Step 1: Load and Inspect the Data

In [ ]:

```
import pandas as pd

# Load the dataset
df = pd.read_csv("Car_Evaluation Original Dataset.csv")
df.head()
```

Out[ ]:

	Buying_Price	Maintenance_Price	No_of_Doors	persons	lug_boot	safety	cla
0	vhigh	vhigh	2	2	small	low	
1	vhigh	vhigh	2	2	small	med	
2	vhigh	vhigh	2	2	small	high	
3	vhigh	vhigh	2	2	med	low	
4	vhigh	vhigh	2	2	med	med	

Step 1.1: Check for Missing Values & Data Types

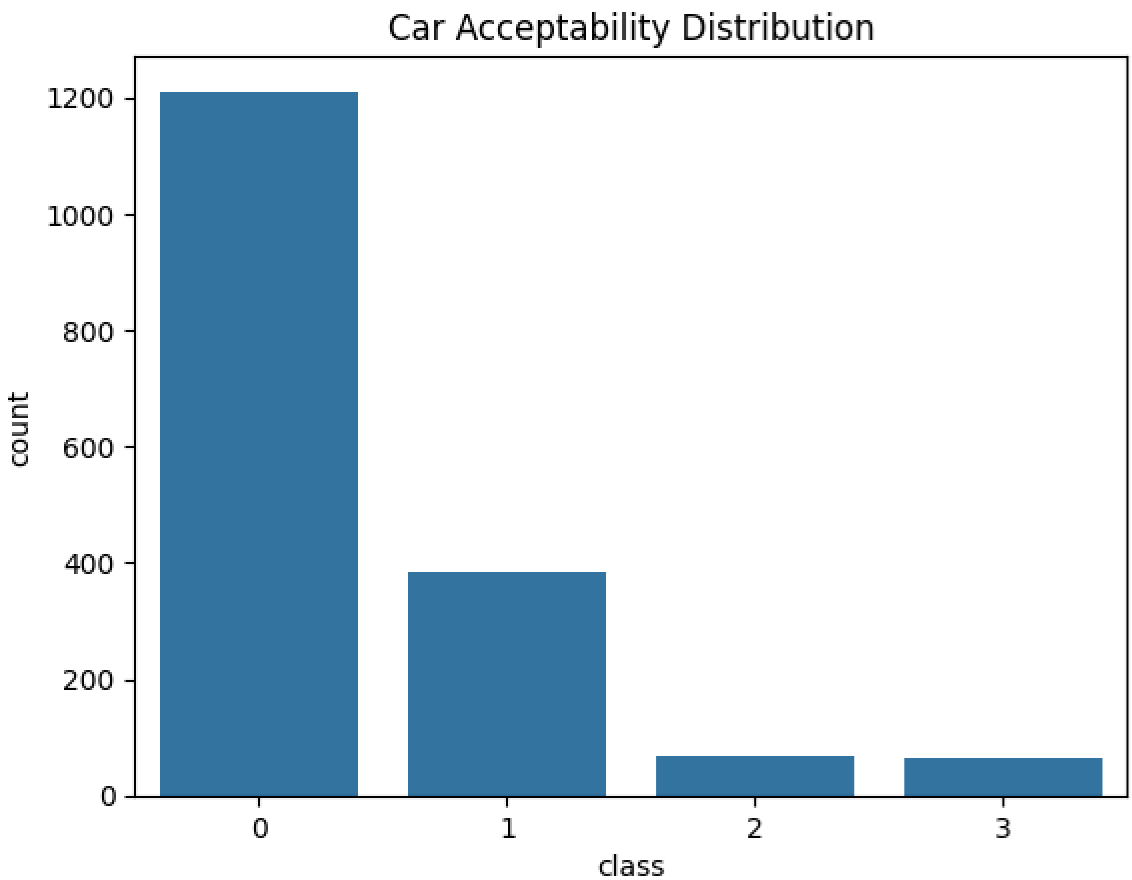
In [ ]:

```
print(df.info())
print("\nMissing values:\n", df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Buying_Price       1728 non-null   object
 1   Maintenance_Price  1728 non-null   object
 2   No_of_Doors        1728 non-null   object
 3   persons            1728 non-null   object
 4   lug_boot           1728 non-null   object
 5   safety             1728 non-null   object
 6   class              1728 non-null   int64 
dtypes: int64(1), object(6)
memory usage: 94.6+ KB
None

Missing values:
 Buying_Price         0
Maintenance_Price    0
No_of_Doors          0
persons              0
lug_boot             0
safety               0
class                0
dtype: int64
```

Step 2: EDA - Feature Distributions and Class Balance

In [ ]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Plot class distribution
sns.countplot(x='class', data=df)
plt.title('Car Acceptability Distribution')
plt.show()

# Pairplot for feature relationships
# sns.pairplot(df, hue='class')
```



Step 3: Classification Models - Train & Compare

In [ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix

# Separate features and target
X = df.drop("class", axis=1)
y = df["class"]

# Encode the target
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split data
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.
```

Logistic Regression, k-NN, Decision Tree, Random Forest, SVM

In [ ]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report

# Load your dataset here
# df = pd.read_csv('your_data.csv')

# Example assumes 'class' is your target column
X = df.drop('class', axis=1)
y = df['class']

# One-hot encode categorical features
X_encoded = pd.get_dummies(X)

# Encode target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y_encoded, test_size=0.2, random_state=42
)

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "k-NN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    print(f"\n{name}:\n")
    print(confusion_matrix(y_test, preds))
    target_names = [str(c) for c in le.classes_]
    print(classification_report(y_test, preds, target_names=target_names))
```

```
Logistic Regression:

[[228   7   0   0]
 [  9  68   6   0]
 [  0   4   6   1]
 [  0   2   0  15]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       235
           1       0.84      0.82      0.83        83
           2       0.50      0.55      0.52        11
           3       0.94      0.88      0.91        17

    accuracy                           0.92       346
   macro avg       0.81      0.80      0.81       346
weighted avg       0.92      0.92      0.92       346


k-NN:

[[229   6   0   0]
 [ 28  51   4   0]
 [  2   5   3   1]
 [  1   9   1   6]]
              precision    recall  f1-score   support

           0       0.88      0.97      0.93       235
           1       0.72      0.61      0.66        83
           2       0.38      0.27      0.32        11
           3       0.86      0.35      0.50        17

    accuracy                           0.84       346
   macro avg       0.71      0.55      0.60       346
weighted avg       0.82      0.84      0.82       346


Decision Tree:

[[235   0   0   0]
 [  3  74   4   2]
 [  0   0  10   1]
 [  0   1   2  14]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       235
           1       0.99      0.89      0.94        83
           2       0.62      0.91      0.74        11
           3       0.82      0.82      0.82        17

    accuracy                           0.96       346
   macro avg       0.86      0.91      0.87       346
weighted avg       0.97      0.96      0.96       346
```

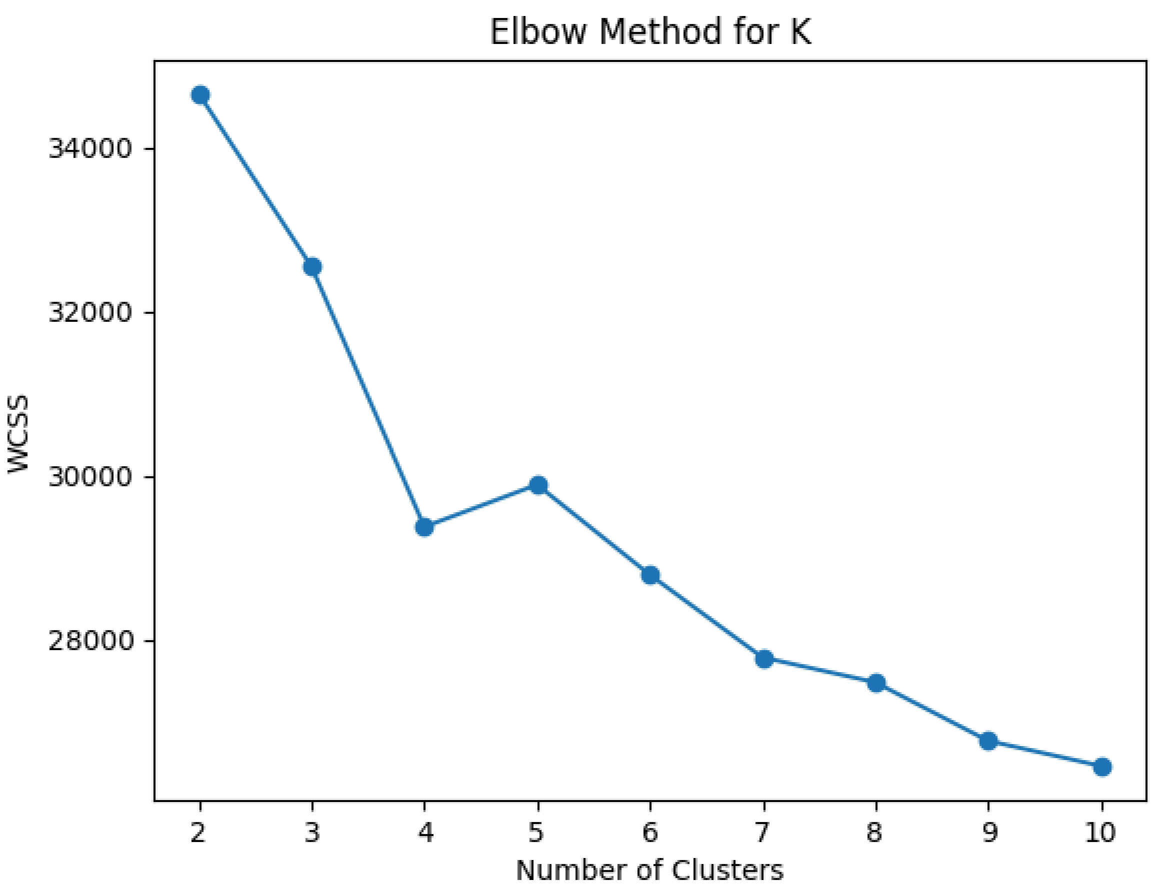
```
Random Forest:

[[235   0   0   0]
 [  0  76   7   0]
 [  0   0   9   2]
 [  0   3   1  13]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       235
           1       0.96      0.92      0.94        83
           2       0.53      0.82      0.64        11
           3       0.87      0.76      0.81        17

    accuracy                           0.96       346
   macro avg       0.84      0.87      0.85       346
weighted avg       0.97      0.96      0.96       346


SVM:

[[235   0   0   0]
 [  0  74   7   2]
 [  0   0  10   1]
 [  0   1   0  16]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       235
           1       0.99      0.89      0.94        83
           2       0.59      0.91      0.71        11
           3       0.84      0.94      0.89        17

    accuracy                           0.97       346
   macro avg       0.85      0.94      0.88       346
weighted avg       0.98      0.97      0.97       346
```

Step 4: K-Means This section demonstrates how to apply clustering techniques to a dataset that contains categorical features such as 'vhigh', 'low', etc. Since most clustering algorithms require numerical input, we first convert these categorical features using one-hot encoding. We then scale the data using StandardScaler before applying clustering algorithms.

In [ ]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Assuming your dataset df is loaded, and X contains features with categorical
# For example:
# X = df.drop('target_column', axis=1)  # no target column for clustering
```

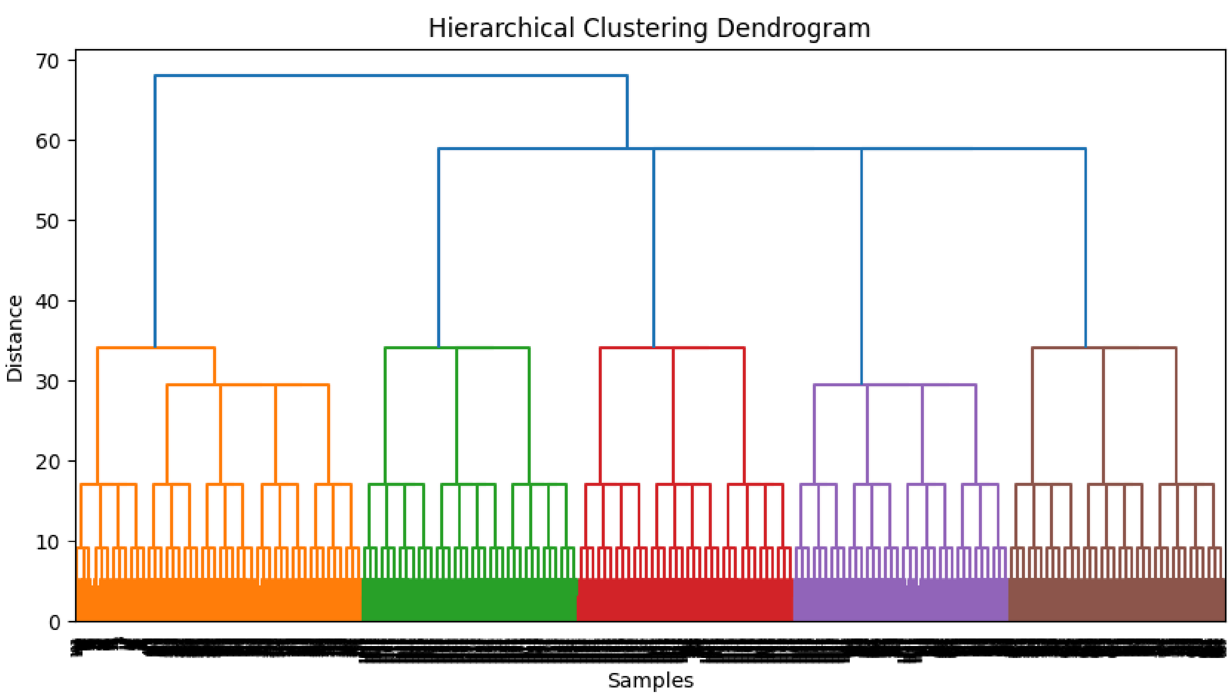
```python
# Step 1: One-hot encode categorical features
X_encoded = pd.get_dummies(X)

# Step 2: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Step 3: Elbow method to find optimal k
wcss = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(2, 11), wcss, marker='o')
plt.title("Elbow Method for K")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()

# Step 4: Fit KMeans with chosen k (say 4)
kmeans = KMeans(n_clusters=4, random_state=42)
labels_kmeans = kmeans.fit_predict(X_scaled)

print("K-Means Silhouette Score:", silhouette_score(X_scaled, labels_kmeans))
```




```
K-Means Silhouette Score: 0.13090261027176128
```

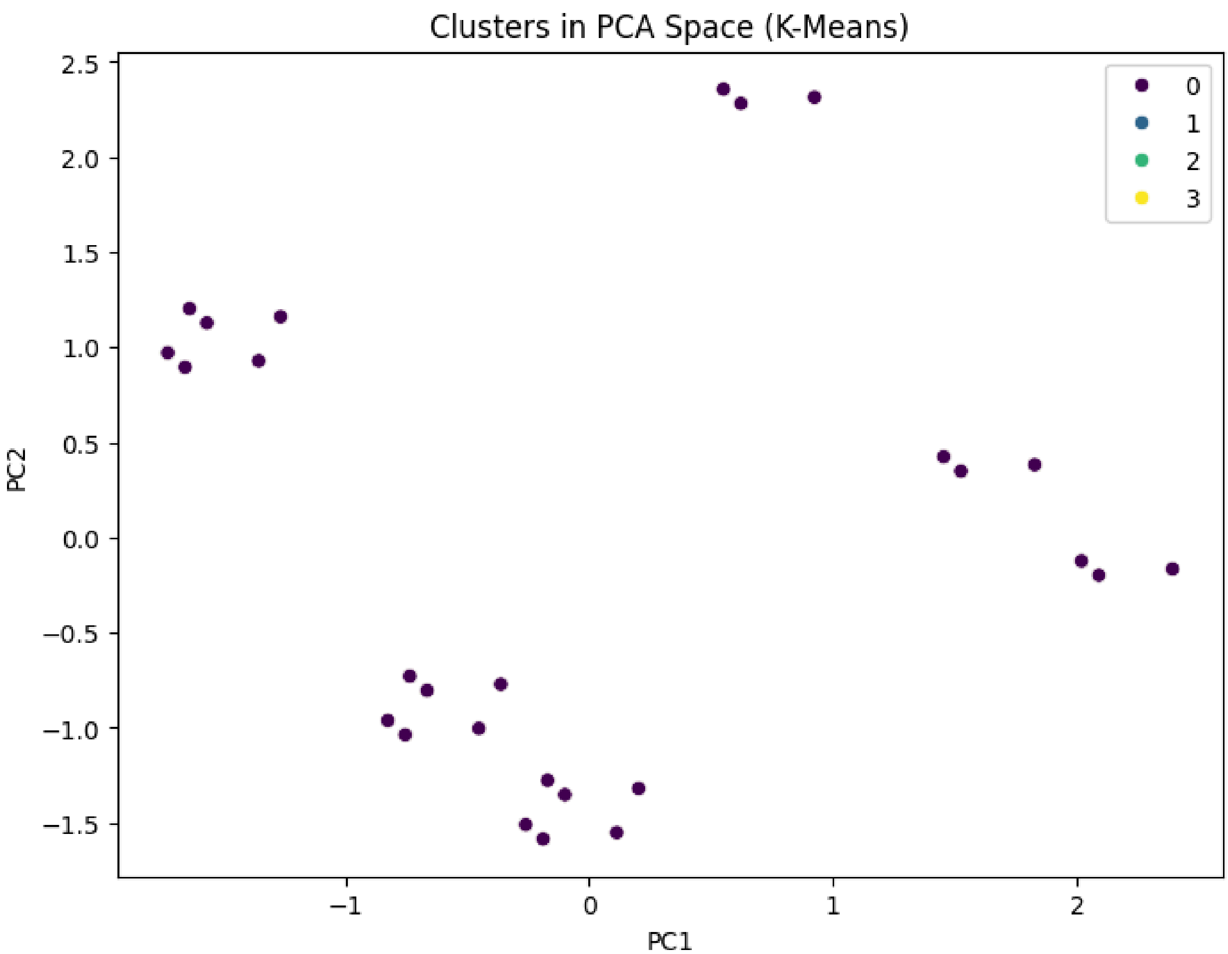
Hierarchical Clustering Hierarchical clustering builds nested clusters by merging or splitting them successively. Similar to K-Means, it also requires numeric input. So, we again use one-hot encoding followed by standard scaling. The dendrogram helps visualize how the clusters are formed and can give insight into the number of natural groupings in the data.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler

# Step 1: One-hot encode categorical features (same as above)
X_encoded = pd.get_dummies(X)

# Step 2: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# Step 3: Compute linkage matrix using Ward's method
linked = linkage(X_scaled, method='ward')

# Step 4: Plot dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linked)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```



Step 5: Apply PCA and Visualize Components

In [ ]:
```
from sklearn.decomposition import PCA

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Visualize clusters in 2D PCA space
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=labels_kmeans, palette='viri
plt.title("Clusters in PCA Space (K-Means)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()

# Explained variance
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```




```
Explained Variance Ratio: [0.07142857 0.07142857]
```

In [ ]:
```
Step 6: Compare Model Performance and Clustering Results
```

Check Class Imbalance (in detail) Add a value count to clearly see class distribution

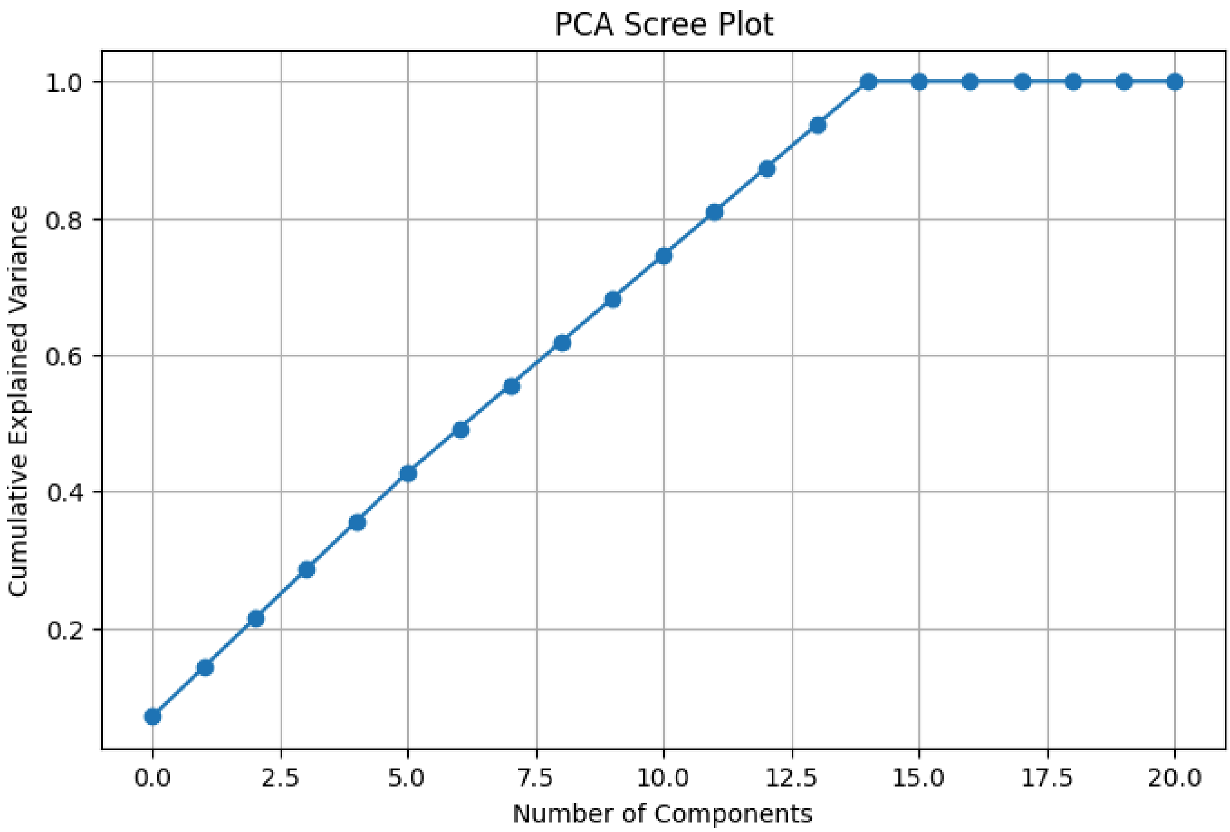
In [ ]:
```
print("Class distribution:\n", df['class'].value_counts())
```

```
Class distribution:
 class
0    1210
1     384
2      69
3      65
Name: count, dtype: int64
```

We used a Scree Plot to visualize the cumulative explained variance from Principal Component Analysis (PCA). This helps determine how many components are needed to retain most of the dataset's information. The plot shows that the first few components capture the majority of variance, making dimensionality reduction effective for visualization and clustering.

In [ ]:

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np  # Make sure this is imported

pca_full = PCA().fit(X_scaled)

# Scree Plot
plt.figure(figsize=(8, 5))
plt.plot(np.cumsum(pca_full.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('PCA Scree Plot')
plt.grid(True)
plt.show()
```



Step 6: Compare Model Performance and Clustering Results

6.1: Compare Classification Model Performance We create a performance summary table for all classifiers using Accuracy, Precision, Recall, and F1-score to evaluate and compare how well each model performs on the classification task.

In [ ]:

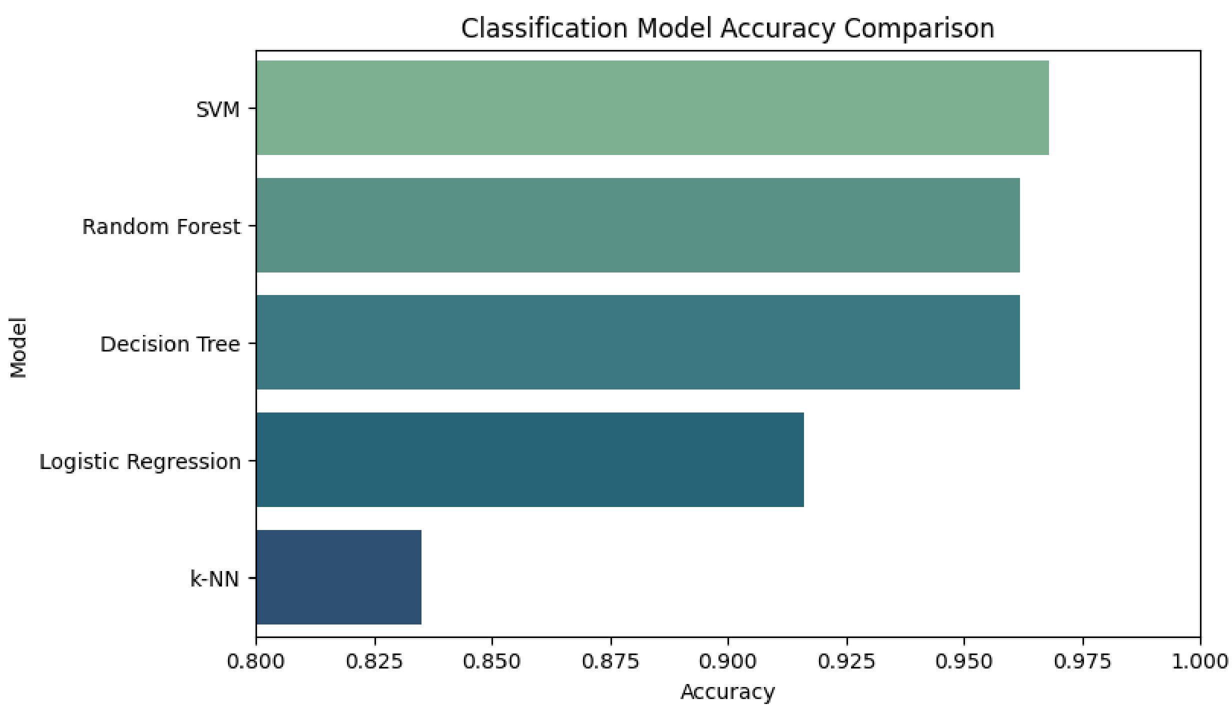
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

# Create a summary dataframe
performance = []

for name, model in models.items():
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    prec = precision_score(y_test, preds, average='weighted')
    rec = recall_score(y_test, preds, average='weighted')
    f1 = f1_score(y_test, preds, average='weighted')

    performance.append({
        "Model": name,
        "Accuracy": round(acc, 3),
        "Precision": round(prec, 3),
        "Recall": round(rec, 3),
        "F1 Score": round(f1, 3)
    })
```

```python
# Convert to DataFrame and display
performance_df = pd.DataFrame(performance)
print(performance_df.sort_values(by="Accuracy", ascending=False))
```

```
                 Model  Accuracy  Precision  Recall  F1 Score
4                  SVM     0.968      0.976   0.968     0.970
3        Random Forest     0.962      0.969   0.962     0.965
2        Decision Tree     0.962      0.968   0.962     0.964
0  Logistic Regression     0.916      0.917   0.916     0.916
1                 k-NN     0.835      0.825   0.835     0.822
```

6.2: Visual Comparison of Model Accuracy

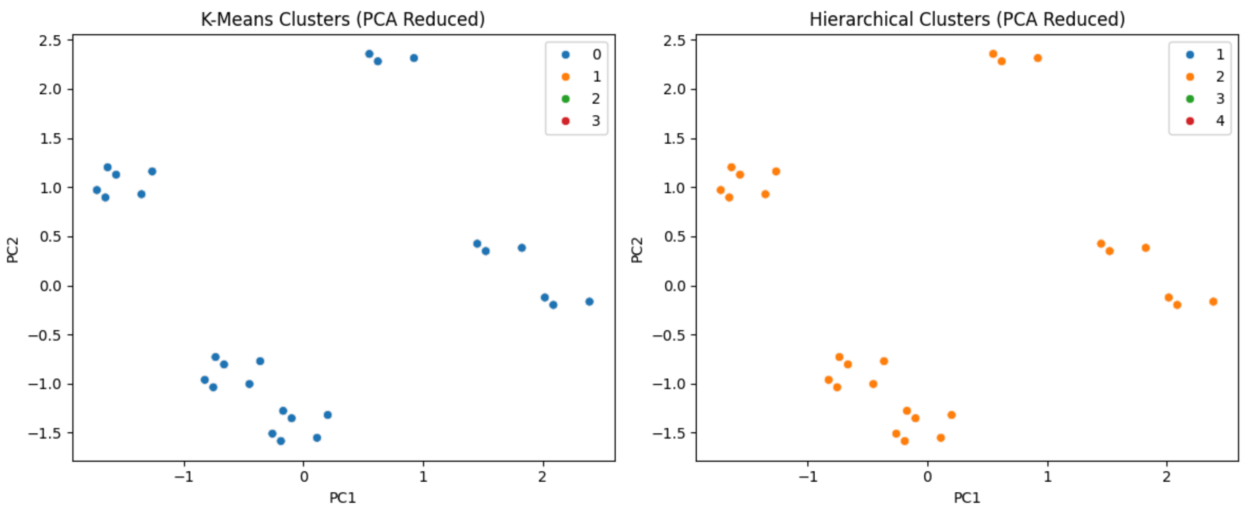
In [ ]:
```python
plt.figure(figsize=(8, 5))
sns.barplot(data=performance_df.sort_values(by="Accuracy", ascending=False), x
plt.title("Classification Model Accuracy Comparison")
plt.xlabel("Accuracy")
plt.ylabel("Model")
plt.xlim(0.8, 1.0)  # Adjust as per your dataset
plt.show()
```



6.3: Compare Clustering Results Using PCA (Visual Check) We already visualized PCA clusters in Step 5 using labels_kmeans. Let's add hierarchical clustering labels to the PCA plot and compare.

In [ ]:
```python
# Get labels for hierarchical clustering
from scipy.cluster.hierarchy import fcluster

# Choose number of clusters, e.g., 4
labels_hierarchical = fcluster(linked, t=4, criterion='maxclust')
```

```python
# Plot KMeans Clustering
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=labels_kmeans, palette='tab1
plt.title("K-Means Clusters (PCA Reduced)")
plt.xlabel("PC1")
plt.ylabel("PC2")

# Plot Hierarchical Clustering
plt.subplot(1, 2, 2)
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=labels_hierarchical, palette
plt.title("Hierarchical Clusters (PCA Reduced)")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.tight_layout()
plt.show()
```



6.4: Clustering Evaluation (Silhouette Scores)

In [ ]:
```python
from sklearn.metrics import silhouette_score

sil_kmeans = silhouette_score(X_scaled, labels_kmeans)
sil_hierarchical = silhouette_score(X_scaled, labels_hierarchical)

print(f"K-Means Silhouette Score: {sil_kmeans:.3f}")
print(f"Hierarchical Clustering Silhouette Score: {sil_hierarchical:.3f}")
```

```
K-Means Silhouette Score: 0.131
Hierarchical Clustering Silhouette Score: 0.094
```

Summary Classification models are evaluated using accuracy, precision, recall, and F1-score.

Clustering models (K-Means and Hierarchical) are compared using silhouette scores and visual PCA plots.

These evaluations help identify which models capture the patterns in data most effectively.