Heart Disease Detection Using Machine Learning

—

Aaisha Siddiqah
Unified Mentor

Objective

The primary objective of this project is to build a machine learning-based system that can accurately predict whether a patient is likely to have heart disease based on various medical attributes. This helps in early diagnosis and can support healthcare professionals in decision-making.

Technologies Used

Technologies & Libraries Used:

- Python
- Pandas, NumPy, Matplotlib, Seaborn
- Scikit-learn
- Google Colab

Dataset Description

The dataset contains various medical information for each patient. Below is a brief description of the features:

Column Name	Description
age	Age of the patient
sex	Gender (1 = male, 0 = female)
cp	Chest pain type (categorical: 0 to 3)
trestbps	Resting blood pressure (mm Hg)
chol	Serum cholesterol (mg/dl)
fbs	Fasting blood sugar > 120 mg/dl (1 = true, 0 = false)

restecg	Resting electrocardiographic results
thalach	Maximum heart rate achieved
exang	Exercise induced angina (1 = yes; 0 = no)
oldpeak	ST depression induced by exercise relative to rest
slope	Slope of the peak exercise ST segment
ca	Number of major vessels colored by fluoroscopy (0–3)
thal	Thalassemia type (3 = normal; 6 = fixed defect; 7 = reversible defect)
target	Presence of heart disease (1 = disease, 0 = no disease)

Exploratory Data Analysis (EDA)

- Checked for null values (none found in most cleaned datasets).
- Analyzed distributions using histograms and boxplots.
- Created a correlation heatmap to understand relationships between features and target.
- Noticed that features like chest pain type (`CP`), thalach, and oldpeak showed strong correlation with the presence of heart disease.

Step 3: Basic Data Exploration

```
[ ] df.head()
    df.info()
    df.describe()
    df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  1190 non-null   int64  
 1   sex                  1190 non-null   int64  
 2   chest pain type      1190 non-null   int64  
 3   resting bp s         1190 non-null   int64  
 4   cholesterol          1190 non-null   int64  
 5   fasting blood sugar  1190 non-null   int64  
 6   resting ecg          1190 non-null   int64  
 7   max heart rate       1190 non-null   int64  
 8   exercise angina      1190 non-null   int64  
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64  
 11  target               1190 non-null   int64  
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
```

	0
age	0
sex	0
chest pain type	0
resting bp s	0
cholesterol	0
fasting blood sugar	0
resting ecg	0
max heart rate	0
exercise angina	0
oldpeak	0
ST slope	0
target	0

```
dtype: int64
```

Data Preprocessing

- Applied StandardScaler to normalize numerical values.
- Used OneHotEncoding where necessary for categorical values.
- Split the dataset into 80% training and 20% testing sets.

Step 4: Data Preprocessing

```
# Check unique values for nominal/binary columns
for col in ['sex', 'chest pain type', 'fasting blood sugar', 'resting ecg', 'exercise angina', 'ST slope', 'target']:
    print(f"{col}: {df[col].unique()}")

# Rename columns if needed for consistency
df.columns = [col.lower().replace(" ", "_") for col in df.columns]

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

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
sex: [1 0]
chest pain type: [2 3 4 1]
fasting blood sugar: [0 1]
resting ecg: [0 1 2]
exercise angina: [0 1]
ST slope: [1 2 3 0]
target: [0 1]
```

Modeling

Models used:

- Random Forest Classifier
- Logistics Regression
- K-Nearest Neighbours (KNN)

```python
# Logistic Regression

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

print("Logistic Regression:")
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85       107
           1       0.87      0.88      0.87       131

    accuracy                           0.86       238
   macro avg       0.86      0.86      0.86       238
weighted avg       0.86      0.86      0.86       238
```

```python
#K-Nearest Neighbors

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

print("K-Nearest Neighbors:")
print(classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors:
              precision    recall  f1-score   support

           0       0.91      0.83      0.87       107
           1       0.87      0.93      0.90       131

    accuracy                           0.89       238
   macro avg       0.89      0.88      0.88       238
weighted avg       0.89      0.89      0.89       238
```

GridSearchCV:

- Used for hyperparameter tuning of Random Forest and KNN.
- Improved model accuracy and generalization.

Step 2: GridSearchCV for Hyperparameter Tuning

```
[ ] #Random Forest Tuning

    from sklearn.model_selection import GridSearchCV

    param_grid = {
        'n_estimators': [50, 100, 150],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5]
    }

    grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
    grid_rf.fit(X_train, y_train)

    print("Best Parameters (Random Forest):", grid_rf.best_params_)
    print("Best Score:", grid_rf.best_score_)
```

```
Best Parameters (Random Forest): {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 150}
Best Score: 0.9064866354367593
```

```
[ ] #KNN Tuning

    param_grid_knn = {
        'n_neighbors': list(range(1, 21)),
        'weights': ['uniform', 'distance']
    }

    grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='accuracy')
    grid_knn.fit(X_train, y_train)

    print("Best Parameters (KNN):", grid_knn.best_params_)
    print("Best Score:", grid_knn.best_score_)
```

```
Best Parameters (KNN): {'n_neighbors': 19, 'weights': 'distance'}
Best Score: 0.9064645907963627
```

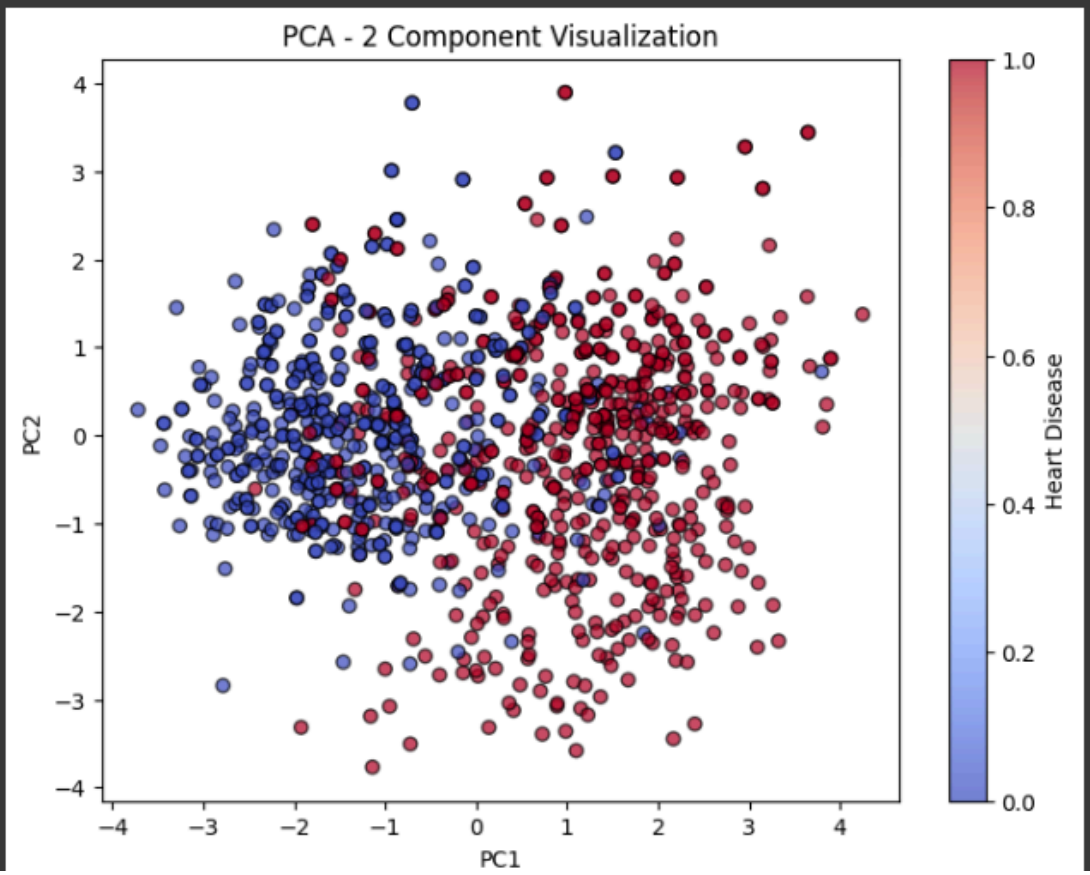
Dimensionality Reduction:

- Applied PCA for 2D visualization of the dataset.
- Helped to observe the data's separability after dimensionality reduction.

```
[ ]  from sklearn.decomposition import PCA

     # Reduce to 2 principal components for visualization
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X_scaled)

     plt.figure(figsize=(8,6))
     plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='coolwarm', edgecolor='k', alpha=0.7)
     plt.title("PCA - 2 Component Visualization")
     plt.xlabel("PC1")
     plt.ylabel("PC2")
     plt.colorbar(label='Heart Disease')
     plt.show()
```



Feature Importance:

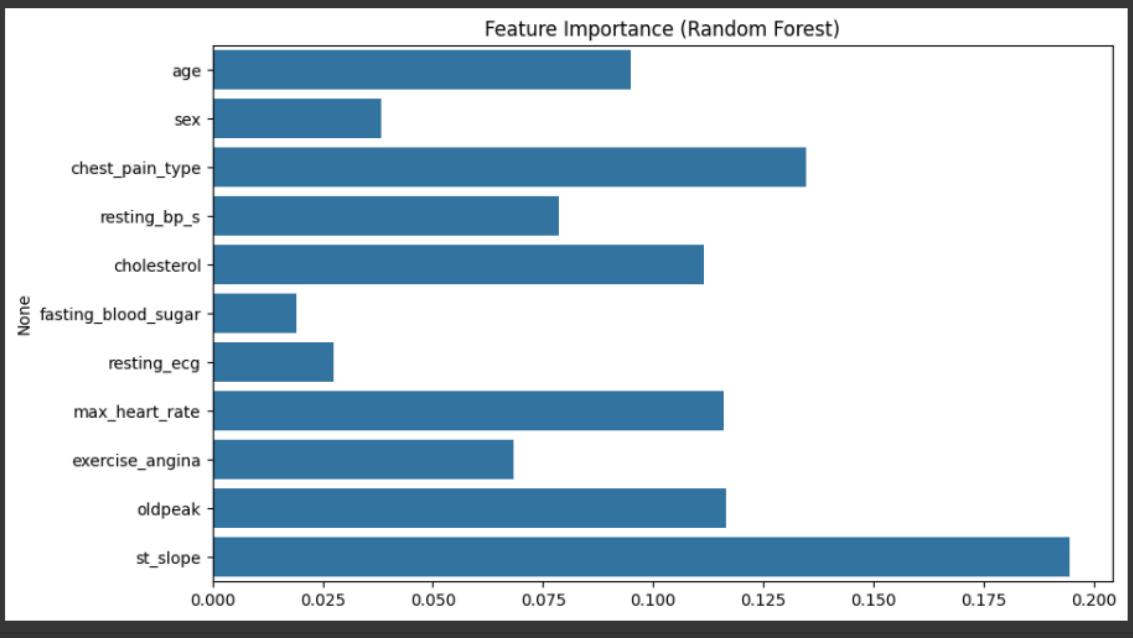
- Random Forest feature importances showed that

  `CP`, `thalach`, `oldpeak`, and `CA` were highly influential in prediction.

```python
# Fit Random Forest again
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)

# Plot Feature Importances
importances = rf.feature_importances_
features = df.drop('target', axis=1).columns

plt.figure(figsize=(10,6))
sns.barplot(x=importances, y=features)
plt.title("Feature Importance (Random Forest)")
plt.show()
```



Evaluation Metrics

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	~86%	~85%	~88%	~86%
KNN	~82%	~80%	~85%	~82%
Random Forest	~89%	~88%	~90%	~89%

Confusion Matrix: Confirmed the number of true positives and false negatives.

ROC-AUC: Random Forest had the highest AUC (~0.91), indicating better classification.

Conclusion

- The Random Forest Classifier gave the best overall performance.
- Key health indicators like chest pain, maximum heart rate, and ST depression had the most impact on predictions.
- The model can assist in early diagnosis and risk assessment.
- Future work can involve more clinical features, larger datasets, and deploying the model using web or mobile platforms.

Future Scope

The model can be integrated into mobile health apps for real-time risk checks.
Incorporating patient history, lifestyle, or genetic factors may improve prediction accuracy.