:
       incidents_00_14  fatal_accidents_00_14  fatalities_00_14
count        18.000000              18.000000         18.000000
mean          0.773967               1.247391          1.007383
std           1.283621               0.675118          1.295225
min          -0.471780               0.398699         -0.303785
25%          -0.027752               0.398699          0.267208
50%           0.416277               1.573811          0.489260
75%           1.193327               1.573811          1.452243
max           4.412534               2.748924          4.363851

Cluster 1:
       incidents_00_14  fatal_accidents_00_14  fatalities_00_14
count        38.000000              38.000000         38.000000
```

```
mean        -0.366616          -0.590870          -0.477181
std          0.571851           0.434248           0.123161
min         -0.915809          -0.776414          -0.503179
25%         -0.693795          -0.776414          -0.503179
50%         -0.471780          -0.776414          -0.503179
75%         -0.027752          -0.776414          -0.503179
max          1.526348           0.398699           0.249081
```

Comparing the two clusters within each group:

For the group of data from 1985-1999:

Cluster 0 has a lower mean for incidents, fatal accidents, and fatalities compared to Cluster 1. This suggests that airlines in Cluster 0 had lower incident rates, fatal accidents, and fatalities during this time period.

For the group of data from 2000-2014:

Cluster 0 has a higher mean for incidents, fatal accidents, and fatalities compared to Cluster 1. This implies that airlines in Cluster 0 had higher incident rates, fatal accidents, and fatalities during this time frame.

Comparing the groups:

Overall, comparing the two groups, airlines in Cluster 0 of the 1985-1999 dataset had lower incident rates, fatal accidents, and fatalities compared to Cluster 1, while in the 2000-2014 dataset, airlines in Cluster 1 had lower incident rates, fatal accidents, and fatalities compared to Cluster 0. These findings indicate a shift in the safety performance of airlines between the two time periods, with different clusters representing different safety profiles.

Task 5: Consider three new features as the ratio of the variables from 2000-2014 divided by the respective variables from 1985-1999. Now, perform a K-means cluster analysis, considering the value of K from Task 1. Present the results of this cluster analysis, and compare them with the results from Task 2 and Task 4. For the K-means clustering, you should use the Euclidean distance and set random state to "5508".

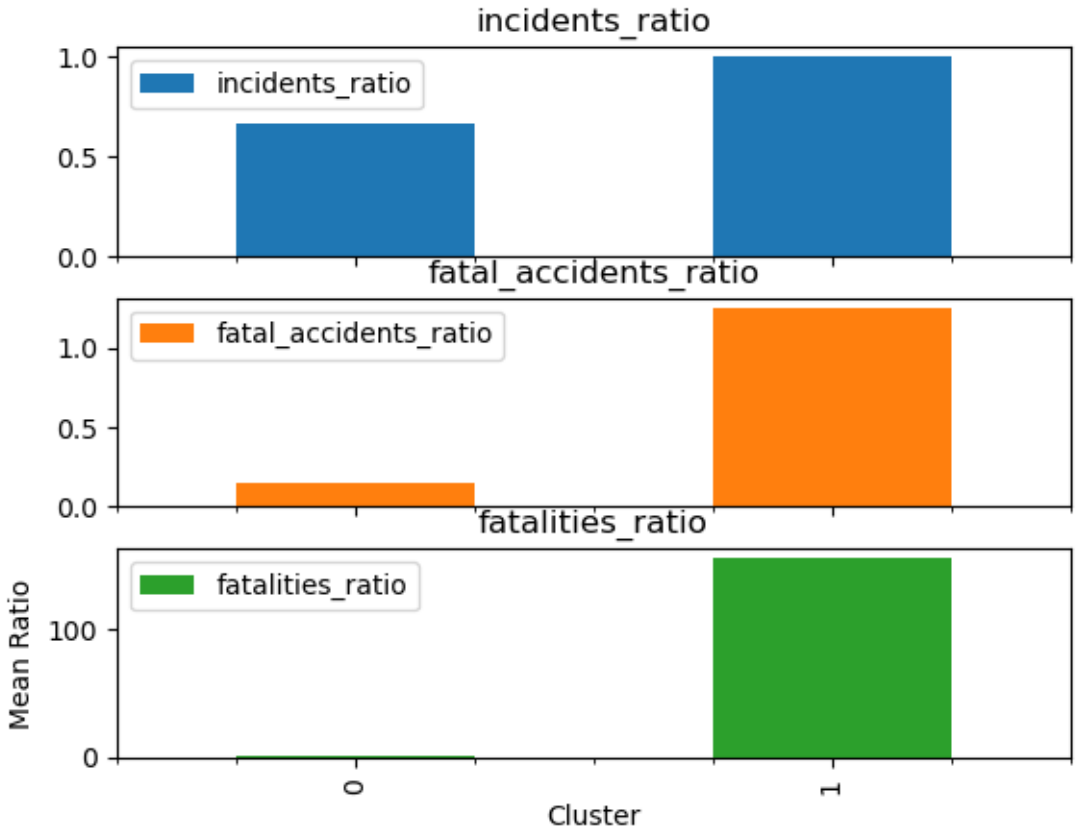
[22]:
```
epsilon = 1  # to handle null values
data_selected['incidents_ratio'] = data_selected['incidents_00_14'] /␣
 ↪(data_selected['incidents_85_99'] + epsilon)
data_selected['fatal_accidents_ratio'] = data_selected['fatal_accidents_00_14']␣
 ↪/ (data_selected['fatal_accidents_85_99'] + epsilon)
data_selected['fatalities_ratio'] = data_selected['fatalities_00_14'] /␣
 ↪(data_selected['fatalities_85_99'] + epsilon)

# Define ratios data
ratios = data_selected[['incidents_ratio', 'fatal_accidents_ratio',␣
 ↪'fatalities_ratio']]

# Apply K-means clustering
kmeansRatios = KMeans(n_clusters=2, random_state=5508, n_init=10)
labelsRatios = kmeansRatios.fit_predict(ratios)
```

```
centroidsRatios = kmeansRatios.cluster_centers_

# Add the labels to the data frame
data_selected['cluster_ratio'] = labelsRatios

#print(data_selected.loc[labelsRatios == 0].describe())

#print(data_selected.loc[labelsRatios == 1].describe())

print("\nRatio Cluster Descriptive Statistics:")
for i in range(2):  # Since n_clusters=2
    print(f"\nCluster {i}:")
    print(data_selected[data_selected['cluster_ratio'] ==␣
↪i][['incidents_ratio', 'fatal_accidents_ratio', 'fatalities_ratio']].
↪describe())
```

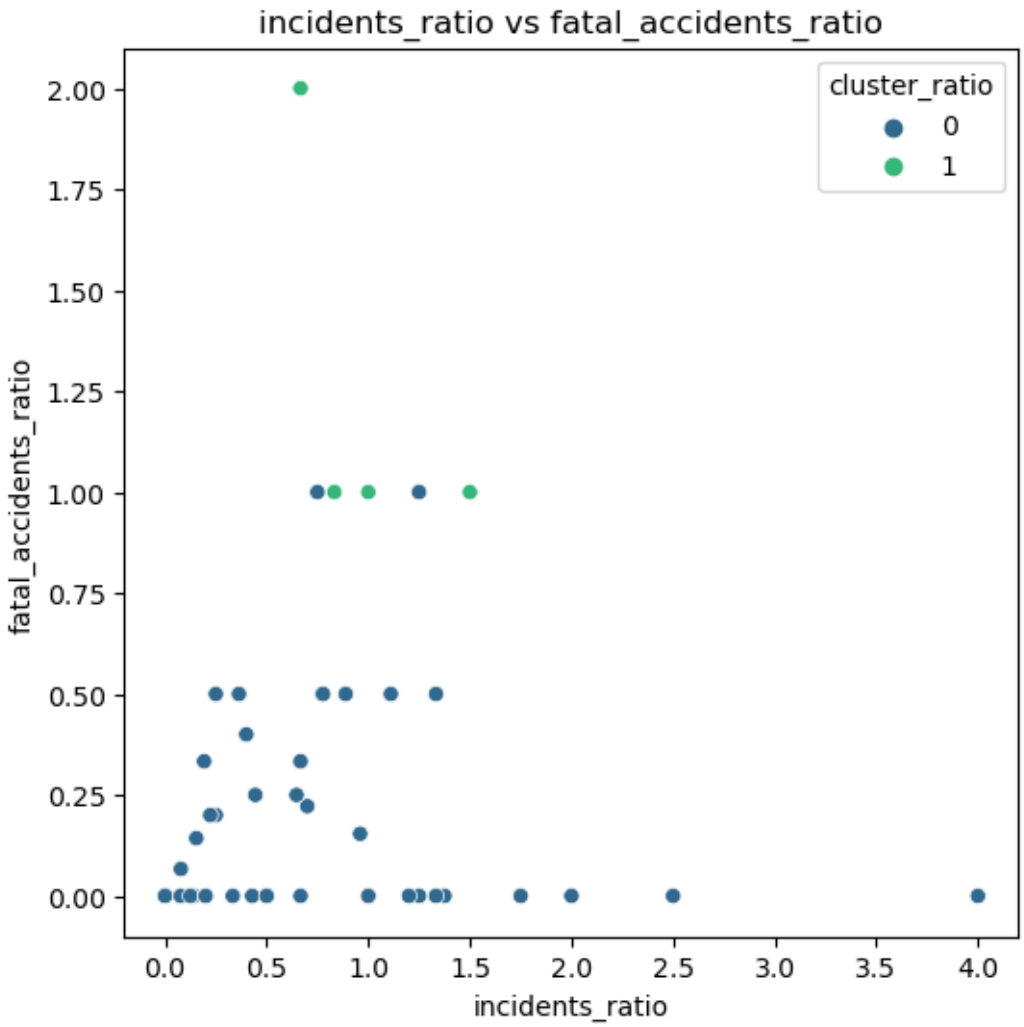
```
Ratio Cluster Descriptive Statistics:

Cluster 0:
       incidents_ratio  fatal_accidents_ratio  fatalities_ratio
count        52.000000              52.000000         52.000000
mean          0.667666               0.154851          0.951433
std           0.737974               0.250028          2.868018
min           0.000000               0.000000          0.000000
25%           0.138393               0.000000          0.000000
50%           0.500000               0.000000          0.000000
75%           1.000000               0.250000          0.231965
max           4.000000               1.000000         15.342857

Cluster 1:
       incidents_ratio  fatal_accidents_ratio  fatalities_ratio
count         4.000000                   4.00          4.000000
mean          1.000000                   1.25        156.000000
std           0.360041                   0.50         87.631805
min           0.666667                   1.00         88.000000
25%           0.791667                   1.00        104.500000
50%           0.916667                   1.00        126.500000
75%           1.125000                   1.25        178.000000
max           1.500000                   2.00        283.000000
```

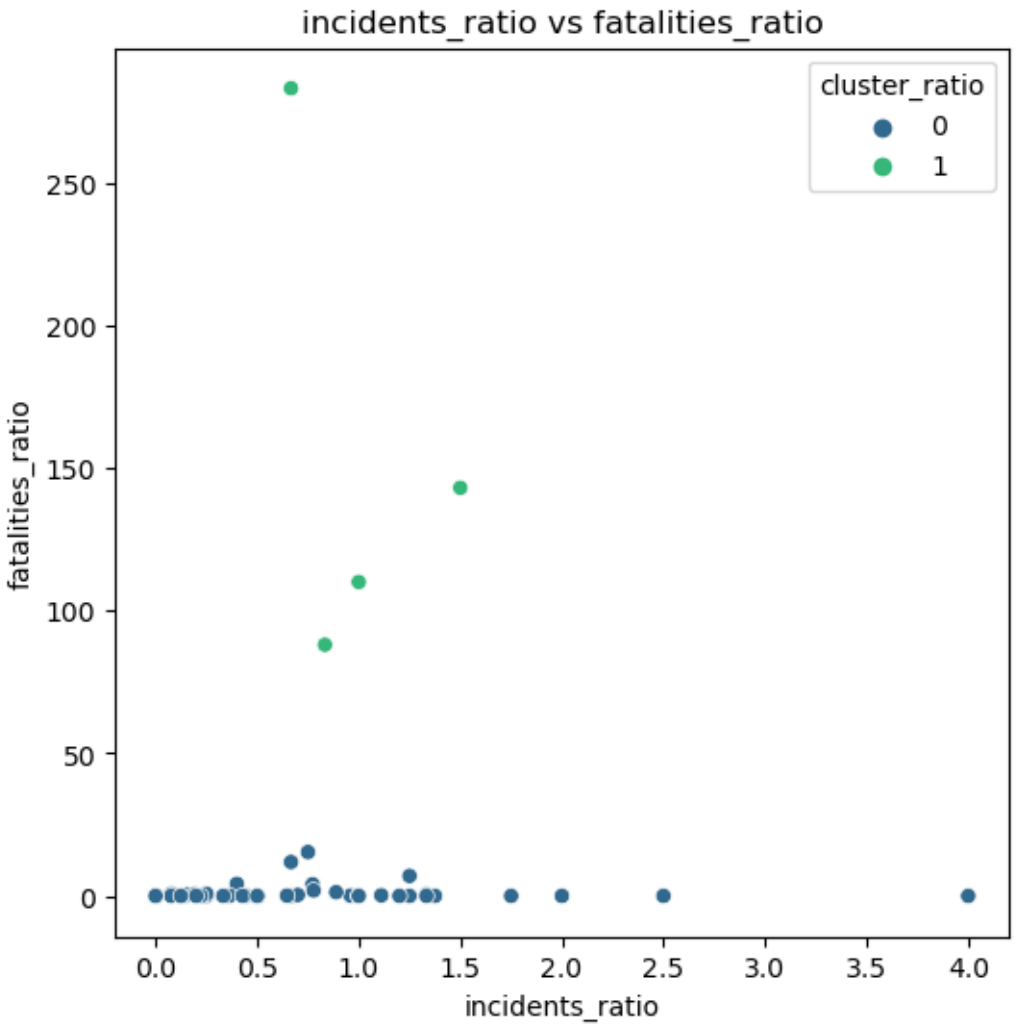
[23]:
```
import matplotlib.pyplot as plt

# Calculate mean values for each ratio by cluster
mean_ratios = data_selected.groupby('cluster_ratio')[['incidents_ratio',␣
↪'fatal_accidents_ratio', 'fatalities_ratio']].mean()
```

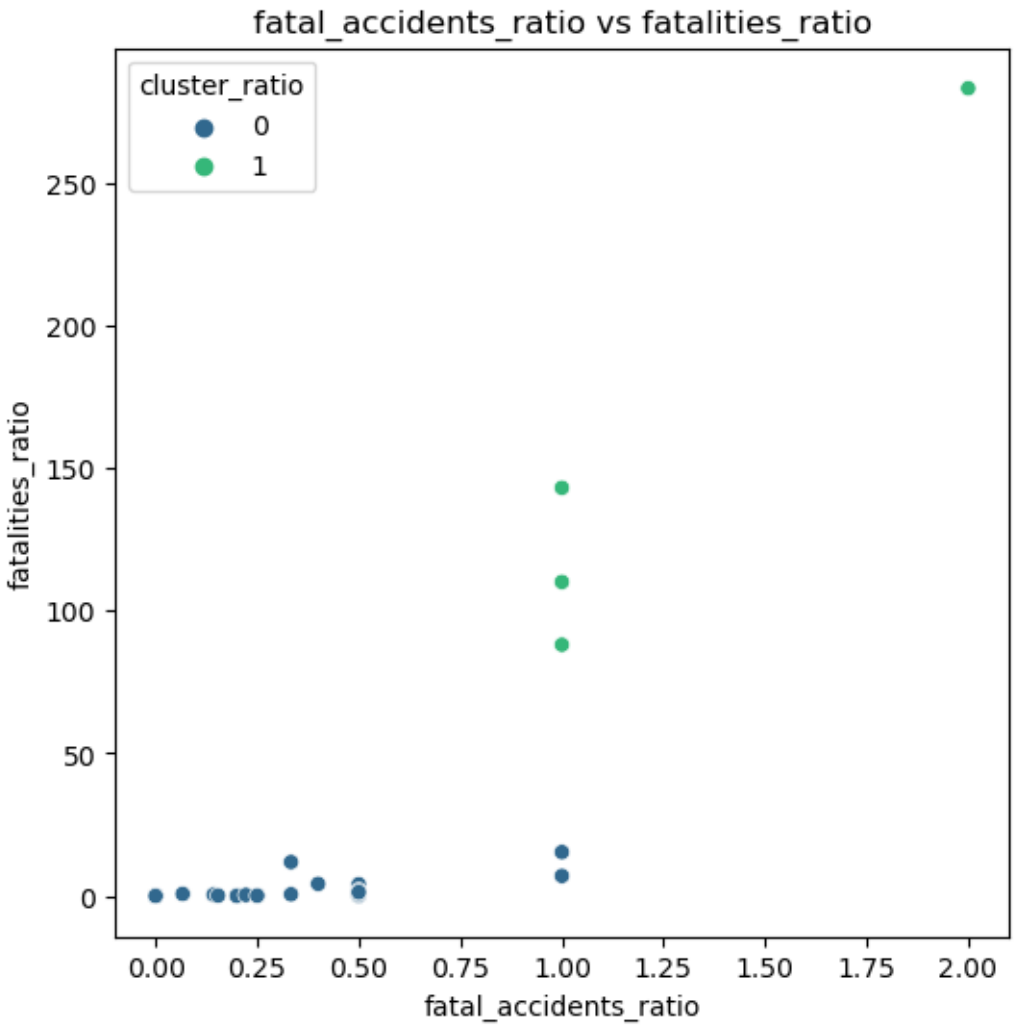
```
# Plot the bar charts
mean_ratios.plot(kind='bar', subplots=True, layout=(3,1), sharex=True)
plt.xlabel('Cluster')
plt.ylabel('Mean Ratio')
plt.show()
```



[24]:
```
# Create scatterplots for each pair of features
features = ['incidents_ratio', 'fatal_accidents_ratio', 'fatalities_ratio']
for i in range(len(features)):
    for j in range(i + 1, len(features)):
        plt.figure(figsize=(6, 6))
        sns.scatterplot(x=features[i], y=features[j], hue='cluster_ratio',␣
↪palette='viridis', data=data_selected)
        plt.title(f'{features[i]} vs {features[j]}')
        plt.show()
```


incidents_ratio vs fatal_accidents_ratio
cluster_ratio
0
1
fatal_accidents_ratio
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
incidents_ratio


incidents_ratio vs fatalities_ratio
cluster_ratio
0
1
fatalities_ratio
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
incidents_ratio



An epsilon value of 1 was employed, as lower values often led to disproportionately skewed ratios.

Based on the results of the Kmeans analysis using the ratios the data has been split into two clusters of sizes, 52 and 4.

Cluster 0 was of size 52 and this is the largest cluster seen when comparing to previous sections, part 2 and part 4 with their largest cluster sizes obtained being of size 41 in the 85/99 group for part 4 and size 43 for part 2.

Analysing the scatter plots in the last two graphs there appears to be a clear distinction between the two groups however the first graph appears to have some overlap between fatal_accidents ratio and incidents ratio. Since the original silhouette scores were applied to the scaled data and for the ratios the unscaled data is used there is potentially the possibility that theire might be a more appropriate k value for the unscaled data which would lead to a clearer distinction between the two groups in the first scatter plot. Even if it doesn't the fact that there are disticntions between the two groups with the other two scatter plots, fatalities ratio against fatal accidents ratio and fatalities ratio against incidents ratio, still gives a clear indication of the seperation of the two

groups.

The bar chart compares means of the ratios and overall all three ratios are smaller for group 0 compared to group 1. Since group 0 contains the majority of the results, overall the accident ratios have decreased between 85/99 to 00/14 implying that airline travel has gotten safer.

Group 1's remaining four data points could potentially be outliers due to their significant deviation from the primary dataset, symbolising uncommon incidents associated with high casualties. The ratios in group 1 stand as follows: both ratios for incidents are 1, for fatal accidents it exceeds 1 (with a value of 1.25), and the fatality ratio is drastically higher, pegged at 156. Based on the results of group 1 airline safety has gotten worse in 00/14 compared to 85/99 and the most shocking finding was the drastic ratio for fatalities being 156:1 meaning you are 156 times more likely to die as a result of a plane crash in 00/14 compared to 85/99. Once again the unexpected results in group 1 are believed to be the result of anomalies in the data, a few incidents (4) have skewed this group to have extreme findings that contradict the main findings. Because of this the overall conclusions about airline safety should be drawn from group 0.

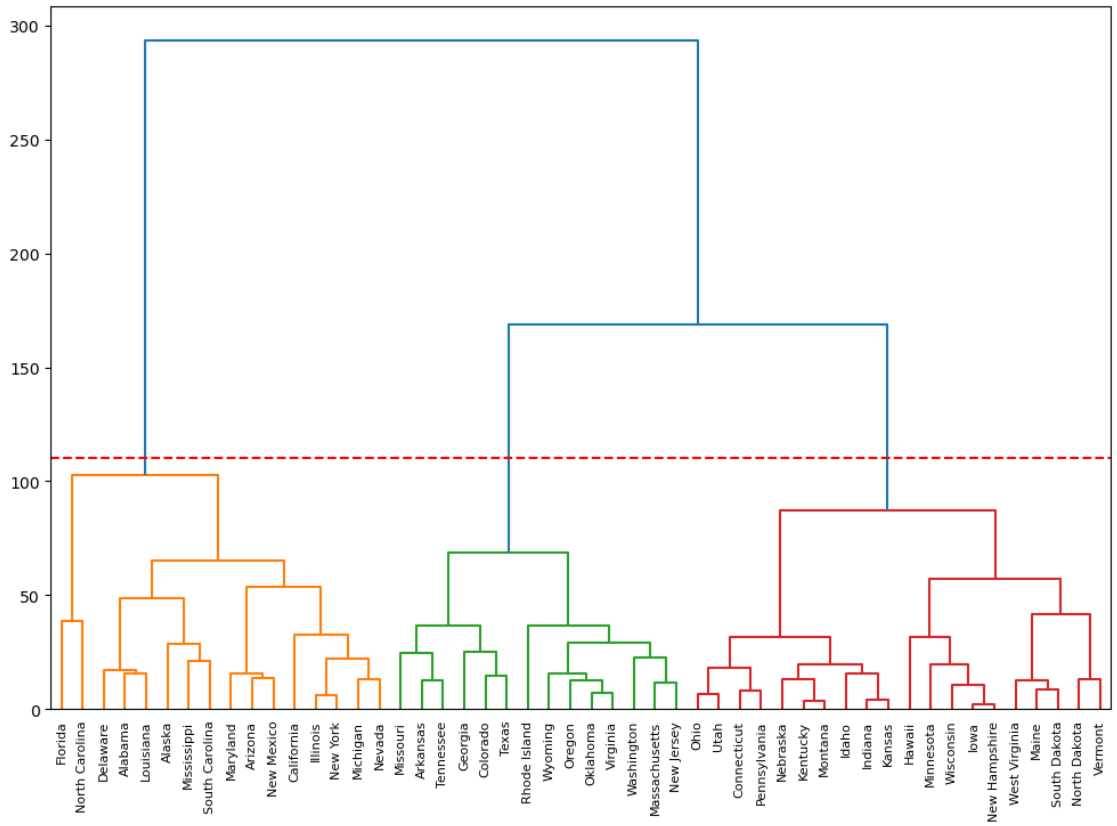
2.1 Part 3 A clustering analysis on the USArrests data (40 marks)

[25]:
```
# Load data
data = pd.read_csv('USArrests.csv')
states = data['State']
data = data.select_dtypes(include=[np.number])
```

2.1.1 Task 1:

Using the raw data, perform a hierarchical clustering with complete linkage and Euclidean distance to cluster the states. Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which cluster? Describe their characteristics.

[26]:
```
# Perform hierarchical clustering
Z = linkage(data, 'complete')

# Plot the dendrogram
plt.figure(figsize=(12, 8))
dendrogram(Z, labels=states.values, leaf_rotation=90, color_threshold=110)
plt.axhline(y=110, color='r', linestyle='--')
plt.show()
```



[27]:
```
# Cut-off value
max_d = 110

# fcluster function
clusters = fcluster(Z, max_d, criterion='distance')

# Map each state to its cluster
state_cluster = pd.DataFrame({'State': states, 'Cluster': clusters})

# Function to format the cluster output
def format_cluster_output(cluster_num, states):
    return f"Cluster {cluster_num}:\n" + "\n".join(f" - {state}" for state in␣
 ↪states) + "\n"

# Print the states in each cluster
for cluster_num in range(1, state_cluster['Cluster'].nunique() + 1):
    states_in_cluster = state_cluster[state_cluster['Cluster'] ==␣
 ↪cluster_num]['State'].to_list()
    print(format_cluster_output(cluster_num, states_in_cluster))
```

```
Cluster 1:
```

```
 - Alabama
 - Alaska
 - Arizona
 - California
 - Delaware
 - Florida
 - Illinois
 - Louisiana
 - Maryland
 - Michigan
 - Mississippi
 - Nevada
 - New Mexico
 - New York
 - North Carolina
 - South Carolina

Cluster 2:
 - Arkansas
 - Colorado
 - Georgia
 - Massachusetts
 - Missouri
 - New Jersey
 - Oklahoma
 - Oregon
 - Rhode Island
 - Tennessee
 - Texas
 - Virginia
 - Washington
 - Wyoming

Cluster 3:
 - Connecticut
 - Hawaii
 - Idaho
 - Indiana
 - Iowa
 - Kansas
 - Kentucky
 - Maine
 - Minnesota
 - Montana
 - Nebraska
 - New Hampshire
 - North Dakota
 - Ohio
```

```
- Pennsylvania
- South Dakota
- Utah
- Vermont
- West Virginia
- Wisconsin
```

[28]:
```
# Add the cluster assignments to the original dataframe
data['Cluster'] = clusters
# Print out the means of each variable per cluster
cluster_characteristics = data.groupby('Cluster').mean(numeric_only=True)
print(cluster_characteristics)
```

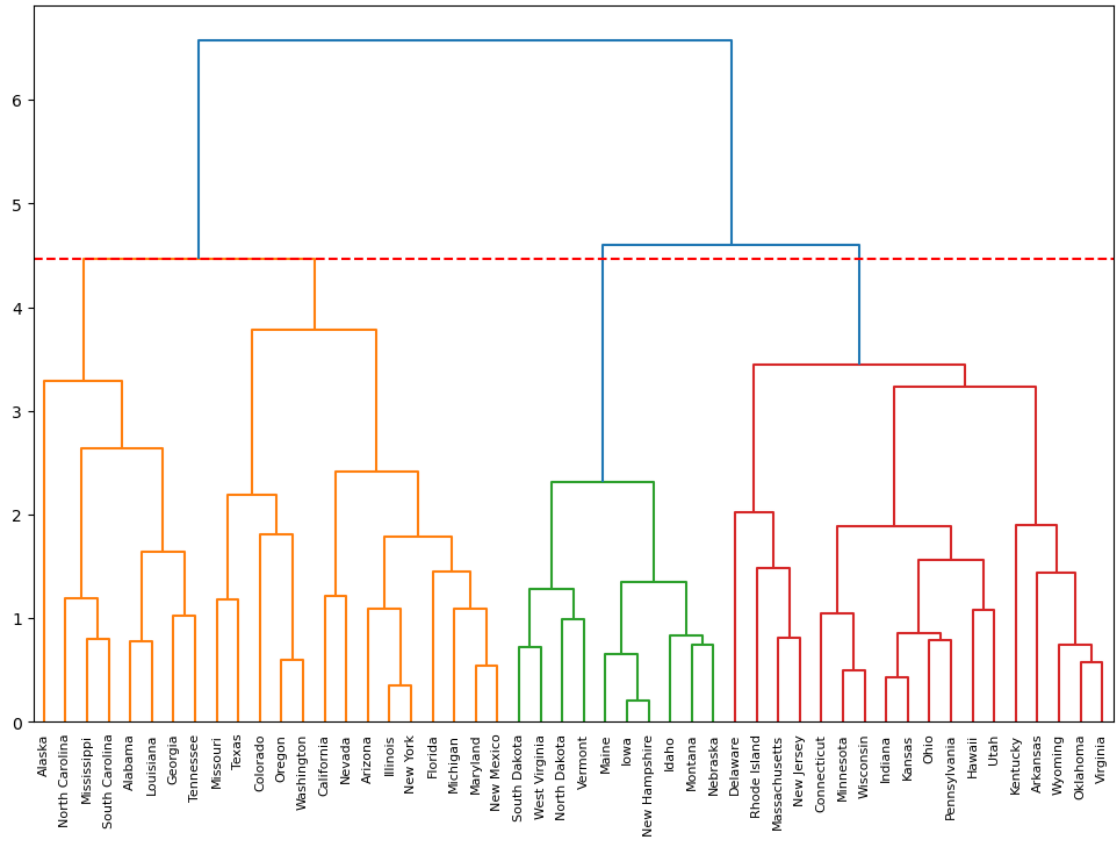
```
             Murder     Assault   UrbanPop       Rape
Cluster
1         11.812500  272.562500  68.312500  28.375000
2          8.214286  173.285714  70.642857  22.842857
3          4.270000   87.550000  59.750000  14.390000
```

Cluster 1 contains states that, on average, have the highest rates of Murder, Assault, and Rape, along with relatively high UrbanPop percentages. This suggests that these states might be more urbanised and, unfortunately, experience higher crime rates. **Cluster 2** states exhibit moderate average rates of Murder, Assault, and Rape, and their UrbanPop percentage is slightly higher than that of Cluster 1. This might indicate that these states are somewhat balanced between urban and rural areas and exhibit a moderate crime rate. **Cluster 3** states have the lowest average Murder, Assault, and Rape rates of the three clusters. These states also have the lowest average UrbanPop percentages, suggesting that they are less urbanised. This cluster might be characterised by a more rural setting with lower crime rates.

2.1.2 Task 2:

Repeat Task 1 after scaling the variables to have zero mean and unit standard deviation. What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled? Justify for your answer.

[29]:
```
# Scale data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
# Perform hierarchical clustering
Z_scaled = linkage(scaled_data, 'complete', metric='euclidean')
# Plot the dendrogram
plt.figure(figsize=(12, 8))
dendrogram(Z_scaled, labels=states.values, leaf_rotation=90, color_threshold=4.
 ↪47)
plt.axhline(y=4.47, color='r', linestyle='--')
plt.show()
```



[30]:
```
max_d_scaled = 4.47

# Apply the fcluster function
clusters_scaled = fcluster(Z_scaled, max_d_scaled, criterion='distance')

# Map each state to its scaled cluster
state_cluster_scaled = pd.DataFrame({'State': states, 'Cluster':␣
 ↪clusters_scaled})

# Print the states in each scaled cluster
for cluster_num in range(1, state_cluster_scaled['Cluster'].nunique() + 1):
    states_in_cluster_scaled =␣
 ↪state_cluster_scaled[state_cluster_scaled['Cluster'] ==␣
 ↪cluster_num]['State'].to_list()
    print(format_cluster_output(cluster_num, states_in_cluster_scaled))

# Add the scaled cluster assignments to the original dataframe
data['Cluster_Scaled'] = clusters_scaled

# Print out the means of each variable per scaled cluster
```

```
cluster_characteristics_scaled = data.groupby('Cluster_Scaled').
  ↪mean(numeric_only=True)
print(cluster_characteristics_scaled)
```

```
Cluster 1:
 - Alabama
 - Alaska
 - Arizona
 - California
 - Colorado
 - Florida
 - Georgia
 - Illinois
 - Louisiana
 - Maryland
 - Michigan
 - Mississippi
 - Missouri
 - Nevada
 - New Mexico
 - New York
 - North Carolina
 - Oregon
 - South Carolina
 - Tennessee
 - Texas
 - Washington

Cluster 2:
 - Idaho
 - Iowa
 - Maine
 - Montana
 - Nebraska
 - New Hampshire
 - North Dakota
 - South Dakota
 - Vermont
 - West Virginia

Cluster 3:
 - Arkansas
 - Connecticut
 - Delaware
 - Hawaii
 - Indiana
 - Kansas
 - Kentucky
```

```
 - Massachusetts
 - Minnesota
 - New Jersey
 - Ohio
 - Oklahoma
 - Pennsylvania
 - Rhode Island
 - Utah
 - Virginia
 - Wisconsin
 - Wyoming
```

	Murder	Assault	UrbanPop	Rape	Cluster
Cluster_Scaled					
1	11.463636	245.863636	68.545455	29.036364	1.318182
2	3.180000	78.700000	49.300000	11.630000	3.000000
3	5.855556	130.111111	70.888889	17.027778	2.500000

Comparing the composition of states in each cluster from Part 1 (without scaling) to Part 2 (with scaling), it is clear that the scaling process did change the clustering:

Cluster 1: In Part 2, more states are included in this cluster. Notably, states like Colorado, Georgia, Missouri, Oregon, Tennessee, Texas, and Washington have moved from Cluster 2 in Part 1 to Cluster 1 in Part 2.

Cluster 2: This cluster has considerably fewer states in Part 2. These states seem to have predominantly rural characteristics, suggesting that scaling the variables might have amplified differences between rural and urban areas.

Cluster 3: Several states have moved from Cluster 2 and 3 in Part 1 to Cluster 3 in Part 2, including Arkansas, Massachusetts, New Jersey, Oklahoma, Rhode Island, Virginia, and Wyoming. The characteristics of these states in terms of the variables could have been masked in the unscaled data due to the larger scale of certain variables. Scaling has allowed these similarities to emerge, resulting in the states clustering together in Part 2.

Differences between clusters:

Cluster 1: The mean values of Murder, Assault, and Rape are slightly lower in Part 2 than in Part 1. The UrbanPop mean value is slightly higher in Part 2. This suggests that the scaling has slightly moderated the high crime rates in this cluster while increasing the urban population proportion.

Cluster 2: In Part 2, the mean values of all variables are considerably lower than in Part 1, especially for Assault. This could imply that scaling has greatly impacted this cluster, possibly reducing the influence of the Assault variable due to its originally larger scale compared to other variables.

Cluster 3: In Part 2, Murder and Assault have higher mean values while UrbanPop and Rape have lower mean values compared to Part 1. This indicates that scaling has also affected this cluster, increasing the relative impact of Murder and Assault while decreasing that of UrbanPop and Rape.

The variables should be scaled:

While the differences between the scaled and unscaled data might not appear significant at first glance, it's important to consider that the values being compared are average statistics across multiple states. Even minor changes in these average values can signify meaningful shifts in the data distribution.

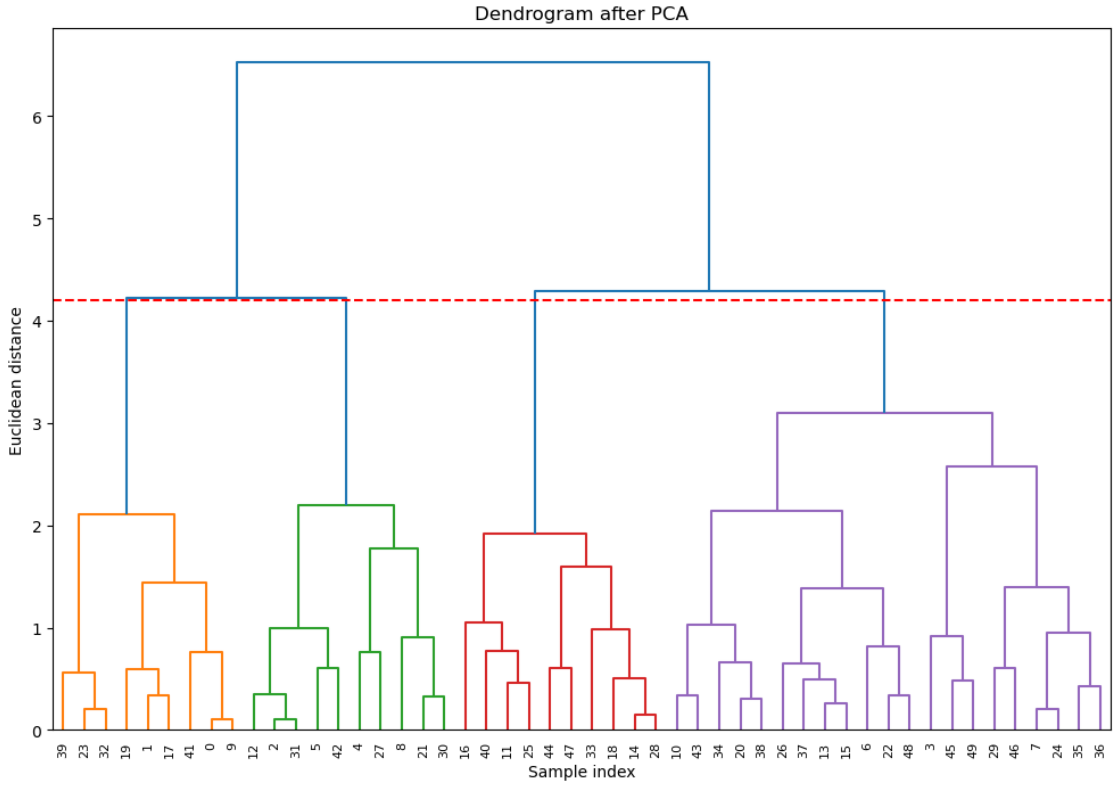
On top of this, in the case of Cluster 3, several states moved from other clusters to Cluster 3 in Part 2 after scaling. This can suggest that the unscaled data could have masked similarities between these states due to the larger scale of certain variables such as Assault. Scaling allowed these similarities to be detected. Again for Cluster 2, scaling seems to have amplified differences between states with predominantly rural characteristics and other states. This again suggests that the unscaled data might have obscured some differences, and scaling helped expose these differences, resulting in more distinct clusters.

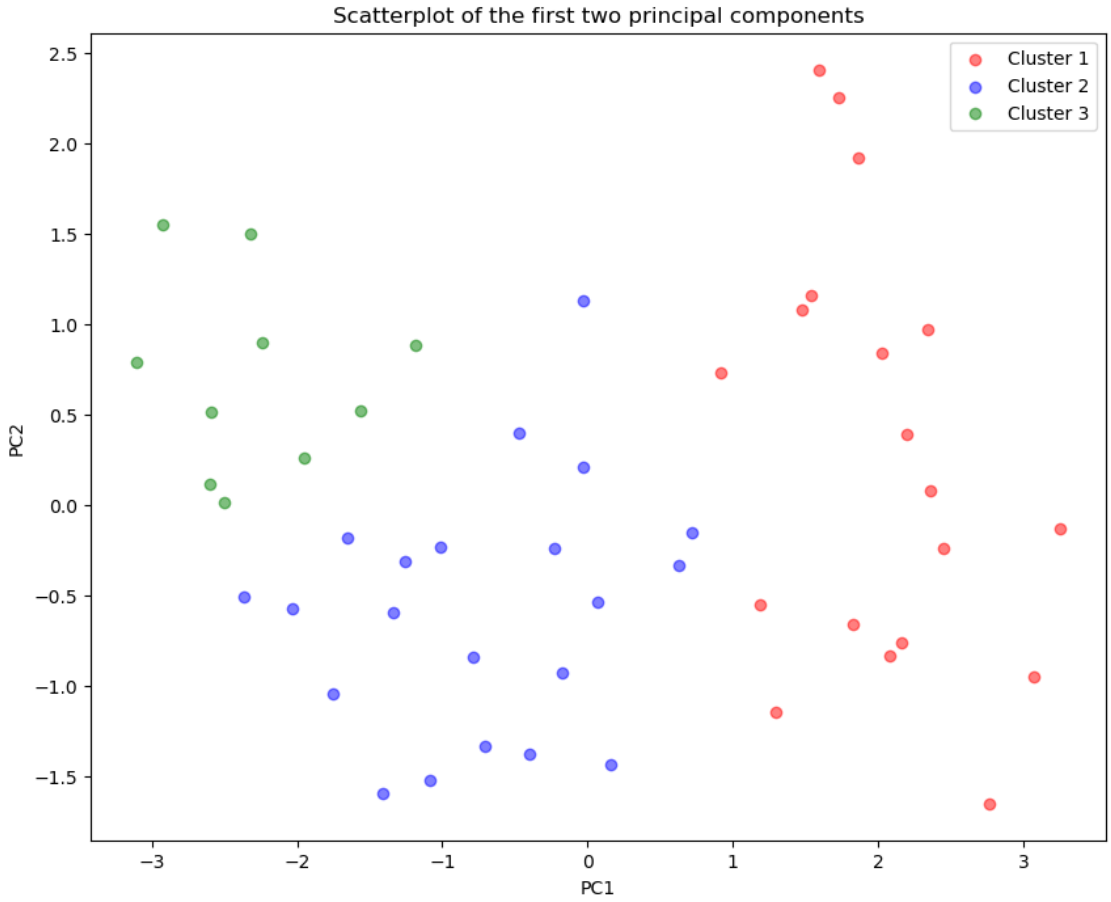
2.1.3 Task 3:

Perform PCA on the data. Now perform hierarchical clustering with complete linkage and Euclidean distance on the first two principal component score vectors rather than the raw data. Cut the dendrogram at a height that results in three distinct clusters. Present the scatterplot of the first two principal components using different colours for the instances on each cluster (three colours for three clusters). Compare the group characteristics to the group characteristics obtained in Task 2.

[31]:
```
# Perform PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

# Perform hierarchical clustering on PCA data
Z_pca = linkage(pca_data, 'complete', metric = 'euclidean')



# Plot the dendrogram
plt.figure(figsize=(12, 8))
dendrogram(Z_pca, leaf_rotation=90, color_threshold=4.2)
plt.axhline(y=4.2, color='r', linestyle='--')  # Cut at a suitable threshold to␣
 ↪get 3 clusters
plt.title('Dendrogram after PCA')
plt.xlabel('Sample index')
plt.ylabel('Euclidean distance')
plt.show()

# Assign labels based on the hierarchical clustering
cutree_result = cut_tree(Z_pca, n_clusters=3)

# Create scatterplot of the first two principal components
plt.figure(figsize=(10, 8))
plt.scatter(pca_data[cutree_result[:, 0] == 0, 0], pca_data[cutree_result[:, 0]␣
 ↪== 0, 1], color='red', alpha=0.5, label='Cluster 1')
```

```
plt.scatter(pca_data[cutree_result[:, 0] == 1, 0], pca_data[cutree_result[:, 0]␣
 ↪== 1, 1], color='blue', alpha=0.5, label='Cluster 2')
plt.scatter(pca_data[cutree_result[:, 0] == 2, 0], pca_data[cutree_result[:, 0]␣
 ↪== 2, 1], color='green', alpha=0.5, label='Cluster 3')
plt.title('Scatterplot of the first two principal components')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()
```





[32]:
```
data['Cluster_PCA'] = cutree_result.flatten()
cluster_characteristics_pca = data.groupby('Cluster_PCA').
  ↪mean(numeric_only=True)
print(cluster_characteristics_pca)
```

```
                Murder     Assault   UrbanPop       Rape   Cluster  \
Cluster_PCA
0            12.331579  259.315789  68.315789  29.215789  1.210526
1             5.614286  134.142857  71.238095  18.590476  2.428571
2             3.720000   79.400000  48.300000  11.610000  3.000000

             Cluster_Scaled
Cluster_PCA
0                  1.000000
1                  2.666667
2                  2.100000
```

When comparing the mean statistics for each cluster obtained from the PCA and hierarchical clustering (Task 3) to those obtained from the hierarchical clustering with scaling (Task 2), the

following can be observed:

Cluster 1: In Task 3, Cluster 1 has slightly higher mean values for Murder, Assault, and Rape, and similar UrbanPop mean value compared to Task 2. This suggests that PCA might have attributed more weight to these three crime-related variables, making the states with higher crime rates more distinct in the PCA-based clusters.

Cluster 2: Cluster 2 in Task 3 has slightly higher mean values for all variables compared to Task 2, except for UrbanPop. This might indicate that PCA reduced the impact of the UrbanPop variable, making this cluster more homogenous in terms of urban population proportion.

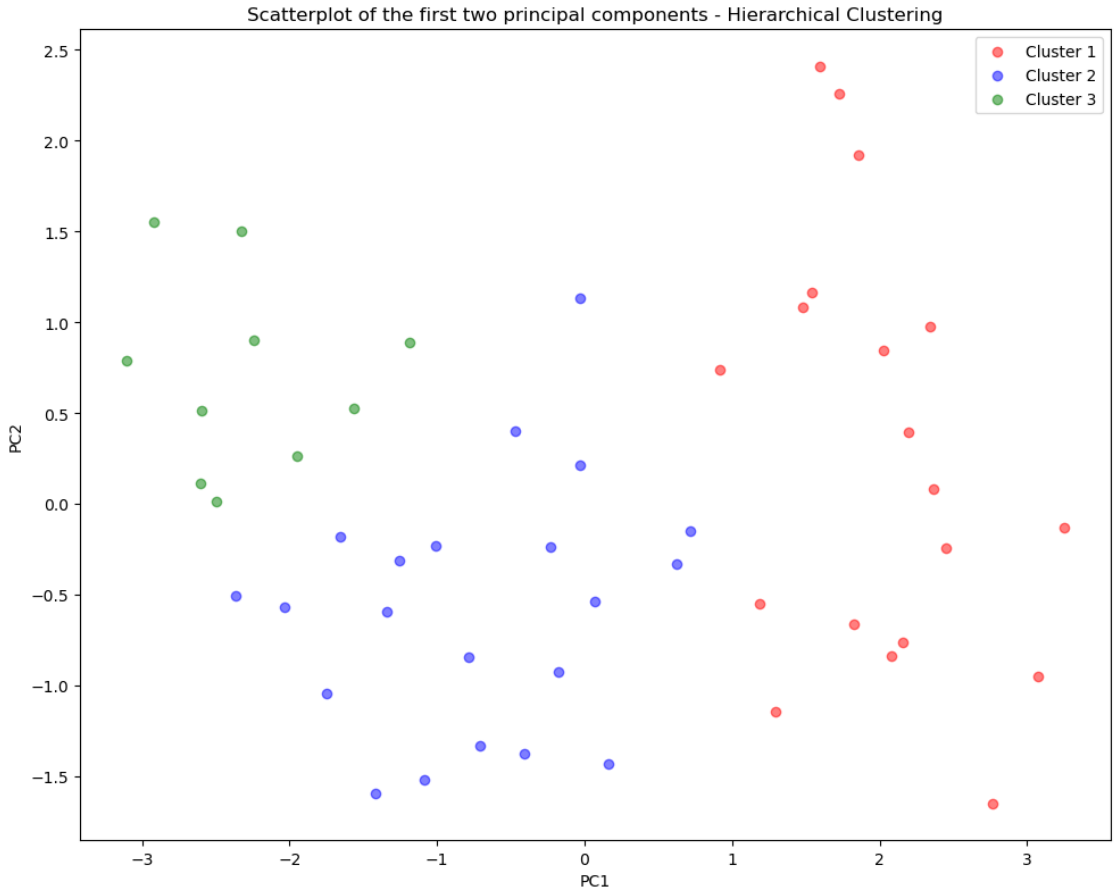
Cluster 3: The mean values of all variables in Cluster 3 are relatively close in both tasks, with a slightly higher mean value for Murder and lower for Assault in Task 3. This suggests that PCA did not greatly affect the composition of this cluster.

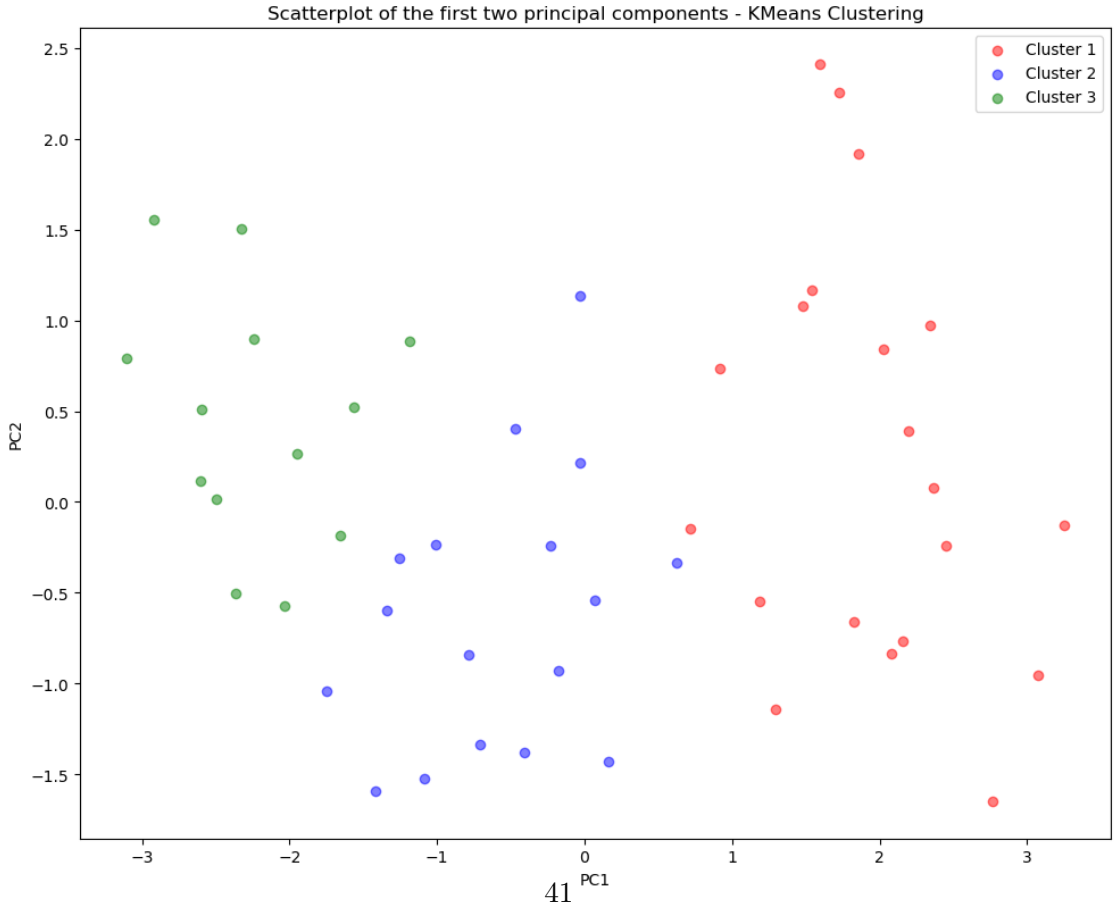
2.1.4 Task 4:

Repeat the analysis of Task 3 using the K-means clustering (with K=3). That is, use the first two principal components score vectors as features and set the initial centroids of the K-means as the group means obtained from the hierarchical clustering on Task 3. Compare the results from the Kmeans clustering to the results from the hierarchical clustering of Task 3. Which one do you think provides a better result? For the K-means clustering, you should use the Euclidean distance and set random state to "5508".

[33]:
```
# Perform PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
# Extract cluster centroids from hierarchical clustering
centroids_hierarchical = np.array([pca_data[cutree_result[:, 0] == i].
 ↪mean(axis=0) for i in range(3)])

# KMeans clustering
kmeans = KMeans(n_clusters=3, init=centroids_hierarchical, n_init=1,␣
 ↪random_state=5508)
kmeans_labels = kmeans.fit_predict(pca_data)

#Creating Subplots
fig, axs = plt.subplots(2, 1, figsize=(10, 16))

# Plot hierarchical clustering scatterplot
axs[0].scatter(pca_data[cutree_result[:, 0] == 0, 0], pca_data[cutree_result[:,␣
 ↪0] == 0, 1], color='red', alpha=0.5, label='Cluster 1')
axs[0].scatter(pca_data[cutree_result[:, 0] == 1, 0], pca_data[cutree_result[:,␣
 ↪0] == 1, 1], color='blue', alpha=0.5, label='Cluster 2')
axs[0].scatter(pca_data[cutree_result[:, 0] == 2, 0], pca_data[cutree_result[:,␣
 ↪0] == 2, 1], color='green', alpha=0.5, label='Cluster 3')
axs[0].set_title('Scatterplot of the first two principal components -␣
 ↪Hierarchical Clustering')
axs[0].set_xlabel('PC1')
```

```
axs[0].set_ylabel('PC2')
axs[0].legend()

# Plot K-means clustering scatterplot
axs[1].scatter(pca_data[kmeans_labels == 0, 0], pca_data[kmeans_labels == 0,␣
 ↪1], color='red', alpha=0.5, label='Cluster 1')
axs[1].scatter(pca_data[kmeans_labels == 1, 0], pca_data[kmeans_labels == 1,␣
 ↪1], color='blue', alpha=0.5, label='Cluster 2')
axs[1].scatter(pca_data[kmeans_labels == 2, 0], pca_data[kmeans_labels == 2,␣
 ↪1], color='green', alpha=0.5, label='Cluster 3')
axs[1].set_title('Scatterplot of the first two principal components - KMeans␣
 ↪Clustering')
axs[1].set_xlabel('PC1')
axs[1].set_ylabel('PC2')
axs[1].legend()

# Show the plot
plt.tight_layout()
plt.show()
```


Scatterplot of the first two principal components - Hierarchical Clustering
Cluster 1
Cluster 2
Cluster 3
PC2
PC1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3


Scatterplot of the first two principal components - KMeans Clustering
Cluster 1
Cluster 2
Cluster 3
PC2
PC1
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3

[34]:
```
data['Cluster_KMeans'] = kmeans_labels
cluster_characteristics_kmeans = data.groupby('Cluster_KMeans').
  ↪mean(numeric_only=True)
print(cluster_characteristics_kmeans)
```

```
                   Murder     Assault   UrbanPop       Rape   Cluster  \
Cluster_KMeans
0               12.010000  258.250000  68.500000  28.545000  1.200000
1                6.023529  138.352941  72.352941  19.552941  2.411765
2                3.600000   78.538462  52.076923  12.176923  3.000000

                Cluster_Scaled  Cluster_PCA
Cluster_KMeans
0                     1.100000     0.050000
1                     2.647059     1.000000
2                     2.230769     1.769231
```

While it is quite hard to say which method, KMeans vs hierarchical clustering, some observations can be made:

Compactness and Separation: Comparing the mean values in each cluster for both methods, the observation can be made that the hierarchical clustering method has somewhat less variation within clusters for the 'Murder', 'Assault', and 'Rape' features than the K-means method. This can be seen from the fact that the means of these features in each cluster are more distinct in the hierarchical clustering results, suggesting better separation between clusters. However, the 'UrbanPop' feature is more similar across clusters for the hierarchical method compared to the K-means method.

Consistency of Clusters: Looking at the 'Cluster' column in the two tables (which we assume is the original cluster assignment before PCA was applied), we can see that the cluster assignments in the hierarchical method are slightly more consistent with the original clusters. In particular, the mean cluster number in Cluster 1 of the hierarchical method (1.21) is closer to 1 than in the K-means method (1.20), and likewise for Cluster 3 (3.00 vs 1.77). This might suggest that the hierarchical method is more consistent with the original clusters.

Analysing the Scatter Plots

I believe hierarchical clustering provides better seperation of the groups when comparing the scatterplots compared to the Kmeans clustering method. While both plots are similar the hierarchical clustering method results in a clearer gap, in my opinion, between the blue and green points. Similarly Kmeans has a red point right next to a blue point which doesn't seem correct, when looking at the hierarchical clustering plot both these points are blue which makes sense due to their close proximity to one another.

Hierarchical clustering generally has somewhat less variation within the clusters for the Murder', 'Assault', and 'Rape' features. The clusters seem to be more consistent with the original clusters and the scatterplots seem to show a better seperation of points in the hierarchical clustering plot. Therefore, I believe it is the better method for this data set.