Import Libraries

```
# imports
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

Read in CSV

```
from google.colab import drive
drive.mount('/content/drive')
# create path to data
dataset_path = '/content/drive/MyDrive/heart_failure_clinical_records_dataset.csv
# print path to directory to check
print(dataset_path)
# reading the csv
data = pd.read_csv(dataset_path)
# turn data into a pandas dataframe
df = pd.DataFrame(data)
```

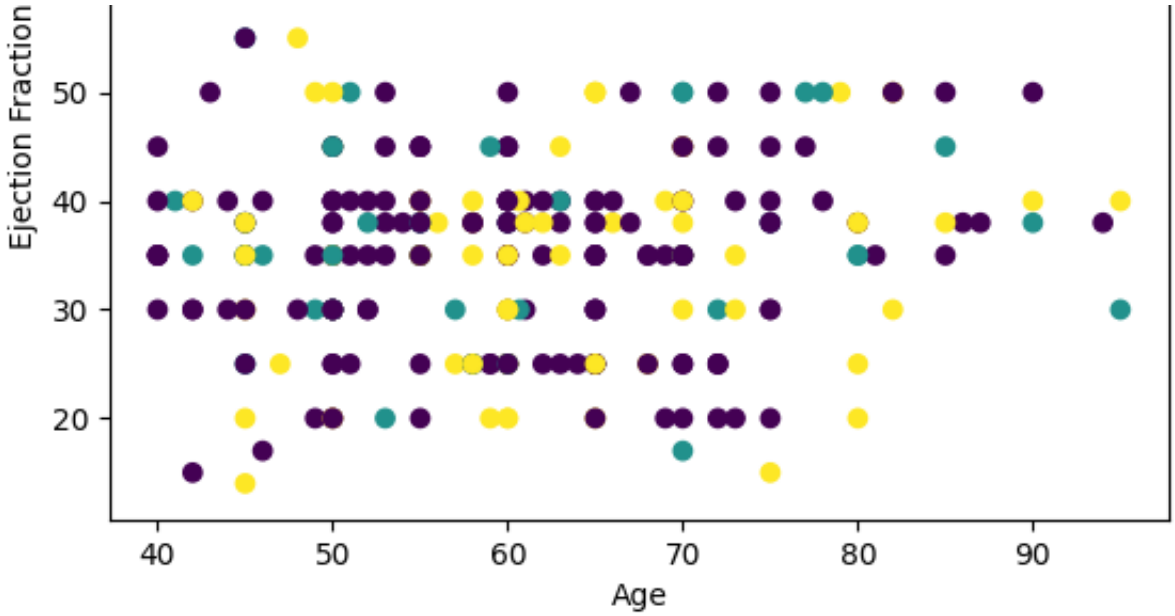
```
Mounted at /content/drive
/content/drive/MyDrive/heart_failure_clinical_records_dataset.csv
```

Task 1: Clustering Analysis

1.1. Apply K-Means clustering to the dataset using all features except 'DEATH_EVENT.'

```
# Drop the 'DEATH_EVENT' column
data = data.drop('DEATH_EVENT', axis=1)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3)  # you can change the number of clusters
data['cluster'] = kmeans.fit_predict(data)

# Print the first 5 rows of the clustered data
print(data.head())

# Plot the data
plt.scatter(data['age'], data['ejection_fraction'], c=data['cluster'])
plt.xlabel('Age')
plt.ylabel('Ejection Fraction')
plt.title('K-Means Clustering')
plt.show()
```

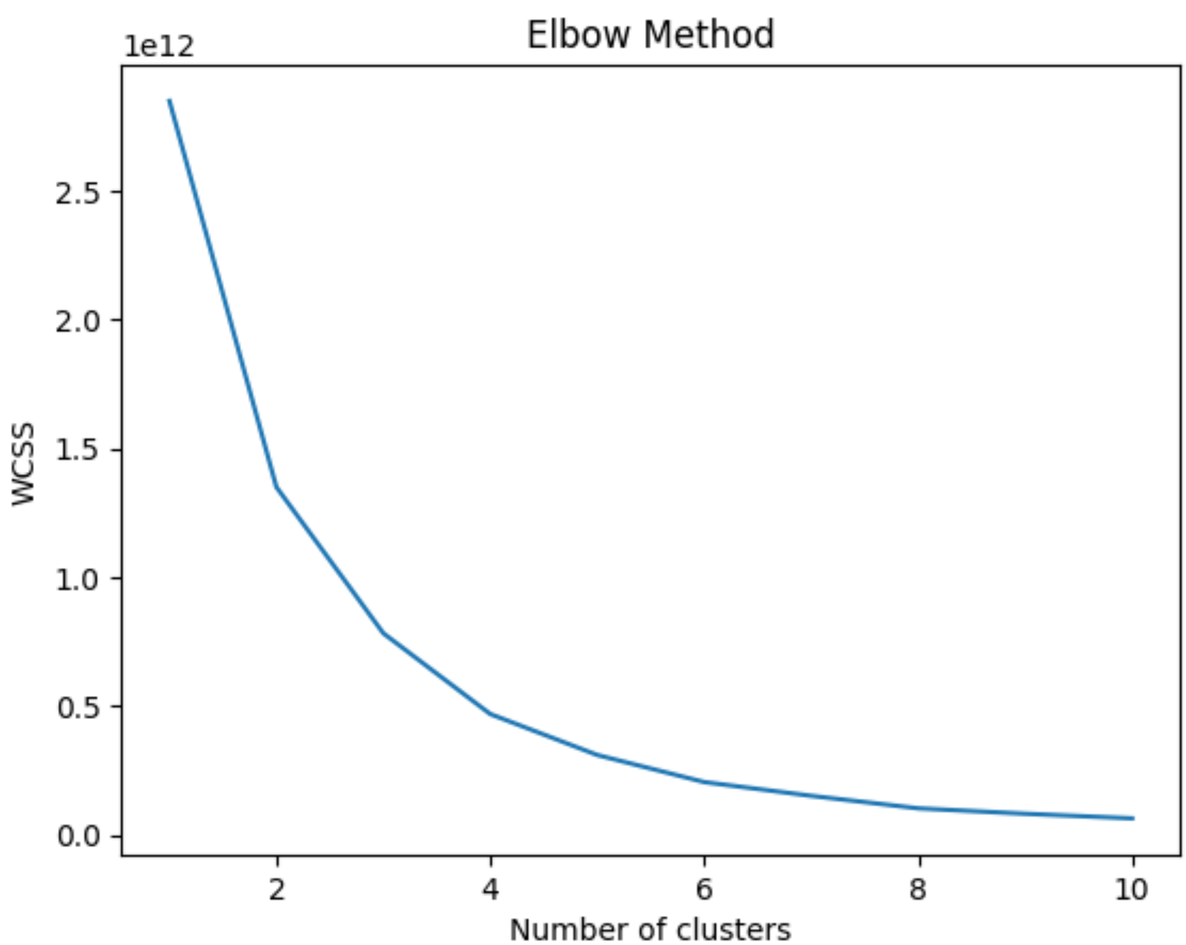
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20
1  55.0        0                      7861         0                 38
2  65.0        0                       146         0                 20
3  50.0        1                       111         0                 20
4  65.0        1                       160         1                 20

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1
1                    0  263358.03               1.1           136    1
2                    0  162000.00               1.3           129    1
3                    0  210000.00               1.9           137    1
4                    0  327000.00               2.7           116    0

   smoking  time  cluster
0        0     4        0
1        0     6        0
2        1     7        2
3        0     7        0
4        0     8        0
```



1.2. Determine the optimal number of clusters and visualize the clusters.

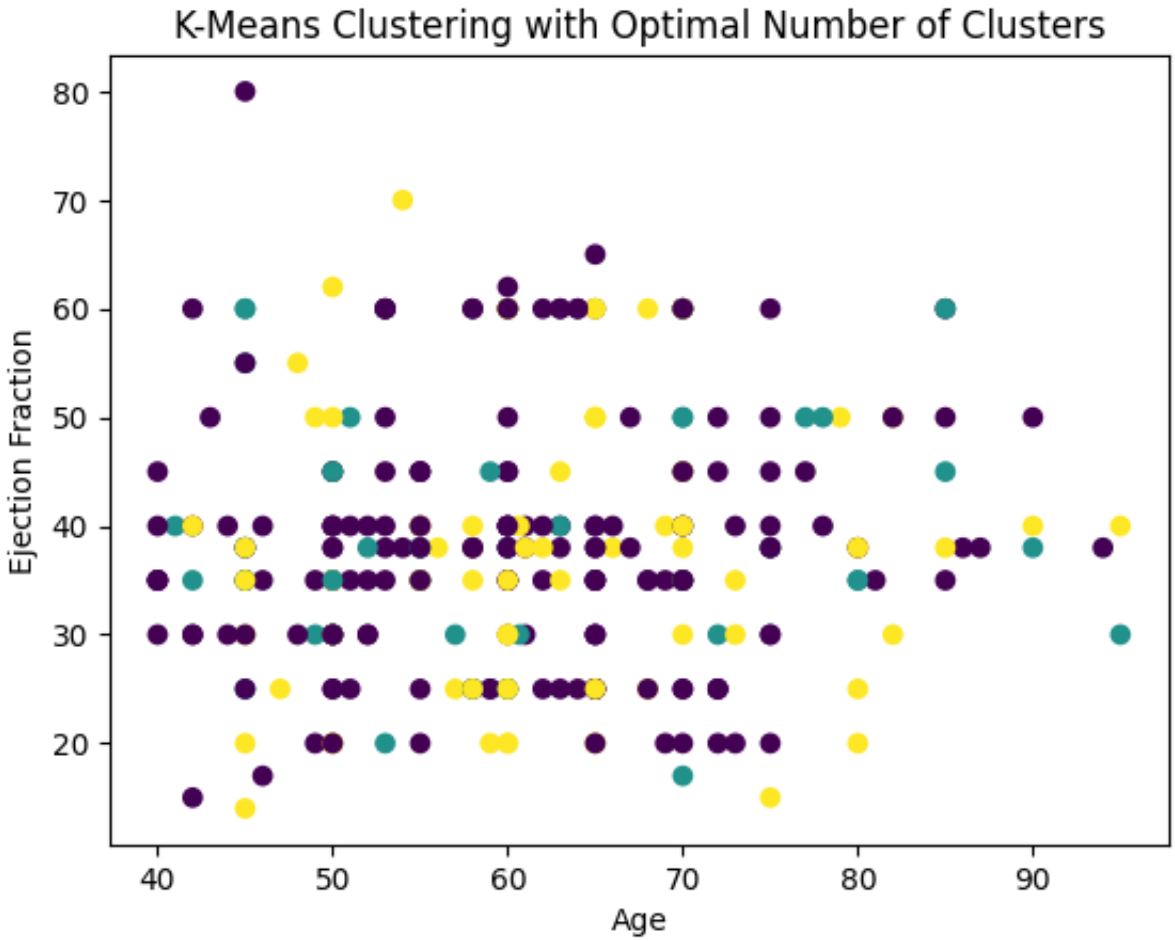
```
# Determine the optimal number of clusters using the Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```



```
# Apply K-Means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
data['cluster'] = kmeans.fit_predict(data)

# Plot the data with the optimal number of clusters
plt.scatter(data['age'], data['ejection_fraction'], c=data['cluster'])
plt.xlabel('Age')
plt.ylabel('Ejection Fraction')
plt.title('K-Means Clustering with Optimal Number of Clusters')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```

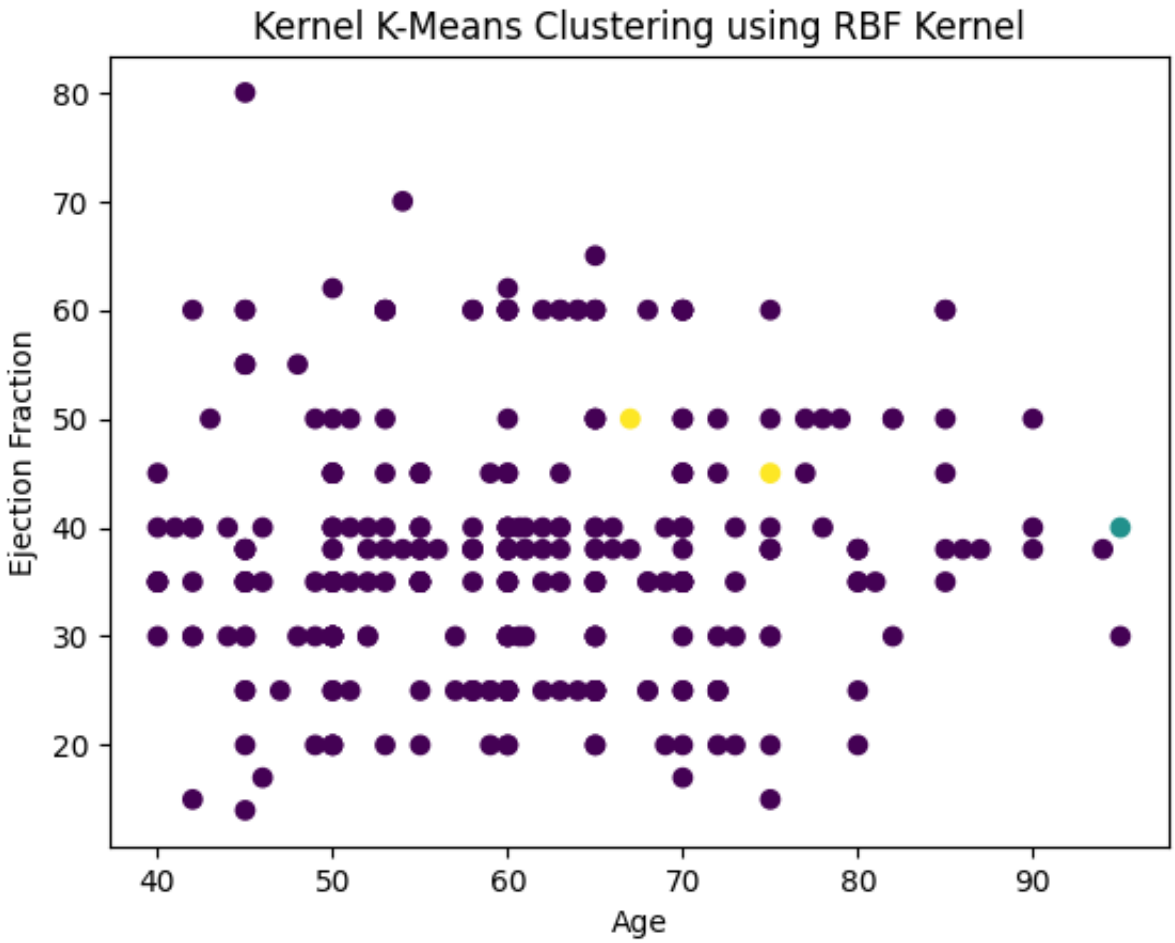


Number of clusters: 3

1.3. Implement Kernel K-Means clustering with the same features and visualize the results.

```
# Compute the RBF kernel (similarity matrix)
K = rbf_kernel(data)

# Apply K-Means clustering to the similarity matrix
kmeans = KMeans(n_clusters=3, random_state=42)
data['kernel_cluster'] = kmeans.fit_predict(K)

# Visualize the clusters
plt.scatter(data['age'], data['ejection_fraction'], c=data['kernel_cluster'])
plt.xlabel('Age')
plt.ylabel('Ejection Fraction')
plt.title('Kernel K-Means Clustering using RBF Kernel')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Future
  warnings.warn(
```



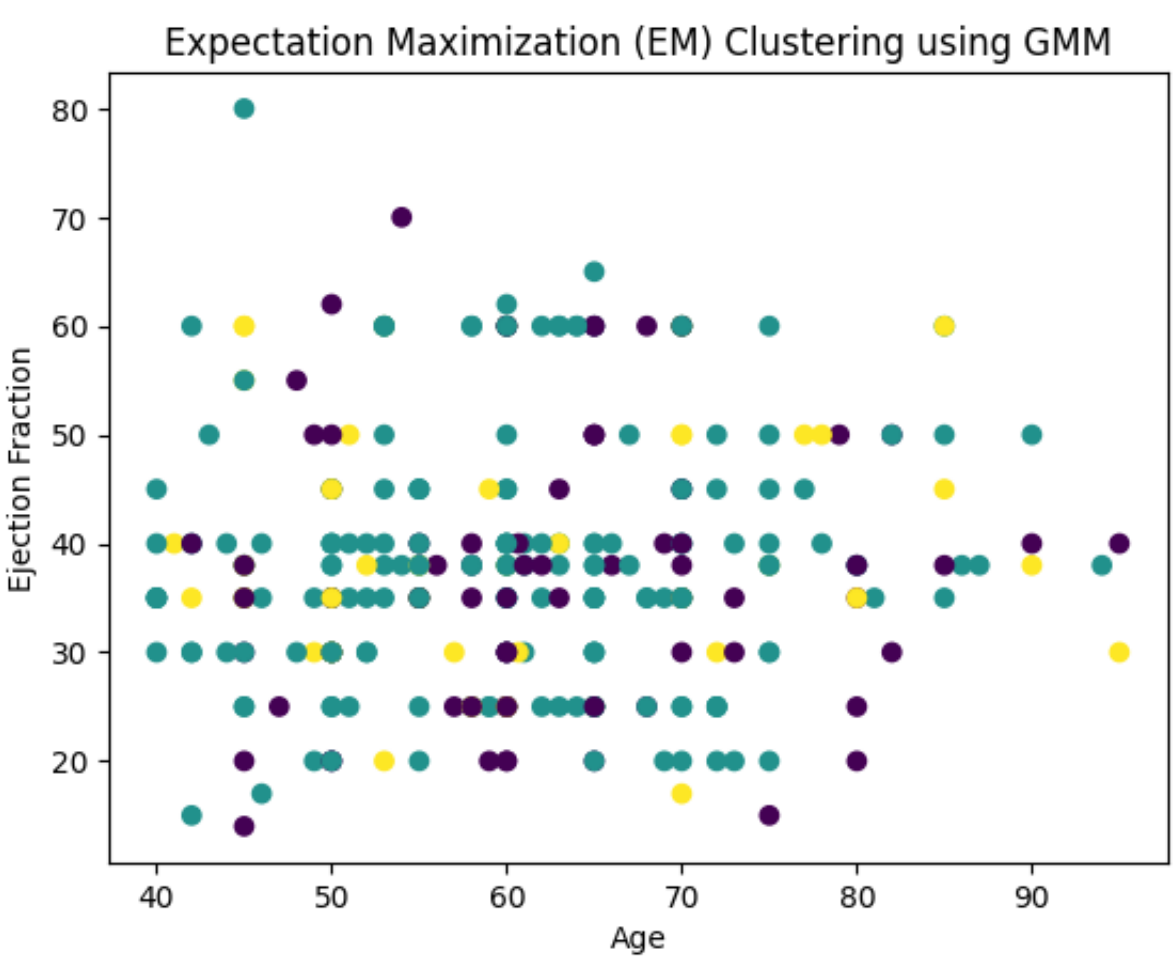
Observations:

Most of the data points are clustered together

There are a few data points (the yellow and green ones) that are separated from the main cluster. These might represent outliers or patients with distinct clinical features

1.4. Apply Expectation Maximization (EM) clustering.

```
# Remove the 'kernel_cluster' column if it exists
if 'kernel_cluster' in data.columns:
    data.drop(columns=['kernel_cluster'], inplace=True)

# Apply GMM
gmm = GaussianMixture(n_components=3, random_state=42)
data['gmm_cluster'] = gmm.fit_predict(data)

# Visualize the clusters
plt.scatter(data['age'], data['ejection_fraction'], c=data['gmm_cluster'])
plt.xlabel('Age')
plt.ylabel('Ejection Fraction')
plt.title('Expectation Maximization (EM) Clustering using GMM')
plt.show()
```



1.5. Compare and contrast the clustering results from K-Means, Kernel K-Means, and EM.

Comparison:

The "K-Means Clustering" output demonstrates distinct and well-separated clusters, predominantly influenced by the "Ejection Fraction" metric. On the other hand, the "Kernel K-Means" method, while maintaining the emphasis on the "Ejection Fraction", portrays a nuanced interrelation with the age parameter, suggesting a multifaceted data interconnection. The "Expectation Maximization (EM) Clustering" delineation presents an overlapping cluster phenomenon, indicating that some data points have probabilities associated with multiple clusters, thereby offering a more probabilistic perspective on data categorization.

Contrasting:

The distinctions between these clustering techniques are evident when looking closely at the visual results. K-Means provides a straightforward, spherical clustering, which seems to segment data based mainly on the "Ejection Fraction", whereas the Kernel K-Means, while still relying on "Ejection Fraction", seems to incorporate age more into its clustering decision. This inclusion of age, along with a more complex decision boundary, allows for a more nuanced clustering, especially for non-linearly separable data. EM, however, stands out by offering a softer boundary and allowing for overlaps. It portrays a more probabilistic view, giving each data point a potential membership in multiple clusters, hence the overlap in the visualization. Each method offers a unique lens through which to view and interpret the data, with strengths and potential pitfalls evident in each visualization.

1.6. Analyze the clinical significance of patient clusters in predicting 'DEATH_EVENT.'

```
print(data.columns)
```

```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time', 'cluster
       'gmm_cluster'],
      dtype='object')
```

```
merged_data = df.copy()
merged_data['cluster'] = data['cluster']

# 1. Distribution of 'DEATH_EVENT' Across Clusters
death_event_distribution = merged_data.groupby('cluster')['DEATH_EVENT'].value_co
```

```
print(death_event_distribution)

# 2. Examine Cluster Characteristics
for cluster in death_event_distribution.index:
    cluster_data = merged_data[merged_data['cluster'] == cluster]
    print("\nCluster", cluster)
    print(cluster_data.describe())

# 3. Test for Significance
from scipy.stats import chi2_contingency

contingency_table = death_event_distribution.values
chi2, p, _, _ = chi2_contingency(contingency_table)

print("\nChi-square Value =", chi2)
print("P-value =", p)
```

```
mean           1.460227    137.477273   0.590909   0.540909  136.636364
std            1.405733      3.316545   0.497350   0.479495   87.732175
min            0.500000    131.000000   0.000000   0.000000   10.000000
25%            0.900000    136.000000   0.000000   0.000000   57.500000
50%            1.000000    138.000000   1.000000   0.000000  147.000000
75%            1.425000    139.250000   1.000000   1.000000  206.000000
max            9.400000    145.000000   1.000000   1.000000  285.000000

       DEATH_EVENT  cluster
count    44.000000     44.0
mean      0.363636      1.0
std       0.486607      0.0
min       0.000000      1.0
25%       0.000000      1.0
50%       0.000000      1.0
75%       1.000000      1.0
max       1.000000      1.0

Cluster 2
              age    anaemia  creatinine_phosphokinase   diabetes  \
count  75.000000  75.000000                 75.000000  75.000000
mean   62.182227   0.520000                426.226667   0.360000
std    11.486941   0.502964                554.047773   0.483232
min    42.000000   0.000000                 47.000000   0.000000
25%    55.000000   0.000000                 98.500000   0.000000
50%    60.000000   1.000000                212.000000   0.000000
75%    69.500000   1.000000                582.000000   1.000000
max    95.000000   1.000000               2794.000000   1.000000

       ejection_fraction  high_blood_pressure      platelets  \
count          75.000000            75.000000      75.000000
mean           36.746667             0.346667  158334.666667
```

```
std            12.684416             0.479113   43496.817197
min            14.000000             0.000000   25100.000000
25%            25.000000             0.000000  138000.000000
50%            35.000000             0.000000  166000.000000
75%            42.500000             1.000000  193000.000000
max            70.000000             1.000000  212000.000000

       serum_creatinine  serum_sodium         sex    smoking         time  \
count         75.000000     75.000000  75.000000  75.000000   75.000000
mean           1.477067    136.826667   0.760000   0.360000  129.493333
std            1.205186      4.542214   0.429959   0.483232   83.100618
min            0.600000    121.000000   0.000000   0.000000    7.000000
25%            1.000000    134.000000   1.000000   0.000000   67.500000
50%            1.100000    137.000000   1.000000   0.000000  121.000000
75%            1.450000    140.000000   1.000000   1.000000  200.500000
max            9.000000    145.000000   1.000000   1.000000  280.000000

       DEATH_EVENT  cluster
count    75.000000     75.0
mean      0.400000      2.0
std       0.493197      0.0
min       0.000000      2.0
25%       0.000000      2.0
50%       0.000000      2.0
75%       1.000000      2.0
max       1.000000      2.0

Chi-square Value = 4.056846689389792
```

Cluster 0 (180 records)

Death Events: 130 survived, 50 deceased.

Death rate: ~27.78%.

Age: Average ~60.3 years, with a range from 40 to 94 years.

Anaemia: ~39.44% of individuals have anaemia.

Creatinine Phosphokinase (CPK): Average value ~649.4, ranging from 30 to 7861.

Ejection Fraction: Average ~38.23%, ranging from 15% to 80%.

High Blood Pressure: ~34.44% of individuals have high blood pressure.

Platelets: Average count ~266,088, ranging from 213,000 to 348,000.

Serum Creatinine: Average value ~1.34, ranging from 0.6 to 6.8.

Serum Sodium: Average value ~136.33, ranging from 113 to 148.

Sex: ~61.67% are presumably male (assuming 1 denotes male).

Smoking: 30% of individuals smoke.

Cluster 1 (44 records)

Death Events: 28 survived, 16 deceased. Death rate: ~36.36%.

Age: Average ~60.8 years, with a range from 41 to 95 years.

Anaemia: ~43.18% of individuals have anaemia.

CPK: Average value ~570.7, ranging from 23 to 7702.

Ejection Fraction: Average ~39.77%, ranging from 17% to 60%.

High Blood Pressure: ~38.64% of individuals have high blood pressure.

Platelets: Average count ~431,205, ranging from 350,000 to 850,000.

Serum Creatinine: Average value ~1.48, ranging from 0.5 to 9.4.

Serum Sodium: Average value ~137.48, ranging from 131 to 145.

Sex: ~59.09% are presumably male.

Smoking: 34.09% of individuals smoke.

Custer 2 (75 records)

Death Events: 45 survived, 30 deceased. Death rate: ~40%.

Age: Average ~62.18 years, with a range from 42 to 95 years.

Anaemia: 52% of individuals have anaemia.

CPK: Average value ~426.2, ranging from 47 to 2794.

Ejection Fraction: Average ~36.75%, ranging from 14% to 70%.

High Blood Pressure: ~34.67% of individuals have high blood pressure.

Platelets: Average count ~158,335, ranging from 25,100 to 212,000.

Serum Creatinine: Average value ~1.48, ranging from 0.6 to 9.

Serum Sodium: Average value ~136.83, ranging from 121 to 145.

Sex: 76% are presumably male.

Smoking: 36% of individuals smoke.

Task 2: Regression Analysis

2.1. Split the dataset into training and testing sets.

```
X = df.drop('DEATH_EVENT', axis=1)  # features (drop the target column)
y = df['DEATH_EVENT']  # target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
```

2.2. Perform Linear Regression using relevant clinical features to predict 'DEATH_EVENT.'

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
lin_pred = lin_reg.predict(X_test)
```

2.3. Implement Ridge Regression with the same features.

```
ridge_reg = Ridge(alpha=1.0)  # You can adjust the alpha value
ridge_reg.fit(X_train, y_train)
ridge_pred = ridge_reg.predict(X_test)
```

2.4. Evaluate the performance of both models using appropriate regression metrics (e.g., Mean Squared Error).

```
lin_mse = mean_squared_error(y_test, lin_pred)
ridge_mse = mean_squared_error(y_test, ridge_pred)

print(f"Linear Regression MSE: {lin_mse}")
print(f"Ridge Regression MSE: {ridge_mse}")
```

```
Linear Regression MSE: 0.18135130426271548
Ridge Regression MSE: 0.18095449818797266
```

2.5. Discuss the strengths and weaknesses of the Linear and Ridge Regression models for mortality prediction.

Linear Regression:

MSE: 0.18135130426271548

Strengths:

Competitive Performance: Despite its simplicity, the Linear Regression model achieved an MSE of 0.18135. This indicates that the model's predictions were relatively close to the actual values in the test set. Interpretability: The straightforward nature of Linear Regression makes it easy to understand and explain the relationship between predictors and the outcome, which is particularly valuable in clinical contexts.

Weaknesses:

Potential Overfitting: While the MSE was competitive, without regularization, Linear Regression is prone to overfitting, especially with many features or multicollinearity. It's worth noting that Ridge Regression, which includes regularization, achieved a slightly better MSE.

Ridge Regression:

MSE: 0.18095449818797266

Strengths:

Improved Performance: The Ridge Regression model provided a slightly better MSE of 0.18095 compared to Linear Regression. This improvement, albeit marginal, suggests that introducing regularization helped in optimizing the model's performance. Regularization: Ridge Regression's inherent ability to handle multicollinearity through its L2 penalty is evident in its performance. The slight edge in MSE over Linear Regression might be attributed to this regularization.

Weaknesses:

Parameter Tuning: Achieving optimal performance with Ridge Regression requires tuning the regularization parameter. While the improvement in MSE was slight, it's possible that different regularization strengths might yield varied results. Reduced Interpretability: Due to the regularization, Ridge Regression can be slightly harder to interpret than Linear Regression. While it offers benefits in terms of handling overfitting, the trade-off comes in the form of reduced transparency.

Task 3: Classification Analysis

3.1. Define a binary classification task to predict 'DEATH_EVENT.'

3.2. Apply Bayesian Classifier and Naive Bayes to predict 'DEATH_EVENT.'

```
# Naive Bayes Classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)

print(f"Naive Bayes Accuracy: {nb_accuracy:.4f}")
```

```
Naive Bayes Accuracy: 0.7778
```

3.3. Implement k-Nearest Neighbors (KNN) classification with the same target variable.

```
# KNN Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # Adjust the number of neig
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)

print(f"KNN Accuracy: {knn_accuracy:.4f}")
```

```
KNN Accuracy: 0.5556
```

3.4. Evaluate the performance of each classification model using metrics such as accuracy.

3.5. Discuss the suitability of each classification method for identifying patients at risk.

Naive Bayes Classifier (Accuracy: 77.78%):

Suitability: Given its accuracy of approximately 77.78%, the Naive Bayes classifier seems to be a decent predictor for this dataset. Its simplicity and efficiency make it a good first choice for classification tasks.

k-Nearest Neighbors (KNN) Classifier (Accuracy: 55.56%):

Suitability: With an accuracy of 55.56%, KNN seems to be underperforming on this dataset. The lower accuracy might be due to reasons like not scaling the features, inappropriate value of 'k', or the dataset might have many irrelevant features that are affecting the KNN's decision-making.

Naive Bayes proves to be a more effective method, providing a reasonably high accuracy. Given its performance, it might be more suited for preliminary risk identification in clinical settings.

KNN, on the other hand, may not be the best choice with its current configuration. If one wishes to further use KNN, it would be advisable to experiment with different values of 'k', consider feature scaling, and possibly perform feature selection or extraction for better results.