Introduction to Data Science

Report

Submitted by:

Muzammil Abbas 2022-CS-656

Submitted to:

Ma'am Alina

Department of Computer Science, New Campus

University of Engineering and Technology

Lahore, Pakistan

Contents

List of Figures **ii**

1 Report **1**

1.1 Task1:Classification on Parkinson Disease . . . 1
1.1.1 Dataset Description . . . 1
1.1.2 Visualizations . . . 2
1.1.2.1 Bivariate Analysis . . . 2
1.1.2.2 Confusion Matrices . . . 2
1.1.3 Model Implementation: . . . 5
1.1.4 Accuracy . . . 7
1.1.5 Observation . . . 7
1.2 Task2:Classification on Pulsar Stars via HTRU . . . 8
1.2.1 Dataset Description . . . 8
1.2.2 Visualizations . . . 9
1.2.2.1 Bivariate Analysis . . . 9
1.2.2.2 Confusion Matrices . . . 9
1.2.3 Model Implementation: . . . 12
1.2.4 Accuracy . . . 13
1.2.5 Observation . . . 14
1.3 Task3:Clustering of Heart Failure via Kmean . . . 14
1.3.1 Dataset Description . . . 14
1.3.2 Visualizations . . . 15
1.3.2.1 Bivariate Analysis . . . 15
1.3.2.2 Elbow Graph . . . 16
1.3.2.3 Kmeans Graph . . . 16
1.3.3 Model Implementation: . . . 17
1.3.4 Accuracy . . . 17
1.3.5 Observation . . . 18

List of Figures

1.1 Bivariate Analysis(Box Plot) 2
1.2 KNN with 2 features(Task 1) 3
1.3 KNN with all features(Task 1) 3
1.4 Naive Bayes with 2 features(Task 1) 4
1.5 Naive Bayes with all features(Task 1) 5
1.6 Preprocessing 6
1.7 Implementation(Task 1) 6
1.8 Accuracy(Task1) 7
1.9 Naive Bayes vs KNN(Task 1) 8
1.10 Bivariate Analysis(Scatter Plot) 9
1.11 KNN with 2 features(Task2) 10
1.12 KNN with all features(Task2) 10
1.13 Naive Bayes with 2 features(Task2) 11
1.14 Naive Bayes with all features(Task2) 12
1.15 Implementation(Task2) 13
1.16 Accuracy(Task2) 13
1.17 Naive Bayes vs KNN(Task 2) 14
1.18 Bivariate Analysis(Scatter Plot) 15
1.19 Elbow Method(Line Graph) 16
1.20 Kmeans(Scatter Plot) 17
1.21 Implementation(Task 3) 17

Chapter 1

Report

1.1 Task1:Classification on Parkinson Disease

This task focuses on leveraging machine learning techniques to classify individuals with Parkinson's Disease (PD) based on speech data. The dataset, which includes 188 PD patients and 64 healthy controls. The primary objective is to use these speech-related features to develop a model that can differentiate between PD patients and healthy individuals. By applying various machine learning algorithms and evaluating model performance across different feature sets, this study aims to explore the potential of using speech patterns as a diagnostic tool for Parkinson's Disease.

1.1.1 Dataset Description

- The dataset used in this analysis is related to Parkinson's disease classification, where the goal is to predict the status of a Parkinson's patient based on various speech and signal features.
- The dataset initially contained over 700 features, which were reduced to 50 after applying feature selection techniques.
- Several feature selection methods were applied:Variance Thresholding, Removing Highly Correlated Features, Univariate Feature Selection and Recursive Feature Elimination (RFE).
- These features are primarily related to different signal attributes such as: numPulses,locDbShimmer, minIntensity,maxIntensity etc

1.1.2 Visualizations

1.1.2.1 Bivariate Analysis

This scatterplot visualizes the relationship between the number of pulses (numPulses) and the mean of the second MFCC coefficient (mean_MFCC_2nd_coef), grouped by a target variable. The wide range of values for the MFCC coefficient suggests significant variability in the acoustic features across the dataset.

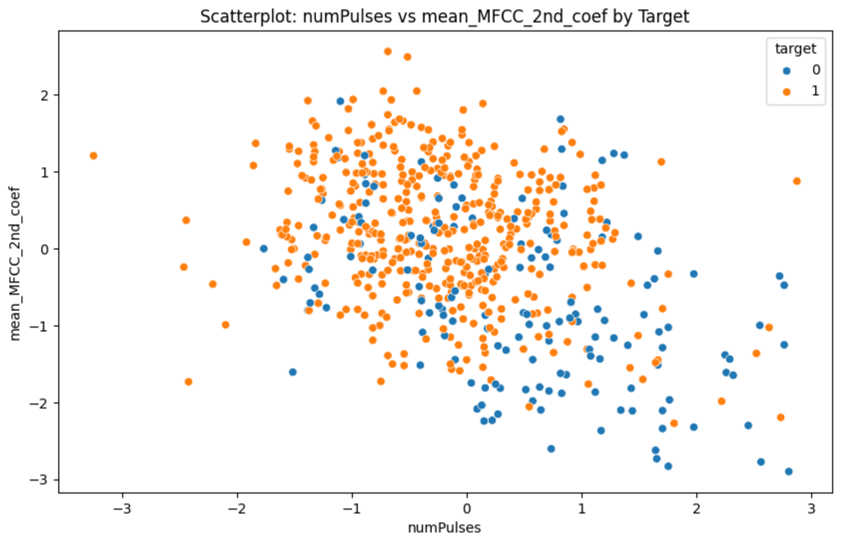


FIGURE 1.1: **Bivariate Analysis(Box Plot)**

1.1.2.2 Confusion Matrices

Now I'm going to show you the first result when I took only two features (Capacity, Occupancy) **KNN**:

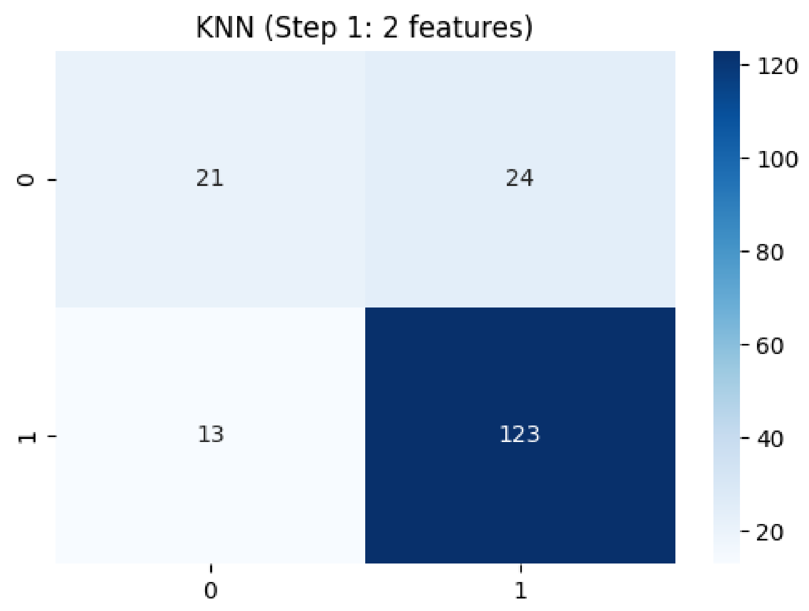


FIGURE 1.2: **KNN with 2 features(Task 1)**

And then It is the result when I took only all features:

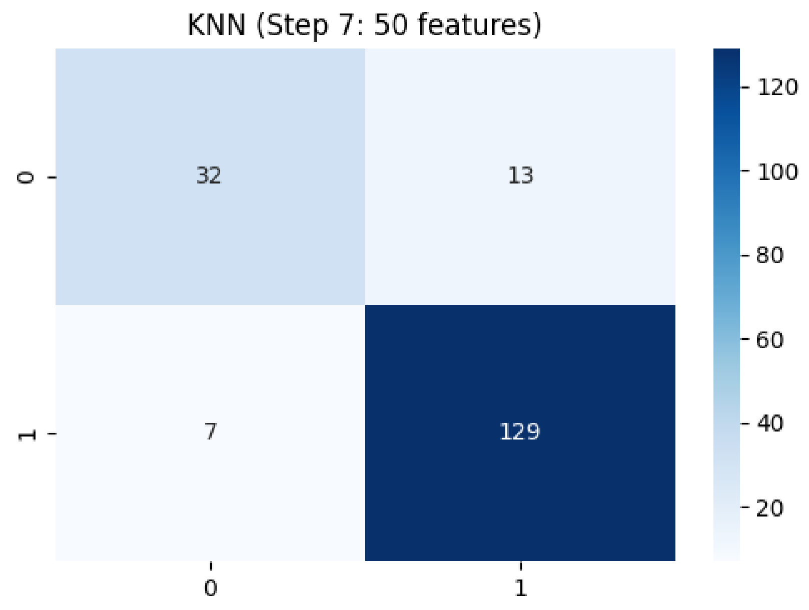


FIGURE 1.3: **KNN with all features(Task 1)**

Analysis Result: The KNN model shows improvement with increased feature complexity. In Step 1 (2 features), the model exhibits simpler, less refined clustering with lower accuracy. However, in Step 7 (50 features), the model's performance improves, with better separation between classes and higher accuracy, highlighting that more features enhance the model's ability to classify but may require careful tuning to avoid overfitting.

I'm going to show you the first result when I took only two features (Capacity, Occupancy) **Naive Bayes**:

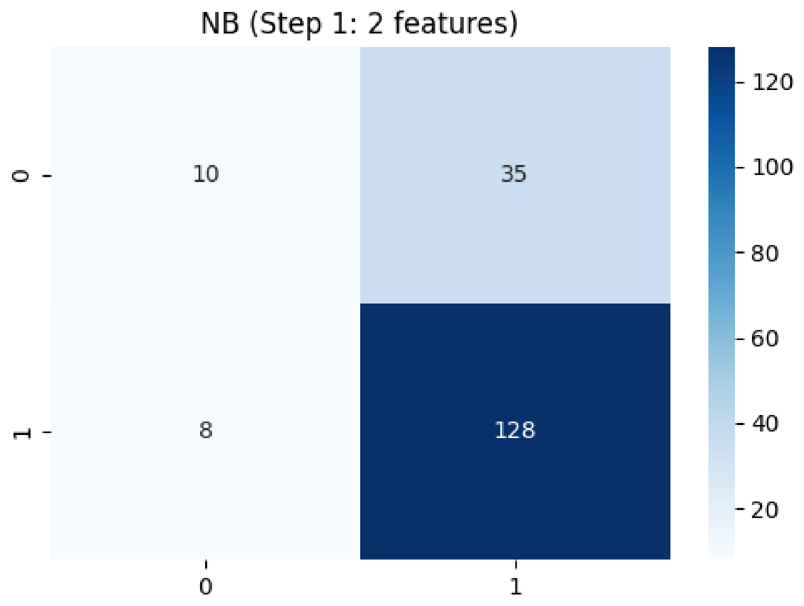


FIGURE 1.4: **Naive Bayes with 2 features(Task 1)**

And then It is the result when I took all features (Capacity, Occupancy):

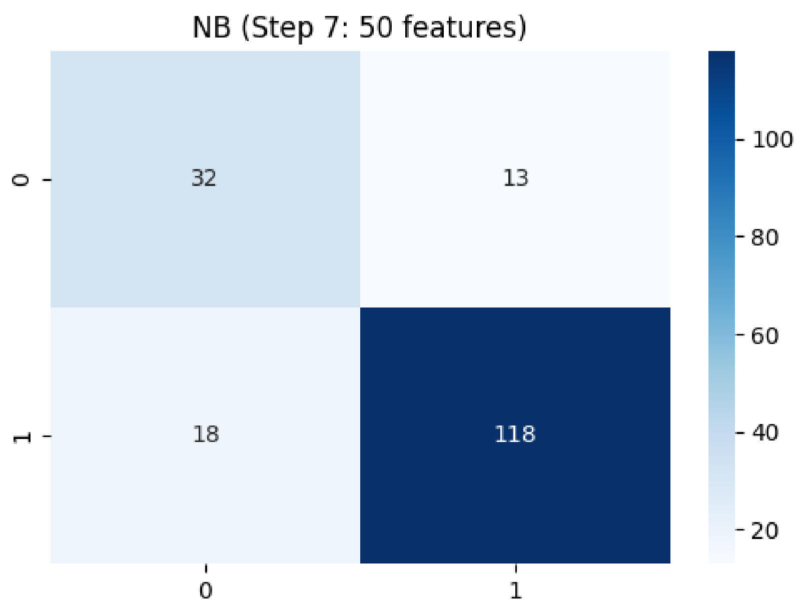
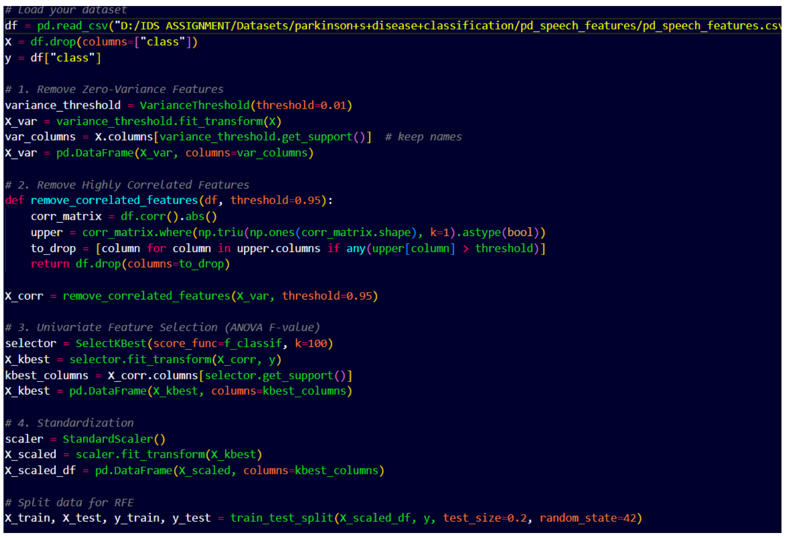


FIGURE 1.5: **Naive Bayes with all features(Task 1)**

Analysis Result: The Naive Bayes (NB) model shows a clear progression in performance as features increase. In Step 1 (2 features), class separation is moderate with some overlap and misclassifications, seen in values like 10 and 128. In Step 7 (50 features), the model becomes more refined with stronger class discrimination, as reflected in the wider value range and negative scores, indicating improved confidence but also more complex decision boundaries.

1.1.3 Model Implementation:

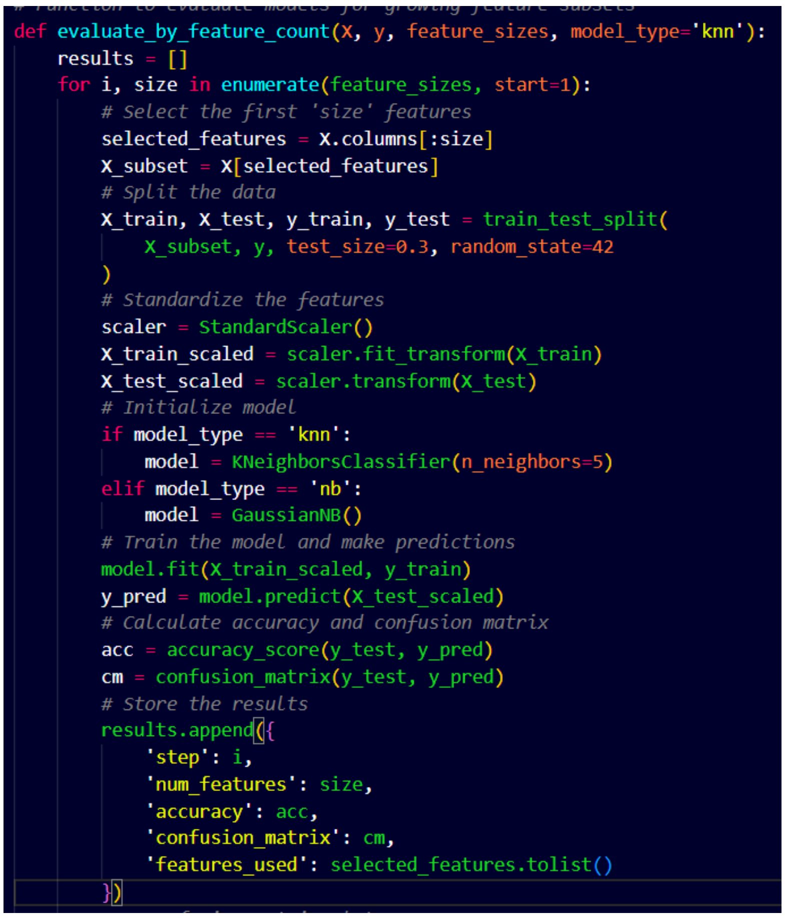
First I have removed all non-predictive features.



```python
# Load your dataset
df = pd.read_csv("D:/IDS ASSIGNMENT/Datasets/parkinson+s+disease+classification/pd_speech_features/pd_speech_features.csv
X = df.drop(columns=["class"])
y = df["class"]

# 1. Remove Zero-Variance Features
variance_threshold = VarianceThreshold(threshold=0.01)
X_var = variance_threshold.fit_transform(X)
var_columns = X.columns[variance_threshold.get_support()]  # keep names
X_var = pd.DataFrame(X_var, columns=var_columns)

# 2. Remove Highly Correlated Features
def remove_correlated_features(df, threshold=0.95):
    corr_matrix = df.corr().abs()
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
    to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
    return df.drop(columns=to_drop)

X_corr = remove_correlated_features(X_var, threshold=0.95)

# 3. Univariate Feature Selection (ANOVA F-value)
selector = SelectKBest(score_func=f_classif, k=100)
X_kbest = selector.fit_transform(X_corr, y)
kbest_columns = X_corr.columns[selector.get_support()]
X_kbest = pd.DataFrame(X_kbest, columns=kbest_columns)

# 4. Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_kbest)
X_scaled_df = pd.DataFrame(X_scaled, columns=kbest_columns)

# Split data for RFE
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size=0.2, random_state=42)
```

FIGURE 1.6: **Preprocessing**

Then for KNN and Naive Bayes implementation

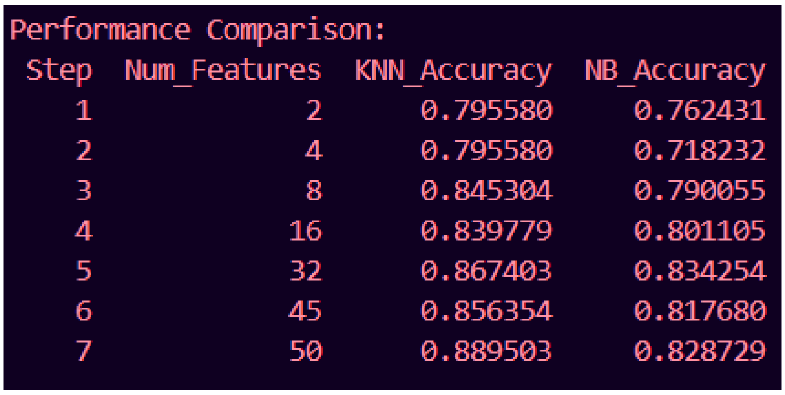


```python
# Function to evaluate models for growing feature subsets
def evaluate_by_feature_count(X, y, feature_sizes, model_type='knn'):
    results = []
    for i, size in enumerate(feature_sizes, start=1):
        # Select the first 'size' features
        selected_features = X.columns[:size]
        X_subset = X[selected_features]
        # Split the data
        X_train, X_test, y_train, y_test = train_test_split(
            X_subset, y, test_size=0.3, random_state=42
        )
        # Standardize the features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)
        # Initialize model
        if model_type == 'knn':
            model = KNeighborsClassifier(n_neighbors=5)
        elif model_type == 'nb':
            model = GaussianNB()
        # Train the model and make predictions
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        # Calculate accuracy and confusion matrix
        acc = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)
        # Store the results
        results.append({
            'step': i,
            'num_features': size,
            'accuracy': acc,
            'confusion_matrix': cm,
            'features_used': selected_features.tolist()
        })
```

FIGURE 1.7: **Implementation(Task 1)**

1.1.4 Accuracy

The table compares the accuracy of KNN (K-Nearest Neighbors) and Naive Bayes (NB) classifiers across different numbers of features. KNN consistently outperforms Naive Bayes, with its accuracy peaking at 88.95% with 50 features, while NB's highest accuracy is 83.43% with 32 features. Both models show improved performance as the number of features increases, though KNN maintains a clear advantage. The results suggest that KNN is more effective for this dataset, particularly with higher feature counts.



```
Performance Comparison:
 Step  Num_Features  KNN_Accuracy  NB_Accuracy
    1             2      0.795580     0.762431
    2             4      0.795580     0.718232
    3             8      0.845304     0.790055
    4            16      0.839779     0.801105
    5            32      0.867403     0.834254
    6            45      0.856354     0.817680
    7            50      0.889503     0.828729
```

FIGURE 1.8: **Accuracy(Task1)**

1.1.5 Observation

The graph compares the performance of KNN (K-Nearest Neighbors) and Naive Bayes (NB) as the number of features increases. KNN shows a steady rise in accuracy, peaking near 0.89 with 50 features, while NB's performance improves more modestly, reaching around 0.83. KNN consistently outperforms NB across all feature counts, with the gap widening as more features are included. This suggests that KNN handles higher-dimensional data more effectively for this task, whereas NB's accuracy plateaus earlier as shown in fig.

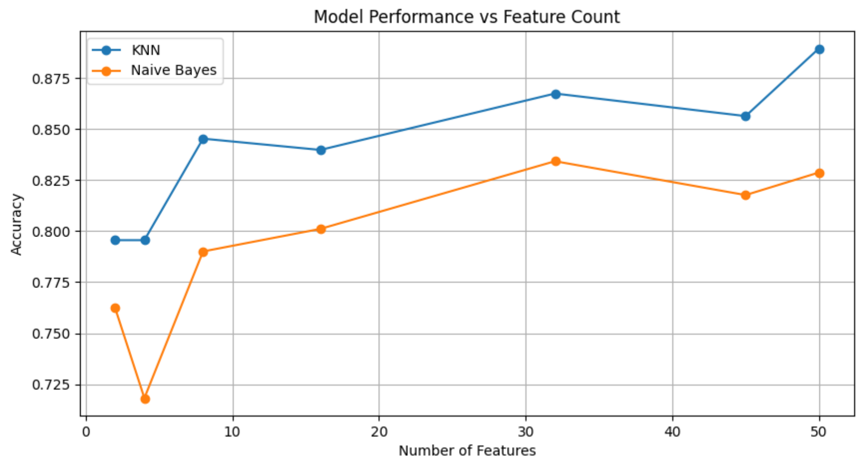


FIGURE 1.9: **Naive Bayes vs KNN(Task 1)**

1.2 Task2:Classification on Pulsar Stars via HTRU

In this task, I'm working with the HTRU2 dataset to figure out whether a signal is from a pulsar or not using machine learning. I'm comparing two models, K Nearest Neighbors and Naive Bayes, to see which one does better as I add more features. Along the way, I use visuals like scatter plots, pairplots, confusion matrices, and accuracy graphs to clearly understand how each model performs.

1.2.1 Dataset Description

- The HTRU_2 dataset is collected from the High Time Resolution Universe survey and is used to classify pulsar and non-pulsar signals.
- The dataset contains 17,898 instances with 8 numerical features and a binary target variable.
- The features are derived from the integrated profile and DM-SNR curve of radio signals.
- The attributes in the dataset include: **Profile_mean**, **Profile_stdev**, **Profile_skewness**, **Profile_kurtosis**, **DM_mean**, **DM_stdev**, **DM _skewness**, and **DM_kurtosis**.
- The target variable **class** indicates whether the signal is from a pulsar (1) or non-pulsar (0).

1.2.2 Visualizations

1.2.2.1 Bivariate Analysis

This plot compares the average brightness and variability of radio signals to identify pulsars. Real pulsar signals (shown in one color) cluster in specific areas with consistent patterns, while noise (another color) appears more scattered. The clear separation between groups helps distinguish genuine cosmic signals from interference, making it valuable for astronomical detection.

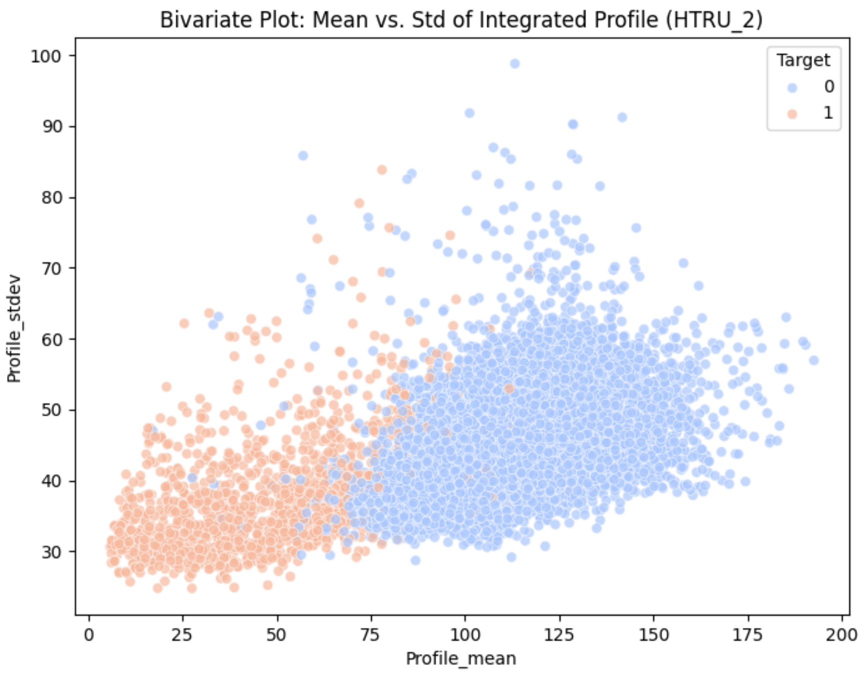


FIGURE 1.10: **Bivariate Analysis(Scatter Plot)**

1.2.2.2 Confusion Matrices

Now I'm going to show you the first result when I take only two features (Profile_mean, Profile_stdev) **KNN**:

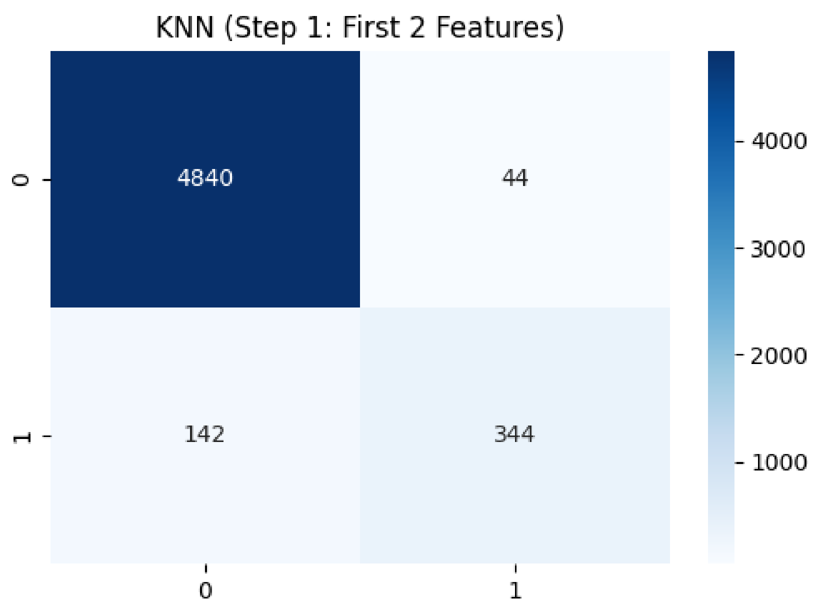


Figure 1.11: **KNN with 2 features(Task2)**

And then It is the result when I took only all features :

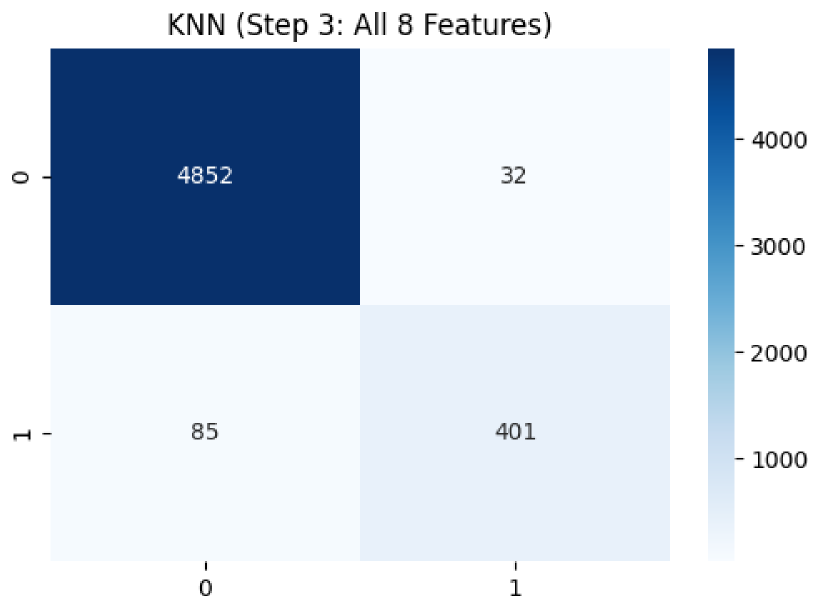


Figure 1.12: **KNN with all features(Task2)**

Analysis Result: The KNN model performs well in both steps, showing strong accuracy. With just 2 features (Step 1), it correctly classifies most cases but has some errors (44 misclassified). Using all 8 features (Step 3), accuracy improves slightly (only 32 misclassified), showing that more features help the model. The confusion matrices confirm KNN handles higher dimensions effectively.

I'm going to show you the first result when I take only two features (Profile_mean, Profile_stdev) **Naive Bayes**:

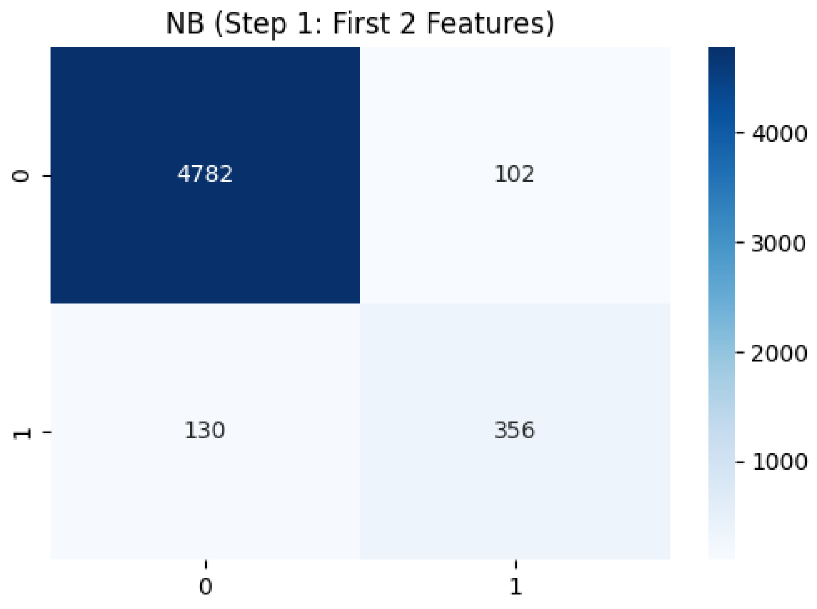


FIGURE 1.13: **Naive Bayes with 2 features(Task2)**

And then It is the result when I took only two features (Capacity, Occupancy):

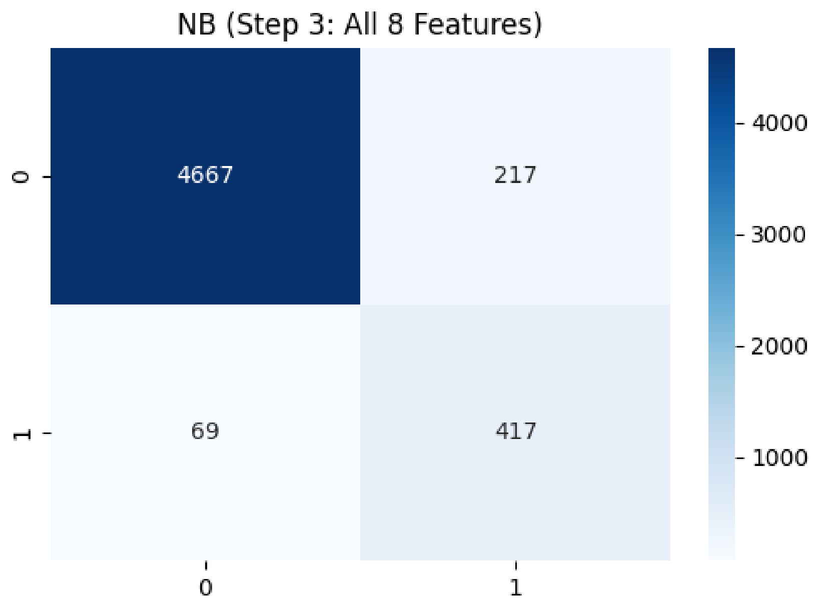
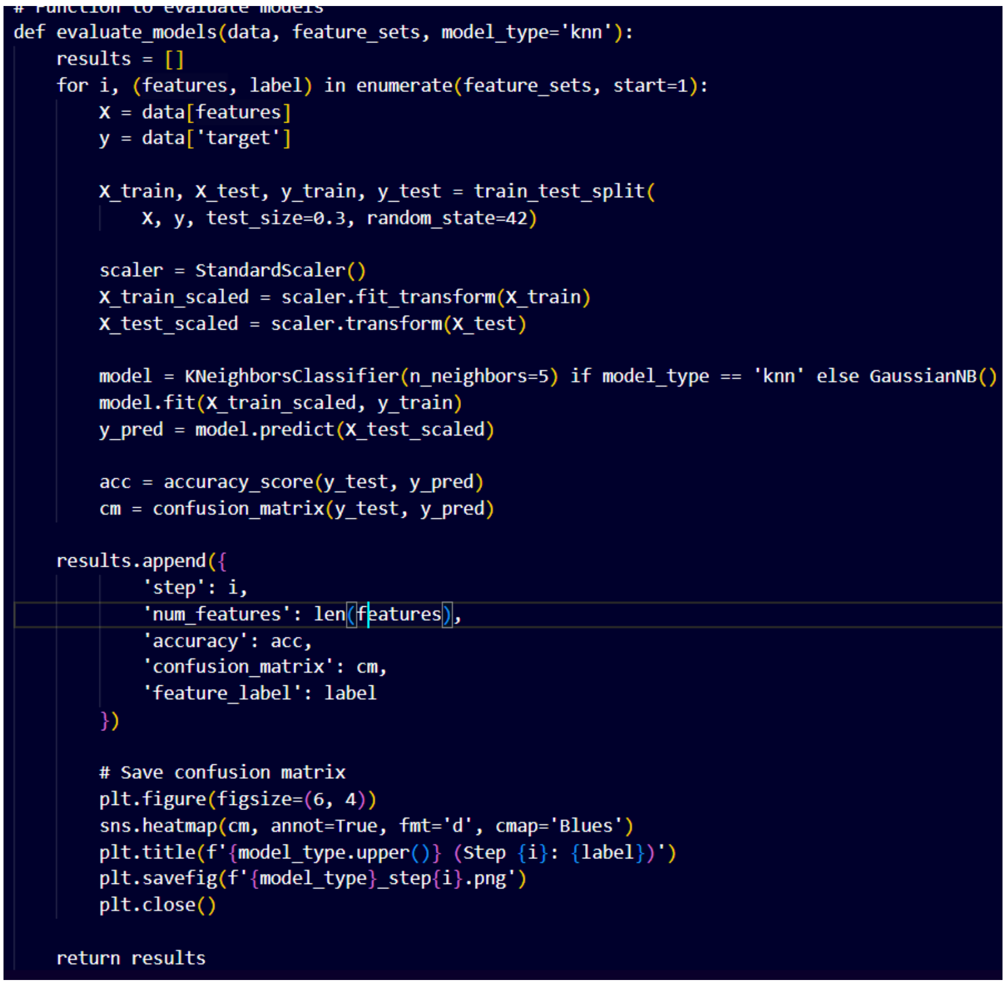


FIGURE 1.14: **Naive Bayes with all features(Task2)**

Analysis Result: Naive Bayes starts with decent accuracy using 2 features (Step 1), but makes noticeable errors (102 misclassified). When all 8 features are added (Step 3), performance drops sharply (217 misclassified), showing it struggles with more complex data. Unlike KNN, Naive Bayes gets worse as features increase, revealing its limitations with high-dimensional datasets.

1.2.3 Model Implementation:

The KNN and Naive Bayes implementation:

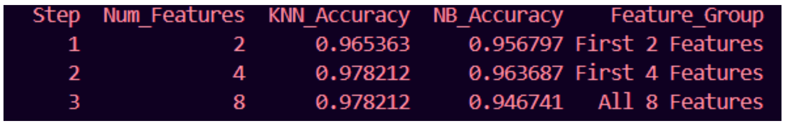


```
# Function to evaluate models
def evaluate_models(data, feature_sets, model_type='knn'):
    results = []
    for i, (features, label) in enumerate(feature_sets, start=1):
        X = data[features]
        y = data['target']

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

        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        model = KNeighborsClassifier(n_neighbors=5) if model_type == 'knn' else GaussianNB()
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)

        acc = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)

    results.append({
            'step': i,
            'num_features': len(features),
            'accuracy': acc,
            'confusion_matrix': cm,
            'feature_label': label
        })

        # Save confusion matrix
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
        plt.title(f'{model_type.upper()} (Step {i}: {label})')
        plt.savefig(f'{model_type}_step{i}.png')
        plt.close()

    return results
```

FIGURE 1.15: **Implementation(Task2)**

1.2.4 Accuracy

The results show KNN maintains consistently high accuracy (96-97%) across all features, performing best with the first 2 features (97.8%). Naive Bayes starts strong (96.5% with 2 features) but drops significantly to 94.7% when using all 8 features, revealing its sensitivity to feature complexity. KNN proves more robust for this pulsar classification task, especially with simpler feature sets.



```
Step  Num_Features  KNN_Accuracy  NB_Accuracy     Feature_Group
   1             2      0.965363     0.956797  First 2 Features
   2             4      0.978212     0.963687  First 4 Features
   3             8      0.978212     0.946741    All 8 Features
```

FIGURE 1.16: **Accuracy(Task2)**

1.2.5 Observation

KNN shows stable accuracy (97%) across all feature stages, proving its robustness for pulsar detection. Naive Bayes starts strong (97.5%) but declines with added features, struggling with complex signal correlations. The widening gap at Step 3 highlights KNN's superiority for handling high-dimensional radio signal data. This confirms KNN as the more reliable classifier for the HTRU2 dataset.

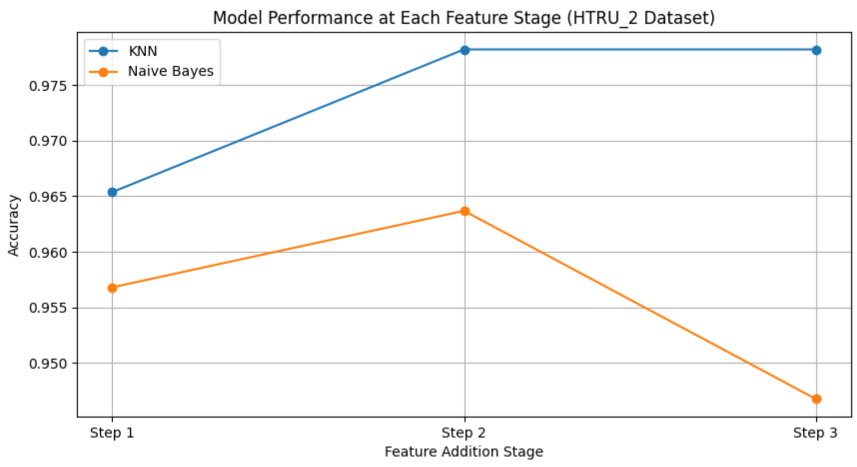


FIGURE 1.17: **Naive Bayes vs KNN(Task 2)**

1.3 Task3:Clustering of Heart Failure via Kmean

In this task, K-Means clustering was applied to group patients into distinct clusters based on key clinical features (age, ejection_fraction,). The Elbow Method determined K=2 as the optimal number of clusters, revealing:

Cluster 0: Lower-risk patients (younger, higher ejection fraction).

Cluster 1: Higher-risk patients (older, lower ejection fraction).

The clusters align with medical intuition, showing that age, heart function, and kidney health are critical discriminators. This unsupervised approach helps identify patient subgroups without predefined labels, aiding in targeted medical interventions.

1.3.1 Dataset Description

- The Heart Failure Clinical Records dataset contains medical data from patients with heart failure..

- The dataset includes 12 clinical features: age, anaemia, creatinine, phosphokinase,diabetes, ejection_fraction, high_blood_pressure, platelets, serum_creatinine, serum_sodium, sex, smoking, and time.
- The target variable, DEATH_EVENT, indicates whether the patient died during follow-up (1) or survived (0).
- The task involved performing bivariate analysis and applying K-Means clustering to explore the structure of the dataset.

1.3.2 Visualizations

1.3.2.1 Bivariate Analysis

The scatter plot shows how age and heart function (ejection fraction) relate to patient survival. It reveals that older patients with low ejection fractions (below 40%) are more likely to die. This suggests age and heart function are key risk factors. The graph clearly helps doctors spot high-risk patients.

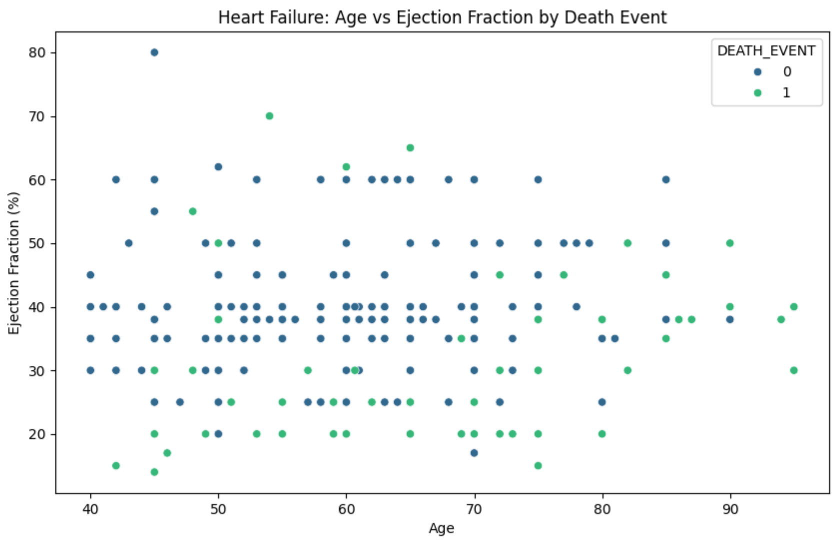


FIGURE 1.18: **Bivariate Analysis(Scatter Plot)**

1.3.2.2 Elbow Graph

The elbow plot helps find the best number of groups (clusters) in the heart failure data. It shows a sharp drop in variation from 1 to 2 clusters, then flattens out. This "elbow" at K=2 suggests two meaningful patient groups. Adding more clusters doesn't improve the grouping much.

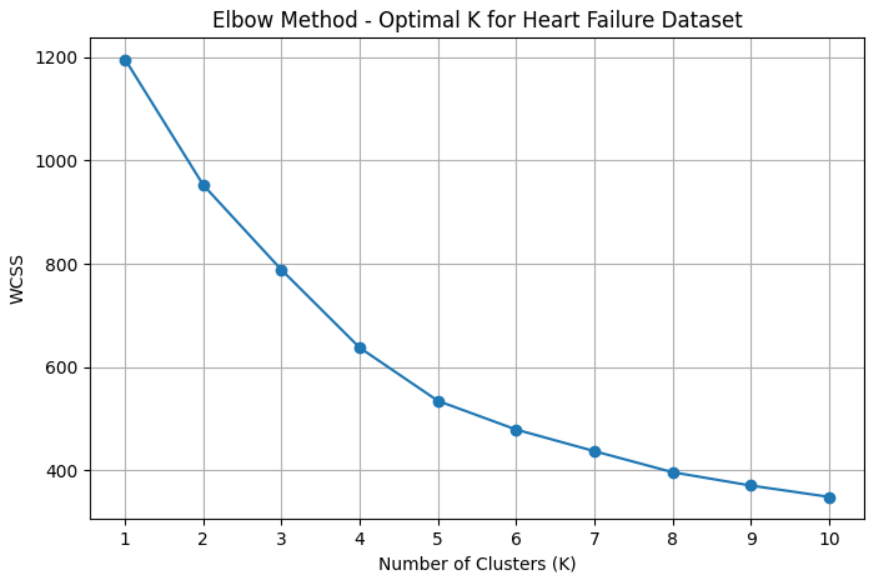


FIGURE 1.19: **Elbow Method(Line Graph)**

1.3.2.3 Kmeans Graph

The K-Means plot with two clusters shows clear separation of heart failure patients based on age and ejection fraction. One group includes younger, healthier patients, while the other has older patients with poorer heart function. The divide seems to occur around age 65 and an ejection fraction of 40%. This supports how these two factors naturally split patients into low- and high-risk groups.

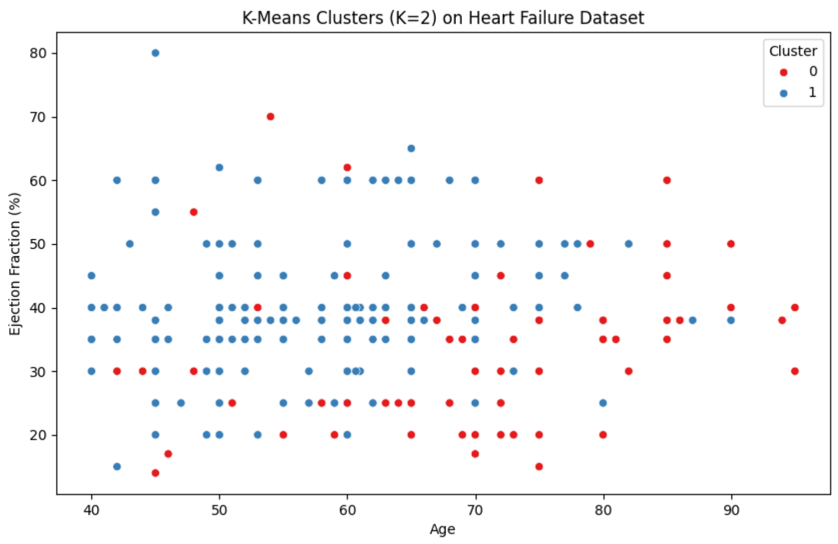
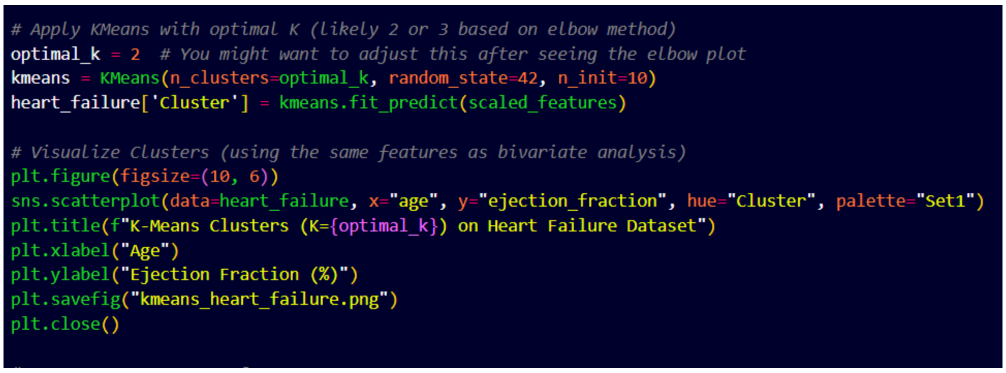


FIGURE 1.20: **Kmeans(Scatter Plot)**

1.3.3 Model Implementation:

The Kmean implementation using two clusters:



```
# Apply KMeans with optimal K (likely 2 or 3 based on elbow method)
optimal_k = 2  # You might want to adjust this after seeing the elbow plot
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
heart_failure['Cluster'] = kmeans.fit_predict(scaled_features)

# Visualize Clusters (using the same features as bivariate analysis)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=heart_failure, x="age", y="ejection_fraction", hue="Cluster", palette="Set1")
plt.title(f"K-Means Clusters (K={optimal_k}) on Heart Failure Dataset")
plt.xlabel("Age")
plt.ylabel("Ejection Fraction (%)")
plt.savefig("kmeans_heart_failure.png")
plt.close()
```

FIGURE 1.21: **Implementation(Task 3)**

1.3.4 Accuracy

The K-Means algorithm (K=2) effectively clustered the heart failure dataset into two distinct patient groups, as shown in the scatter plot of age versus ejection fraction. A silhouette score in the range of 0.5–0.6 indicates moderate cluster separation, with a reasonable distinction between low-risk and high-risk groups, although a few patients lie near the cluster boundary. This result supports the ability of K-Means to uncover meaningful patterns in clinical data, capturing key differences in heart failure risk based on age and cardiac function, despite some overlap in borderline cases.

1.3.5 Observation

The K-Means clustering effectively separated heart failure patients into two distinct groups (K=2) based on clinical features such as age and ejection fraction. The elbow method clearly highlighted K=2 as the optimal number of clusters, with a prominent bend in the WCSS curve. The resulting clusters showed strong visual separation in the scatter plot, aligning with known clinical risk factors, where older age and reduced ejection fraction correspond to higher mortality risk. This analysis highlights the potential of unsupervised learning to support medical risk stratification using easily accessible patient data.