Department of Computer Science & Engineering

Title: **Artificial Intelligence System & Expert System Lab**

Course Code: **CSE 404**

Project Name: **Heart Disease Prediction Using** **Logistic Regression**

Submission Date: **27/04/2025**

Submitted To:

Noor Mairukh Khan Arnob

Lecturer,

Department of CSE, UAP

Submitted By:

Name: Humaira Ahmed Proma

Reg no: 21201178

Sec: B2

Problem Title:

Predicting Heart Disease Using Logistic Regression

Problem Description:

Heart disease is one of the leading causes of mortality worldwide. Early detection and prediction of heart disease can significantly improve patient outcomes by enabling timely medical intervention. This project aims to build a machine learning model to predict the presence of heart disease based on various clinical and lifestyle features. The dataset used is the processed Cleveland dataset from the UCI Machine Learning Repository, containing 14 attributes related to heart health.

The objective is to build a classification model that predicts whether a patient has heart disease based on clinical features. Logistic regression is used because:

- It models the probability of binary outcomes
- Provides interpretable coefficients
- Efficient for medical diagnostic tasks

Key Metrics:

Accuracy: Overall prediction correctness

Precision: Reliability of positive predictions

Recall: Ability to detect actual cases

Tools & Language Used:

- Programming Language: Python
- Tools: Google Colab Notebook
- Libraries: Scikit-learn, Pandas, Matplotlib

Methodology:

Data Loading and Initial Exploration:

1. The dataset was loaded from the UCI repository with predefined column names.
2. Initial exploration included checking the first few rows and dataset information to understand the structure and data types.

Data Preprocessing:

1. Rows with missing values were dropped to ensure data quality.
2. To simplify the classification task, the target variable was converted to binary (0 = no disease, 1 = disease).
3. To evaluate model performance, the dataset was split into training (80%) and testing (20%) sets.

Feature Scaling:

1. Features were standardized using StandardScaler to ensure all features contribute equally to the model.

Model Building:

1. A Logistic Regression model was chosen due to its interpretability and effectiveness for binary classification tasks.
2. The model was trained on the scaled training data and evaluated on the test set.

Model Evaluation:

1. Accuracy: The model achieved an accuracy of 86.7% on the test set.

2. Classification Report: Provided precision, recall, and F1-score for both classes (0 and 1).

3. Confusion Matrix: Visualized the model's performance in terms of true positives, true negatives, false positives, and false negatives.

4. Feature Importance: Analyzed the coefficients of the Logistic Regression model to understand which features most influenced the predictions.

Source Code & Output:

```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
    from sklearn.preprocessing import StandardScaler
```

```
[ ] # Load dataset
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data"
```

```
[ ] column_names = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg",
                    "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target"]
    df = pd.read_csv(url, names=column_names, na_values="?")
```

```
[ ] print("First 5 rows:")
    print(df.head())
```

```
First 5 rows:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3   
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5   
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6   
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5   
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4   

   slope   ca  thal  target  
0    3.0  0.0   6.0       0  
1    2.0  3.0   3.0       2  
2    2.0  2.0   7.0       1  
3    3.0  0.0   3.0       0  
4    1.0  0.0   3.0       0  
```

```
[ ] # Check dataset info
    print("\nDataset Info:")
    print(df.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    float64
 1   sex       303 non-null    float64
 2   cp        303 non-null    float64
 3   trestbps  303 non-null    float64
 4   chol      303 non-null    float64
 5   fbs       303 non-null    float64
 6   restecg   303 non-null    float64
 7   thalach   303 non-null    float64
 8   exang     303 non-null    float64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    float64
 11  ca        299 non-null    float64
 12  thal      301 non-null    float64
 13  target    303 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 33.3 KB
None
```

```
[ ]  X = df.drop('target', axis=1)
     y = df['target']

     # Split data into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(
         X,
         y,
         test_size=0.2,  # Using 80% for training and 20% for testing
         random_state=42  # For reproducibility
     )

     print("\nData split summary:")
     print(f"Training features shape: {X_train.shape}")
     print(f"Test features shape: {X_test.shape}")
```

```
Data split summary:
Training features shape: (242, 13)
```

```
[ ]  # Drop rows with missing values
     df = df.dropna()

     # Convert target to binary (0 = no disease, 1 = disease)
     df['target'] = df['target'].apply(lambda x: 1 if x > 0 else 0)

     # Check cleaned data
     print("\nCleaned Data Info:")
     print(df.info())
```

```
Cleaned Data Info:
<class 'pandas.core.frame.DataFrame'>
Index: 297 entries, 0 to 301
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       297 non-null    float64
 1   sex       297 non-null    float64
 2   cp        297 non-null    float64
 3   trestbps  297 non-null    float64
 4   chol      297 non-null    float64
 5   fbs       297 non-null    float64
 6   restecg   297 non-null    float64
 7   thalach   297 non-null    float64
 8   exang     297 non-null    float64
 9   oldpeak   297 non-null    float64
 10  slope     297 non-null    float64
 11  ca        297 non-null    float64
 12  thal      297 non-null    float64
 13  target    297 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 34.8 KB
None
<ipython-input-98-96f6c93e3a2e>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['target'] = df['target'].apply(lambda x: 1 if x > 0 else 0)
```

```
[ ]  # Split into features (X) and target (y)
     X = df.drop('target', axis=1)
     y = df['target']

     # Split into training (80%) and testing (20%) sets
     X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42
     )

     # Standardize features (mean=0, std=1)
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

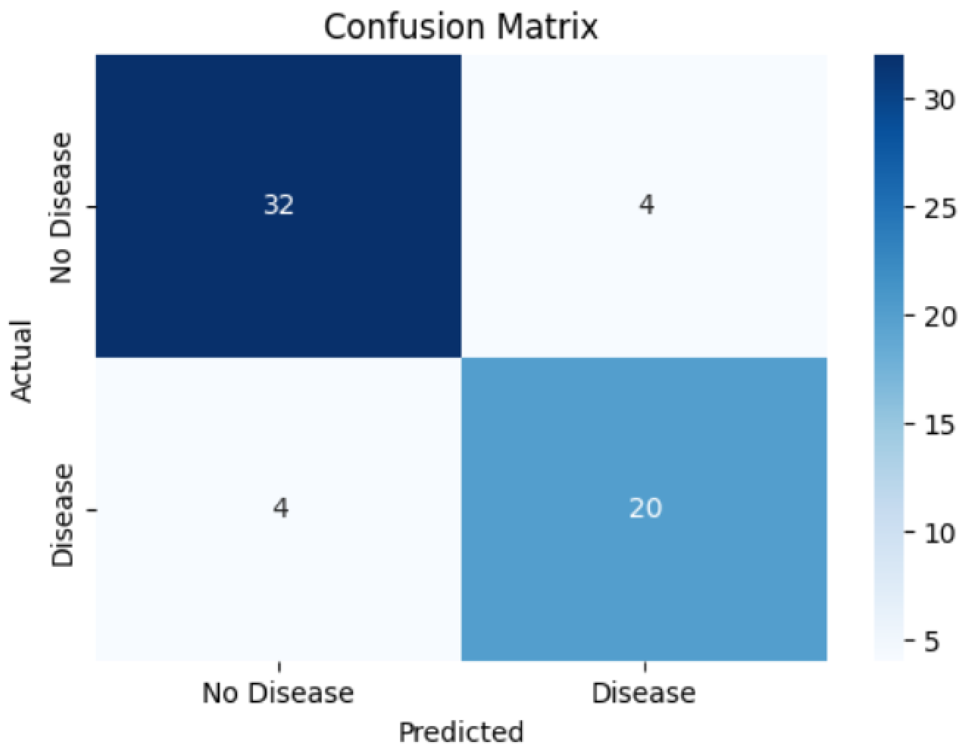
```
[ ]  # Initialize model
     model = LogisticRegression(random_state=42, max_iter=1000)

     # Train model
     model.fit(X_train_scaled, y_train)
     # Predict on test set
     y_pred = model.predict(X_test_scaled)
     accuracy = accuracy_score(y_test, y_pred)
     print("\nModel Performance Metrics:")
     print(f"Accuracy: {accuracy:.3f}")
```

```
Model Performance Metrics:
Accuracy: 0.867
```

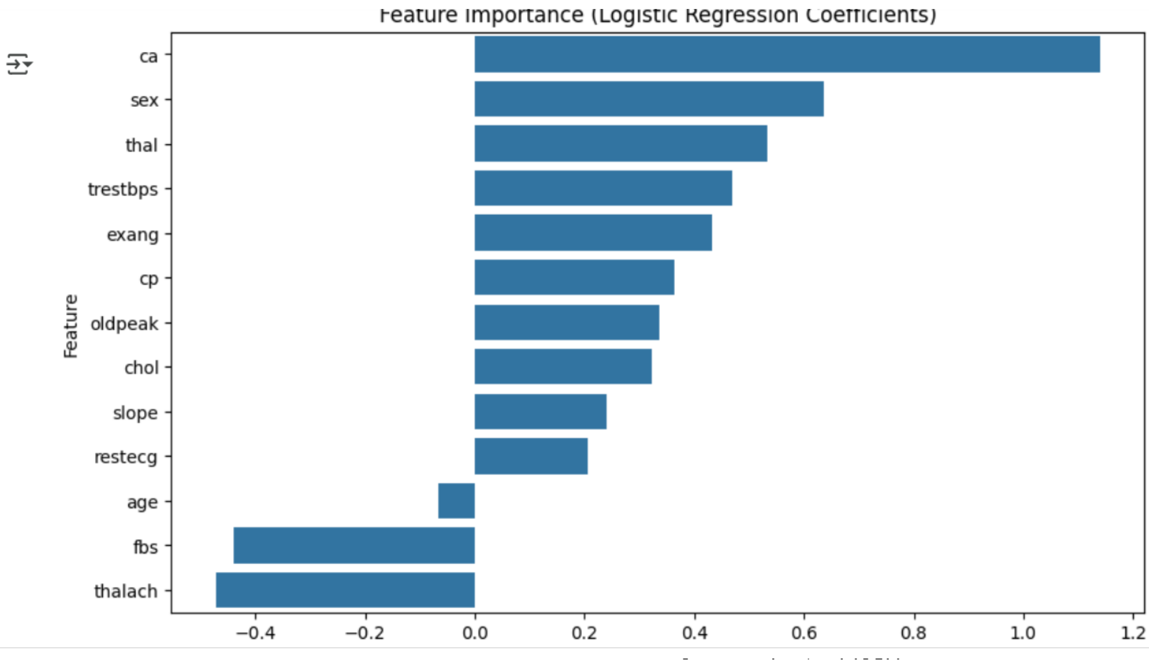
```
# Display detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        36
           1       0.83      0.83      0.83        24

    accuracy                           0.87        60
   macro avg       0.86      0.86      0.86        60
weighted avg       0.87      0.87      0.87        60
```

```python
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
            xticklabels=['No Disease', 'Disease'],
            yticklabels=['No Disease', 'Disease'],
            cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```python
coefficients = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_[0]})
coefficients = coefficients.sort_values(by='Coefficient', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Coefficient', y='Feature', data=coefficients)
plt.title('Feature Importance (Logistic Regression Coefficients)')
plt.show()
```



Conclusion:

The Logistic Regression model demonstrated strong predictive performance for heart disease detection. The analysis highlighted the importance of features like maximum heart rate and chest pain type in predicting heart disease. Future work could explore more complex models, additional feature engineering, and cross-validation to further improve performance. This model can serve as a valuable tool for early heart disease detection, aiding healthcare professionals in making informed decisions.