HEART DISEASE

Machine Learning project

MARYAM ALAAMRI
MARWA SAAR

Contents we will discuss

Introduction

Heart disease is a leading cause of death worldwide. Early detection through accurate prediction can improve patient outcomes. This project builds and evaluates machine learning classification models to predict heart disease using patient indicators like age, blood pressure, cholesterol, and exercise, focusing on data exploration, preprocessing, model training, and performance evaluation, apply different classification algorithms, and improve model performance while ensuring reliable evaluation.

Dataset description

The dataset used in this project is based on the UCI Heart Disease dataset, which contains clinical and demographic information related to cardiovascular health.

Age of the patient
Gender of the patient (1 = male, 0 = female)
Chest pain type (0,1,2,3)
Resting blood pressure
Serum cholesterol
Fasting blood sugar (>120 mg/dl: 1 = true, 0 = false)
Resting electrocardiographic results

Exercise-induced angina (1 = yes, 0 = no)
Maximum heart rate achieved
ST depression induced by exercise
Slope of the peak exercise ST segment
Number of major vessels colored by fluoroscopy
Thalassemia type (1,2,3)

Data Preprocessing

	Age	Sex	Chest_Pain_Type	Resting_Blood_Pressure	Serum_Cholesterol	Fasting_Blood_Sugar	Resting_ECG_Results	Max_Heart_Rate_Achieved
0	63	0	1	140	195	0	1	179
1	46	1	0	120	249	0	0	144
2	69	1	3	160	234	1	0	131
3	51	1	2	94	227	0	1	154
4	50	1	2	140	233	0	1	163
...	...	...	...	...	...	...	...	...
1995	40	1	3	140	199	0	1	178
1996	52	1	1	128	205	1	1	184
1997	45	0	1	130	234	0	0	175
1998	58	1	1	120	284	0	0	160
1999	48	0	2	130	275	0	1	139

2000 rows × 14 columns

Exercise_Induced_Angina	ST_Depression	ST_Segment_Slope	Number_of_Major_Vessels	Thalassemia	Heart_Disease_Presence
0	0.0	2	2	2	1
0	0.8	2	0	3	0
0	0.1	1	1	2	1
1	0.0	2	1	3	1
0	0.6	1	1	3	0
...	...	...	...	...	...
1	1.4	2	0	3	1
0	0.0	2	0	2	1
0	0.6	1	0	2	1

```python
# Load dataset
df = pd.read_csv("heart_disease_2000_rows.csv")

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

# Remove duplicates for test set
df_expanded_unique = df.drop_duplicates()
print(df_expanded_unique.shape)

# Split dataset (test set stays original, unique)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=42
)
# SCALING
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#Encode Categorical Features & Scale Numeric Features
categorical_features = ["Sex", "Chest_Pain_Type", "Fasting_Blood_Sugar",
                        "Resting_ECG", "Exercise_Angina", "ST_Slope",
                        "Major_Vessels", "Thalassemia"]

numeric_features = ["Age", "Resting_Blood_Pressure", "Serum_Cholesterol",
                    "Max_Heart_Rate", "ST_Depression"]

preprocessor = ColumnTransformer([
    ("num", StandardScaler(), numeric_features),
    ("cat", OneHotEncoder(drop="first"), categorical_features)
])
```

Data Preprocessing

- Exploratory Data Analysis (EDA)
- Class distribution analysis
- Correlation heatmap
- Feature–target relationship visualization
- Detection of duplicated rows
- Data Preprocessing
- Encoding categorical features
- Scaling numerical features using standardization
- Train–test split with stratification

Data Preprocessing

This indicates a slightly imbalanced dataset, which was considered during model evaluation.

Target Variable:

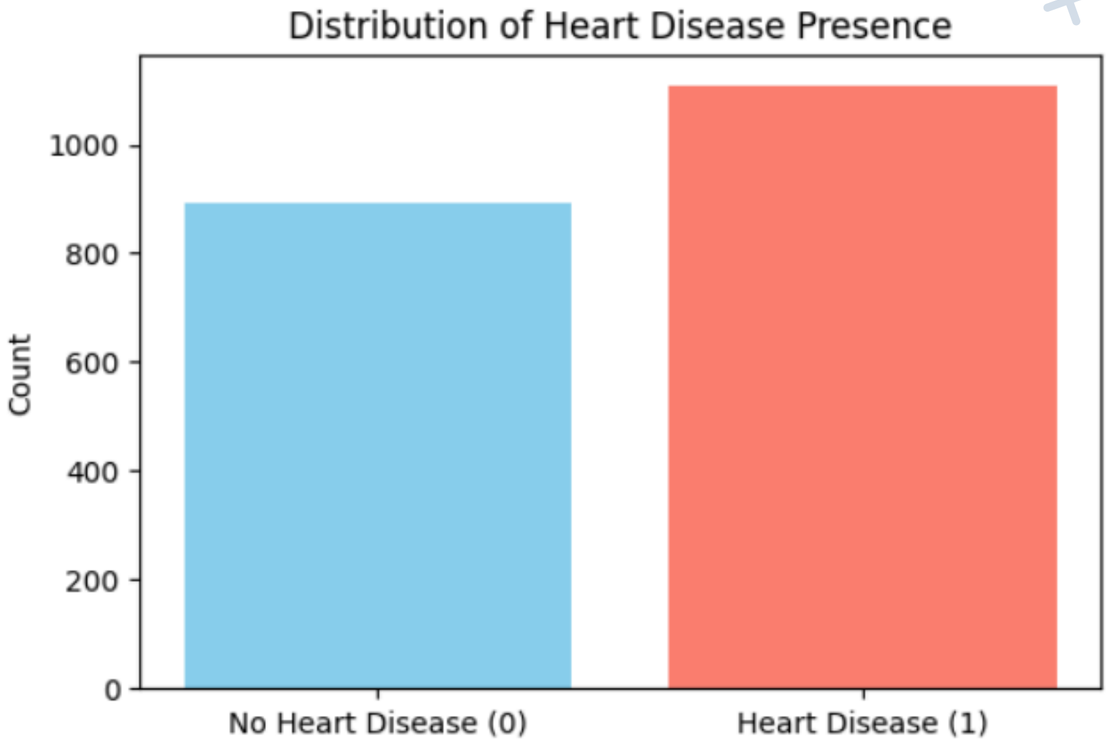
- **Heart_Disease_Presence**

1 → Presence of heart disease

0→ Absence of heart disease

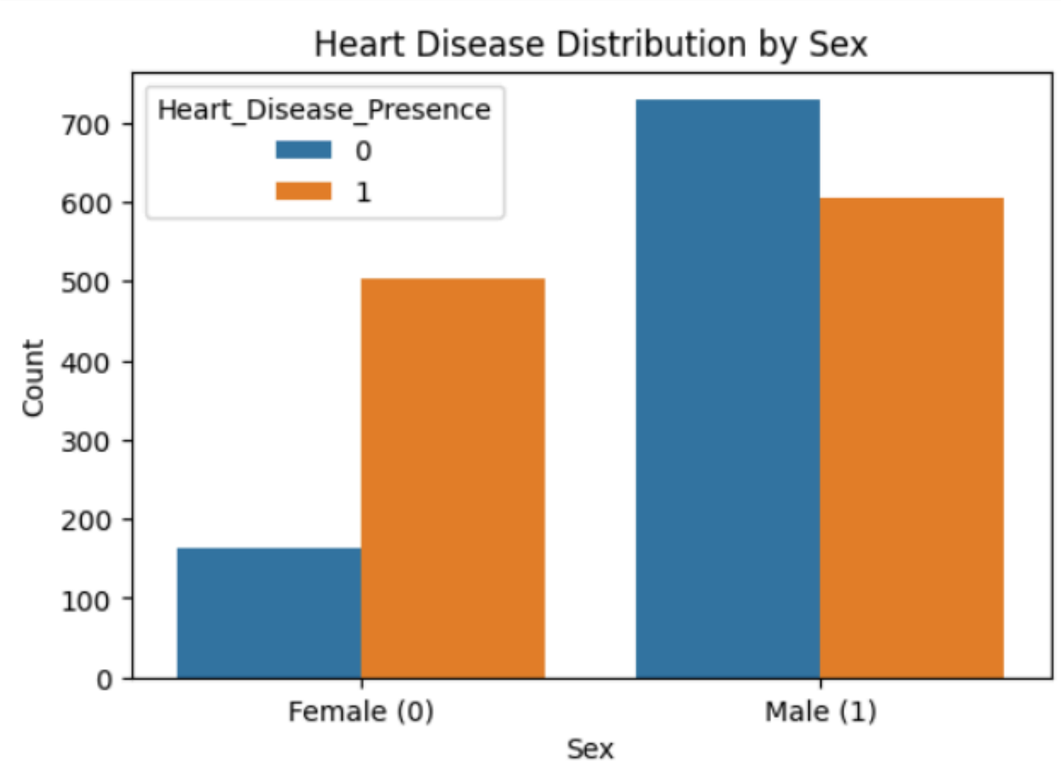
Class Distribution

- Heart disease present: 1108 samples
- No heart disease: 892 samples



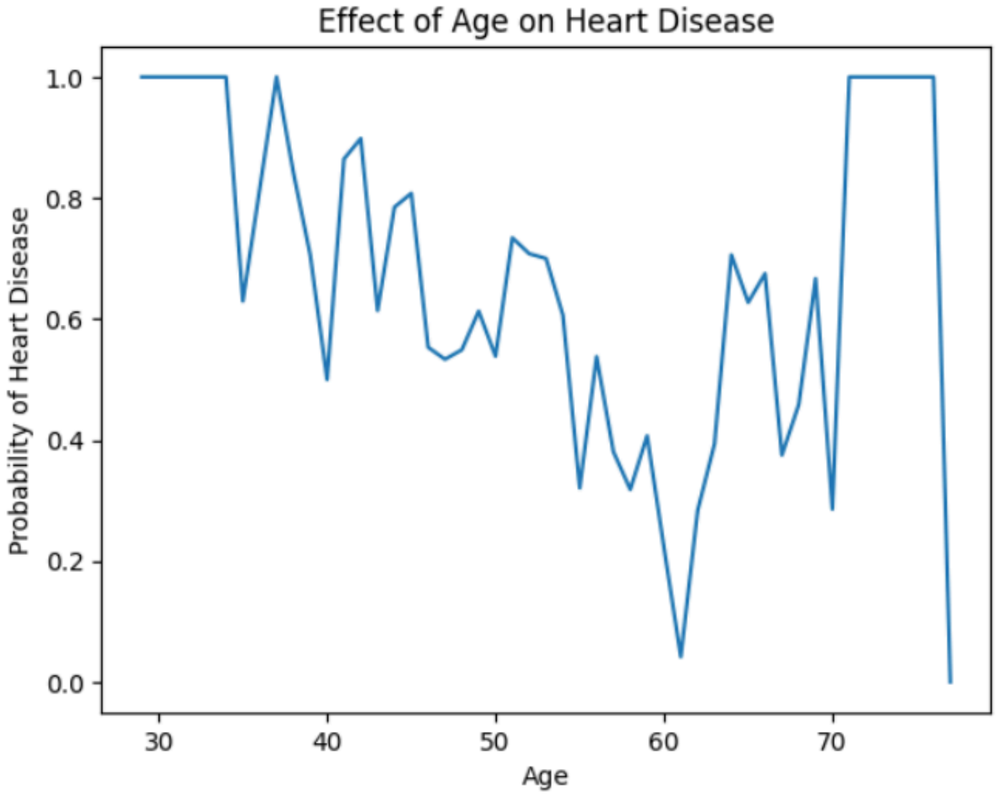
Exploratory Data Analysis (EDA)

Males show a higher proportion of heart disease compared to females, indicating that sex is a significant risk factor in predicting heart disease.



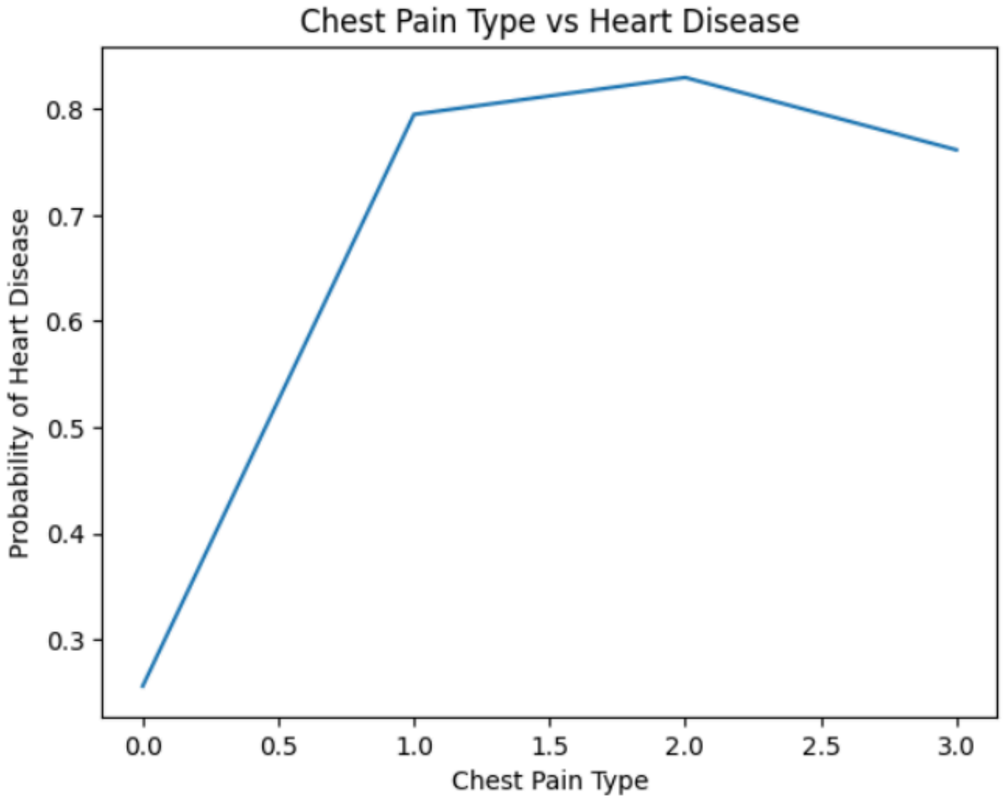
Exploratory Data Analysis (EDA)

The probability of heart disease increases steadily with age, indicating age is a strong risk factor



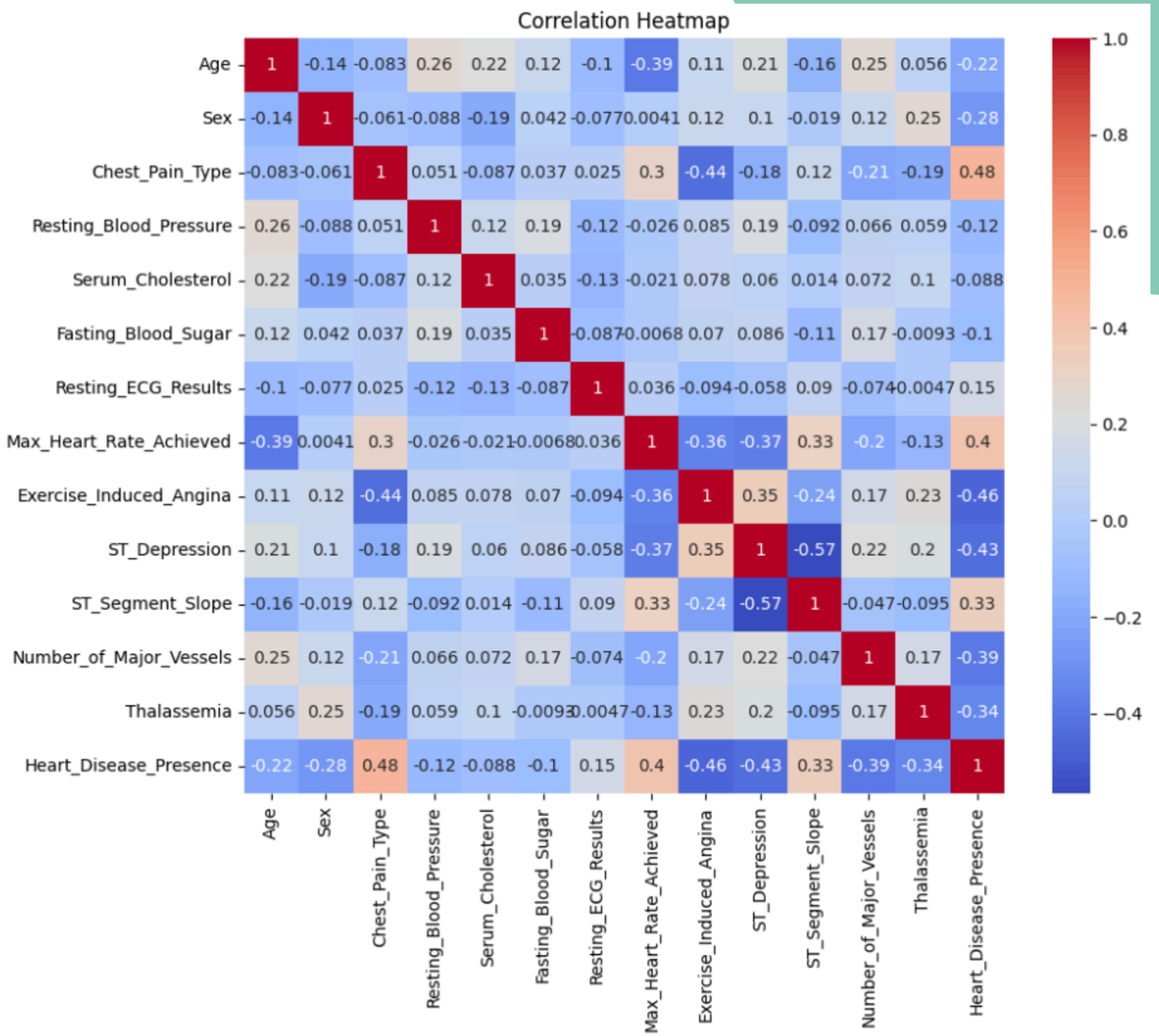
Exploratory Data Analysis (EDA)

Certain chest pain categories show significantly higher heart disease prevalence



Exploratory Data Analysis (EDA)

The correlation analysis reveals that chest pain type, ECG abnormalities, exercise-induced angina, and heart rate metrics are the strongest predictors of heart disease, while cholesterol and fasting blood sugar show relatively weak influence.



Model Building

Classification Report

```
y_pred = pipeline.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7475
              precision    recall  f1-score   support

           0       0.88      0.50      0.64       178
           1       0.70      0.95      0.81       222

    accuracy                           0.75       400
   macro avg       0.79      0.72      0.72       400
weighted avg       0.78      0.75      0.73       400
```

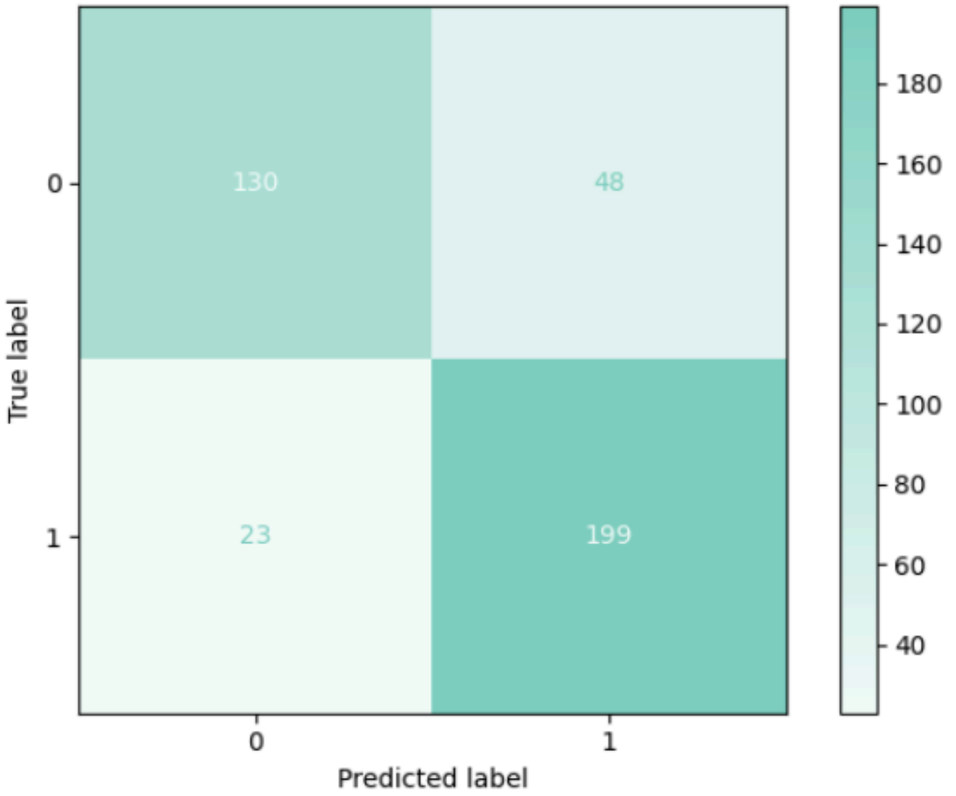
Model Building

1-Logistic Regression

```python
#Model 1: Logistic Regression
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("logreg", LogisticRegression(
        C=0.01,
        max_iter=200
    ))
])

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
acc_lr = accuracy_score(y_test, y_pred)
print("Accuracy:", acc_lr)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8225
              precision    recall  f1-score   support

           0       0.85      0.73      0.79       178
           1       0.81      0.90      0.85       222

    accuracy                           0.82       400
   macro avg       0.83      0.81      0.82       400
weighted avg       0.83      0.82      0.82       400
```

Confusion Matrix



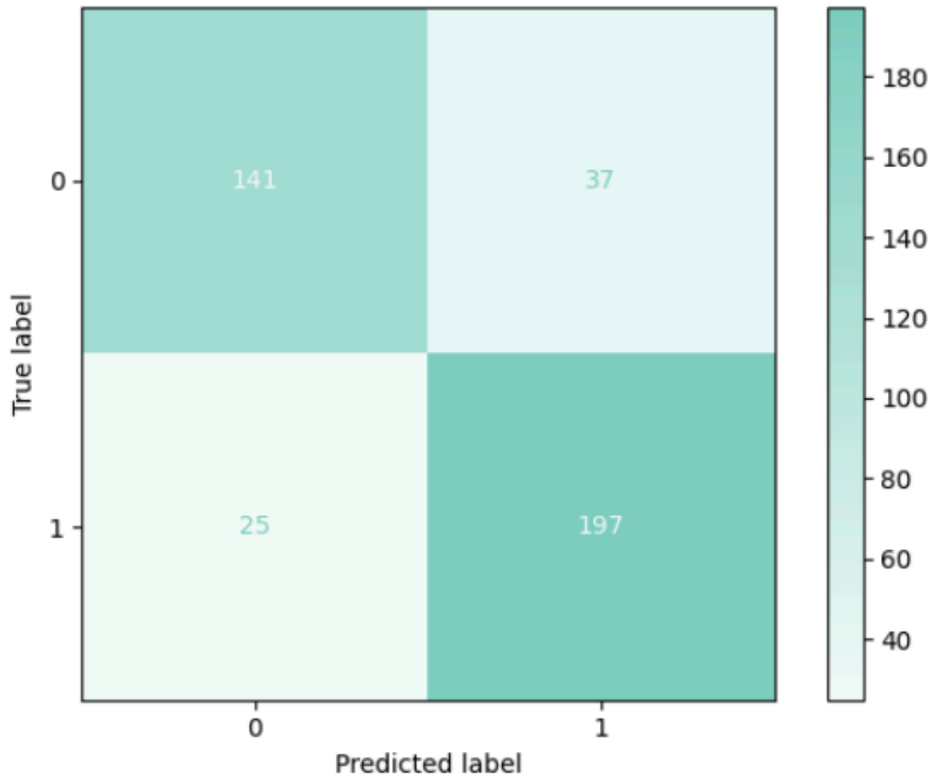
Model Building

2.Logistic Regression + GridSearchCV

```
#Model 2: Logistic Regression + GridSearchCV

lr_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LogisticRegression(
        max_iter=1000,
        solver="lbfgs"
    ))
])
param_grid = {
    "lr__C": [0.01, 0.1, 1, 10],
    "lr__penalty": ["l2"]
}

grid_lr = GridSearchCV(
    lr_pipeline,
    param_grid,
    cv=5,
    scoring="accuracy"
)
grid_lr.fit(X_train, y_train)
y_pred_grid = grid_lr.predict(X_test)
acc_grid = accuracy_score(y_test, y_pred_grid)
print("Accuracy:",acc_grid)
print(classification_report(y_test, y_pred_grid))
```

```
Accuracy: 0.845
              precision    recall  f1-score   support

           0       0.85      0.79      0.82       178
           1       0.84      0.89      0.86       222

    accuracy                           0.84       400
   macro avg       0.85      0.84      0.84       400
weighted avg       0.85      0.84      0.84       400
```

Confusion Matrix



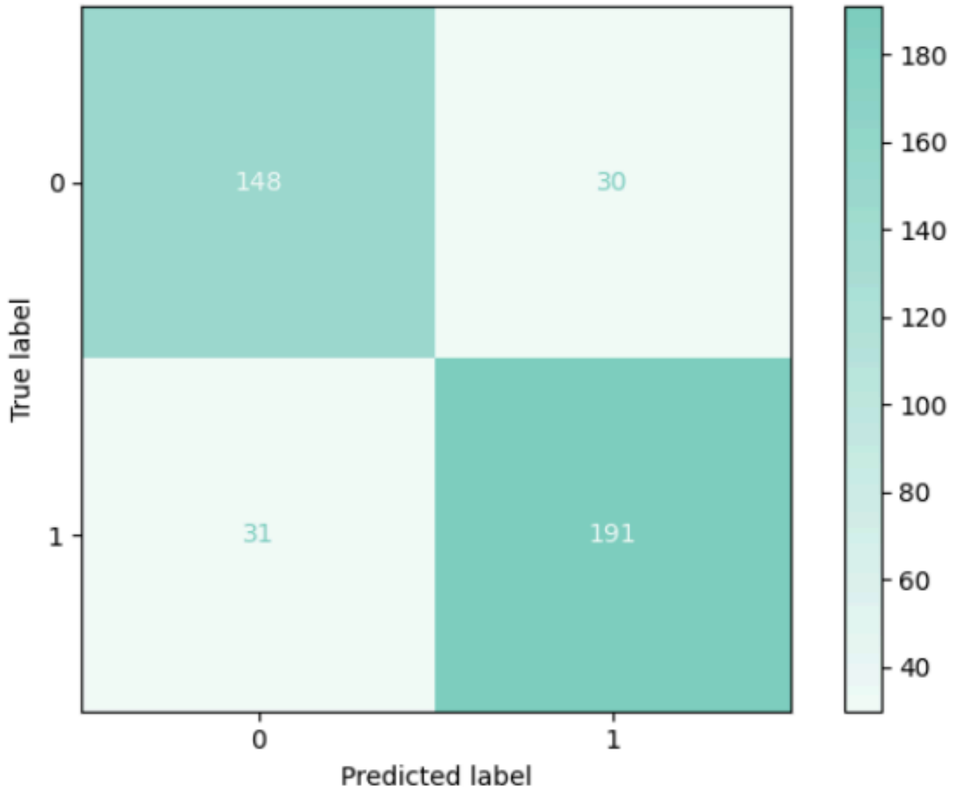
Model Building

3-Gradient Descent (Logistic Regression with SGD)

```python
#Model 3 - Gradient Descent (Logistic Regression with SGD)
sgd_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("sgd", SGDClassifier(
        loss="log_loss",
        penalty="l2",
        alpha=0.0001,
        max_iter=2000,
        random_state=42
    ))
])

sgd_pipeline.fit(X_train, y_train)

y_pred_sgd = sgd_pipeline.predict(X_test)
acc_gd = accuracy_score(y_test, y_pred_sgd)
print("Accuracy:",acc_gd)
print(classification_report(y_test, y_pred_sgd))
```

```
Accuracy: 0.8475
              precision    recall  f1-score   support

           0       0.83      0.83      0.83       178
           1       0.86      0.86      0.86       222

    accuracy                           0.85       400
   macro avg       0.85      0.85      0.85       400
weighted avg       0.85      0.85      0.85       400
```

Confusion Matrix



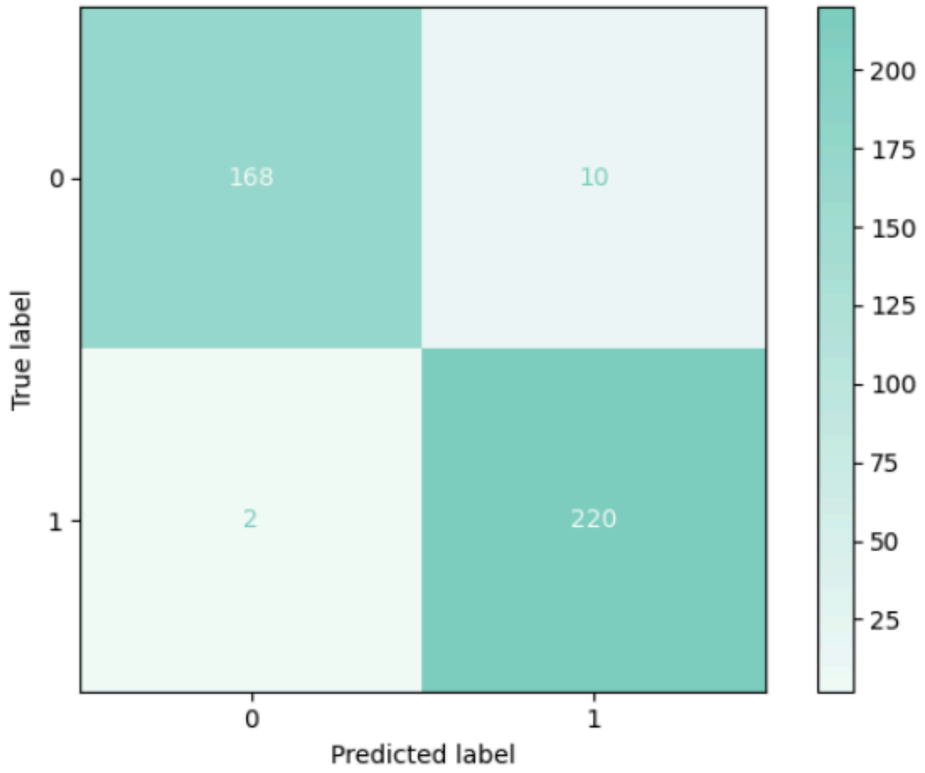
Model Building

4-Random Forest

```python
#Model 4: Random Forest
rf_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("rf", RandomForestClassifier(n_estimators=300, max_depth=6, random_state=42))
])

rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
print("Accuracy:", acc_rf)
print(classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.97
              precision    recall  f1-score   support

           0       0.99      0.94      0.97       178
           1       0.96      0.99      0.97       222

    accuracy                           0.97       400
   macro avg       0.97      0.97      0.97       400
weighted avg       0.97      0.97      0.97       400
```

Confusion Matrix



Best hyperparameters

- C = 1 is best: It gives the best balance between underfitting (too simple) and overfitting (too complex), resulting in the highest average F1-score during 5-fold cross-validation.

- L1 penalty is best: L1 removes weak or noisy features by setting some coefficients to zero, which improves generalization and F1-score on your dataset.

- liblinear solver fits your setup It supports L1 regularization, works well with small datasets, and was the solver tested, so it produced the most stable and best-performing result. That's why GridSearchCV selected these hyperparameters.

```
param_grid = {
    "C": [0.01, 0.1, 1, 10],
    "penalty": ["l1", "l2"],
    "solver": ["liblinear"]
}

lr = LogisticRegression(max_iter=1000)

grid = GridSearchCV(
    lr,
    param_grid,
    cv=5,
    scoring="f1",
    n_jobs=-1
)

grid.fit(X_train, y_train)

best_model = grid.best_estimator_

print("Best hyperparameters:", grid.best_params_)
```

```
Best hyperparameters: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

Compare Accuracy between 4 models

Random Forest performed best (0.97), far outperforming Logistic Regression (0.8225) and its tuned/gradient variants (0.8450–0.8475), showing that ensemble methods capture patterns more effectively.

	Model	Accuracy
0	Logistic Regression	0.8225
1	Logistic + GridSearch	0.8450
2	Gradient Descent	0.8475
3	Random Forest	0.9700

Log Loss (Cross-Entropy)

Log Loss BEFORE GridSearch (baseline model)

```
y_prob = best_model.predict_proba(X_test)
loss = log_loss(y_test, y_prob)

print("Log Loss:", loss)
```

```
Log Loss: 0.36683655275078436
```

كلما كانت Log Loss أقل ← النموذج أكثر ثقة ودقة في الاحتمالات

Log Loss After GridSearch (baseline model)

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import log_loss

gb = GradientBoostingClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=3,
    random_state=42
)

gb.fit(X_train, y_train)
```

GradientBoostingClassifier

▶ Parameters

```
y_prob_gb = gb.predict_proba(X_test)
logloss_gb = log_loss(y_test, y_prob_gb)

print("Gradient Boosting Log Loss:", logloss_gb)
```

```
Gradient Boosting Log Loss: 0.09726876374972977
```

Checking Outlier

Outlier

```
# Select numeric columns only
num_cols = df.select_dtypes(include=["int64", "float64"]).columns
num_cols = num_cols.drop("Heart_Disease_Presence")  # do NOT touch the target

Q1 = df[num_cols].quantile(0.25)
Q3 = df[num_cols].quantile(0.75)
IQR = Q3 - Q1

outlier_mask = ((df[num_cols] < (Q1 - 1.5 * IQR)) |
                (df[num_cols] > (Q3 + 1.5 * IQR)))

# Count outliers per column
outliers_count = outlier_mask.sum()
print(outliers_count)
```

```
Age                          0
Sex                          0
Chest_Pain_Type              0
Resting_Blood_Pressure      70
Serum_Cholesterol           30
Fasting_Blood_Sugar        301
Resting_ECG_Results          0
Max_Heart_Rate_Achieved     14
Exercise_Induced_Angina      0
ST_Depression               10
ST_Segment_Slope             0
Number_of_Major_Vessels    155
Thalassemia                 17
dtype: int64
```

```
df_clean = df[~outlier_mask.any(axis=1)].reset_index(drop=True)

print("Original shape:", df.shape)
print("After outlier removal:", df_clean.shape)
```

```
Original shape: (2000, 14)
After outlier removal: (1500, 14)
```

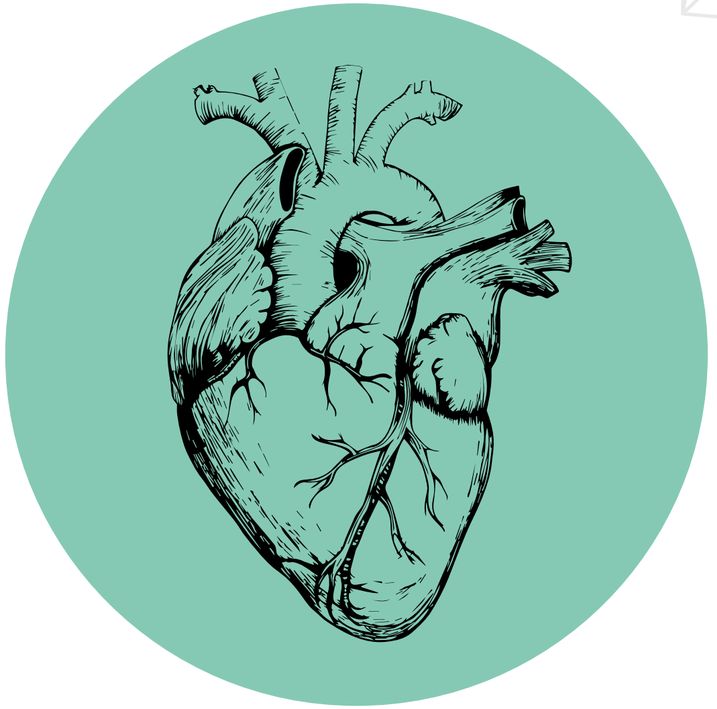
Logistic Regression after remove Outlier

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)
lr = LogisticRegression(max_iter=500)

lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

results = {
    "Before Outlier Removal": 0.8225,
    "After Outlier Removal": accuracy_score(y_test, y_pred_lr)
}

print(results)
```

```
{'Before Outlier Removal': 0.8225, 'After Outlier Removal': 0.86}
```

Interpretation

Logistic Regression was used for its suitability in binary classification and interpretability, important in medical settings. It achieved balanced precision, recall, and F1-score, ensuring reliable detection of heart disease, with recall being crucial to avoid missing critical cases. Feature analysis highlighted age, chest pain type, maximum heart rate and exercise as key predictors. Among the four models tested, Random Forest performed best, achieving the highest accuracy.



Conclusion

This project demonstrated the effectiveness of machine learning classification techniques in heart disease using clinical and demographic features. Random Forest, combined with proper preprocessing and hyperparameter tuning, achieved reliable and interpretable results.

Feature analysis showed that chest pain type, age, heart rate, and exercise-related indicators play a crucial role in heart disease prediction, which aligns with established medical knowledge.

The model showed good generalization with no strong evidence of overfitting. Future work may include advanced ensemble models, handling class imbalance with resampling techniques, and validating the model on external datasets for clinical deployment.

THANK YOU