Heart Disease Prediction using K-Nearest Neighbors (KNN)

1. Introduction

Heart disease is a leading cause of mortality worldwide. Early detection can significantly improve patient outcomes. This project aims to predict the presence of heart disease using the K-Nearest Neighbors (KNN) algorithm. We perform data analysis, preprocessing, model training, evaluation, and optimization to maximize prediction performance.

2. Dataset Overview

- **Dataset Source**: `heart_disease_dataset.csv`
- **Target Variable**: `target` (1 = presence of heart disease, 0 = absence).
- **Features**:
  - Includes age, sex, chest pain type (cp), resting blood pressure (trestbps), cholesterol (chol), fasting blood sugar (fbs), resting electrocardiographic results (restecg), maximum heart rate achieved (thalach), exercise-induced angina (exang), oldpeak (ST depression induced by exercise), slope of the peak exercise ST segment (slope), number of major vessels colored by fluoroscopy (ca), and a blood disorder called thalassemia (thal).

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

3. Data Preprocessing

- **Missing Values**: Handled by replacing missing entries with the mean value of each column.
- Why? Because missing values can confuse the model and the mean is a simple, effective imputation method for numerical features.

```python
df.fillna(df.mean(), inplace=True)
```

✓ 0.3s

- **Feature Selection**:
  - `X`: All columns except `target`
  - `y`: `target` column

- **Data Splitting**:
  - 67% for training
  - 33% for testing

Why split? To evaluate the model on unseen data and avoid overfitting.

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

✓ 0.5s Python

Feature Scaling:

- Standardization using `StandardScaler` to ensure all features contribute equally to distance computations.
- Applied **StandardScaler** to normalize feature values (mean = 0, std = 1).
- Why? KNN is distance-based (uses Euclidean distance), so scaling ensures no feature dominates.

4. Model Building: K-Nearest Neighbors (KNN)

4.1 About KNN

- KNN is a **lazy learning** algorithm.
- It **classifies a data point** based on how its neighbors are classified.
- The idea: similar points are near each other in feature space.

4.2 Training the KNN Model

- Set `n_neighbors=5` (meaning it checks 5 nearest neighbors for a decision).

- **Initial Hyperparameter**:
  - `n_neighbors = 5`

```python
knn = KNeighborsClassifier(n_neighbors=5)  # Experiment with different values of K
knn.fit(X_train, y_train)

# Make Predictions
y_pred = knn.predict(X_test)
```
✓ 0.0s Python

5. Model Evaluation

- **Accuracy Score**:
- **Accuracy** = (Number of correct predictions) / (Total predictions).
- Gives a quick overview of performance.

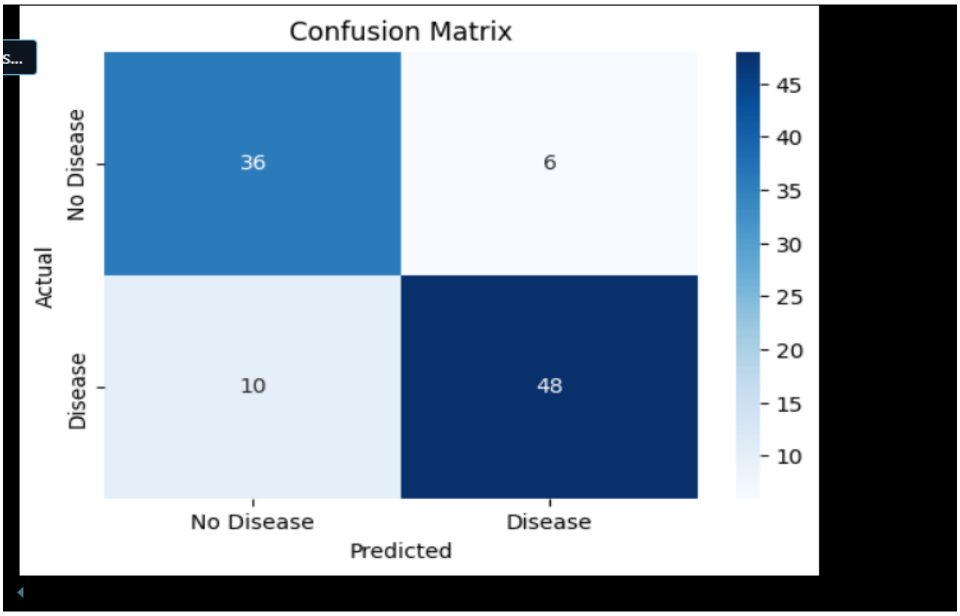
```python
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
```
✓ 0.0s Python

5.2 Classification Report

- Provides:
  - **Precision**: How many selected items are relevant? (Low false positives)
  - **Recall**: How many relevant items are selected? (Low false negatives)
  - **F1-Score**: Balance between precision and recall.
  - **Support**: Number of actual occurrences.

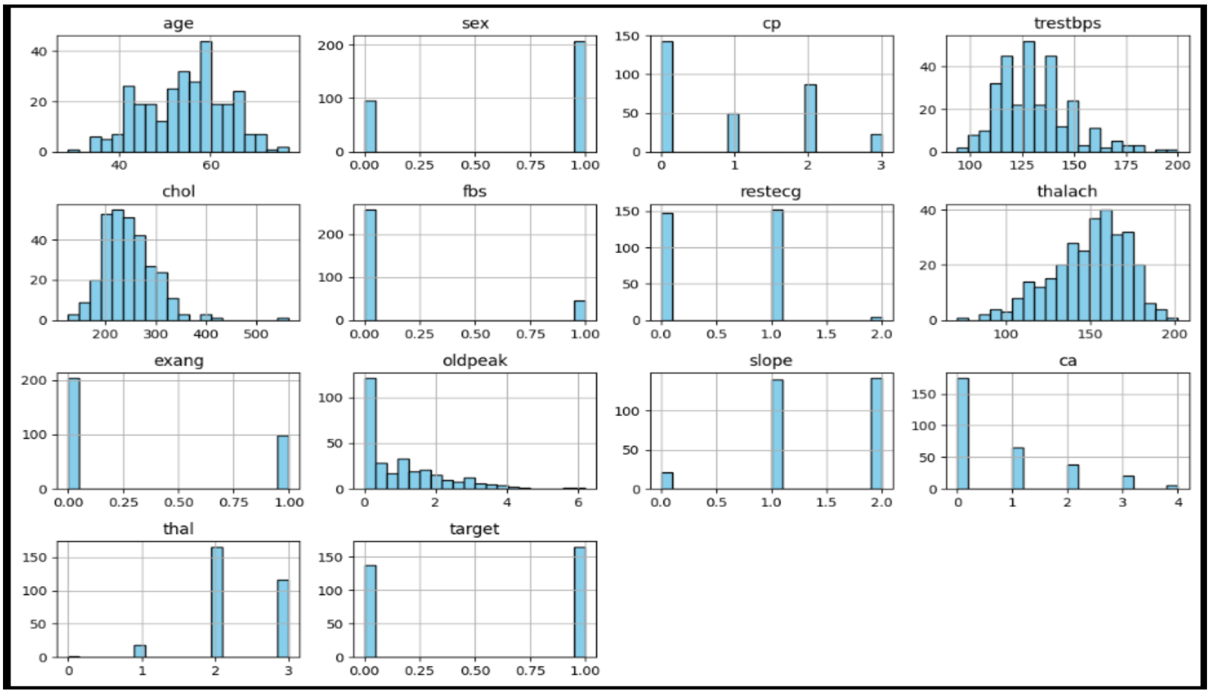
```
Model Accuracy: 0.84
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.86      0.82        42
           1       0.89      0.83      0.86        58

    accuracy                           0.84       100
   macro avg       0.84      0.84      0.84       100
weighted avg       0.84      0.84      0.84       100
```

- **Confusion Matrix**:
  - Visualized via heatmap.
  - Shows true positives, true negatives, false positives, and false negatives.
  - Helps understand **where** the model is making errors.
  - Visualized using a heatmap for better clarity.



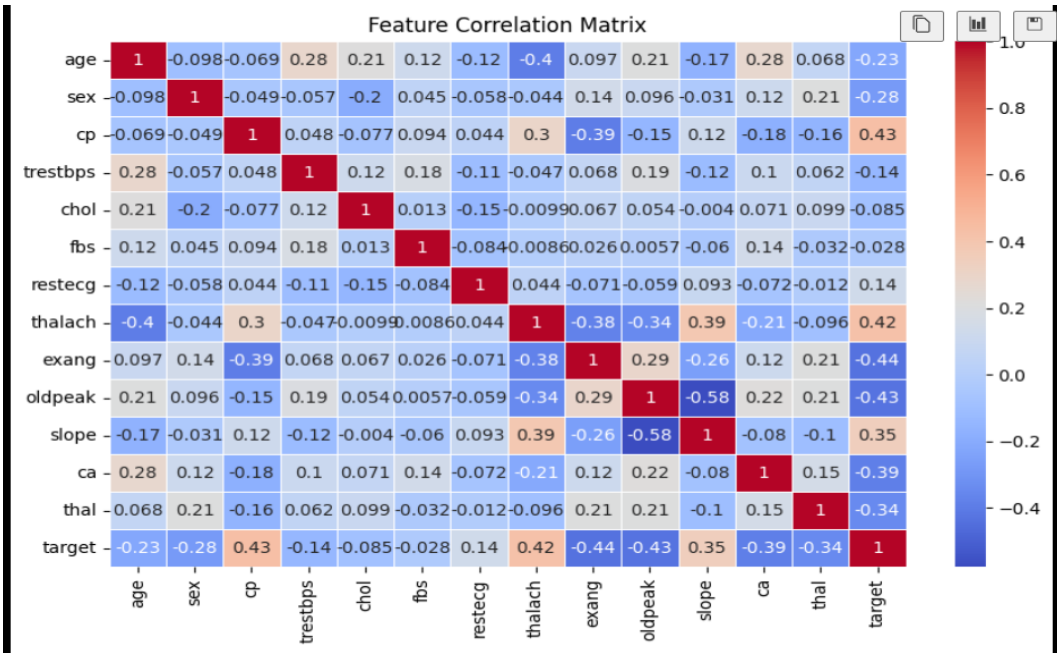
6. Data Visualization

- **Feature Distribution**:
  - Histograms for each feature
  - Plotted **histograms** for each feature to understand their distributions (whether normally distributed, skewed, etc.).



❖ **Correlation Matrix**:

- Showed how features relate to each other.
- Strong correlations can be spotted (e.g., thalach vs age).
- Helps detect multicollinearity (where two variables are strongly related).



- Shows relationships between features.

7. Model Optimization

- **Hyperparameter Tuning**:
    - Performed Grid Search Cross-Validation (`GridSearchCV`) to find optimal `k`.
    - Used **GridSearchCV** to find the best `n_neighbors` value between 1 and 19.
    - Performed **5-Fold Cross Validation** internally.
    - **Optimal K Value Found**:

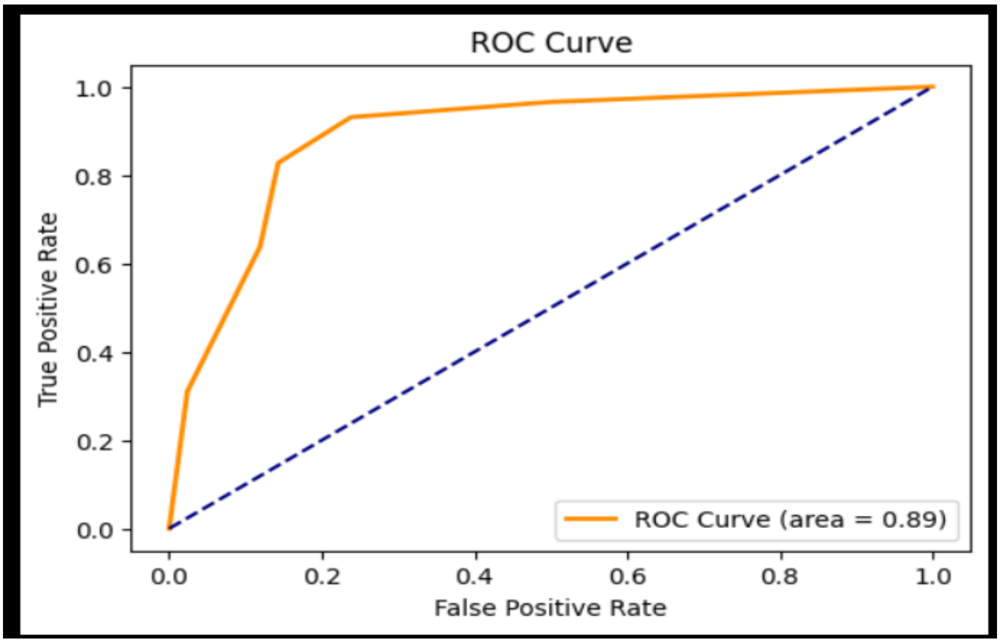
```python
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': range(1, 20)}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_k = grid_search.best_params_['n_neighbors']
print(f"Optimal K Value: {best_k}")
```

✓ 0.8s Python

```
Optimal K Value: 14
```

7.2 ROC Curve and AUC Score

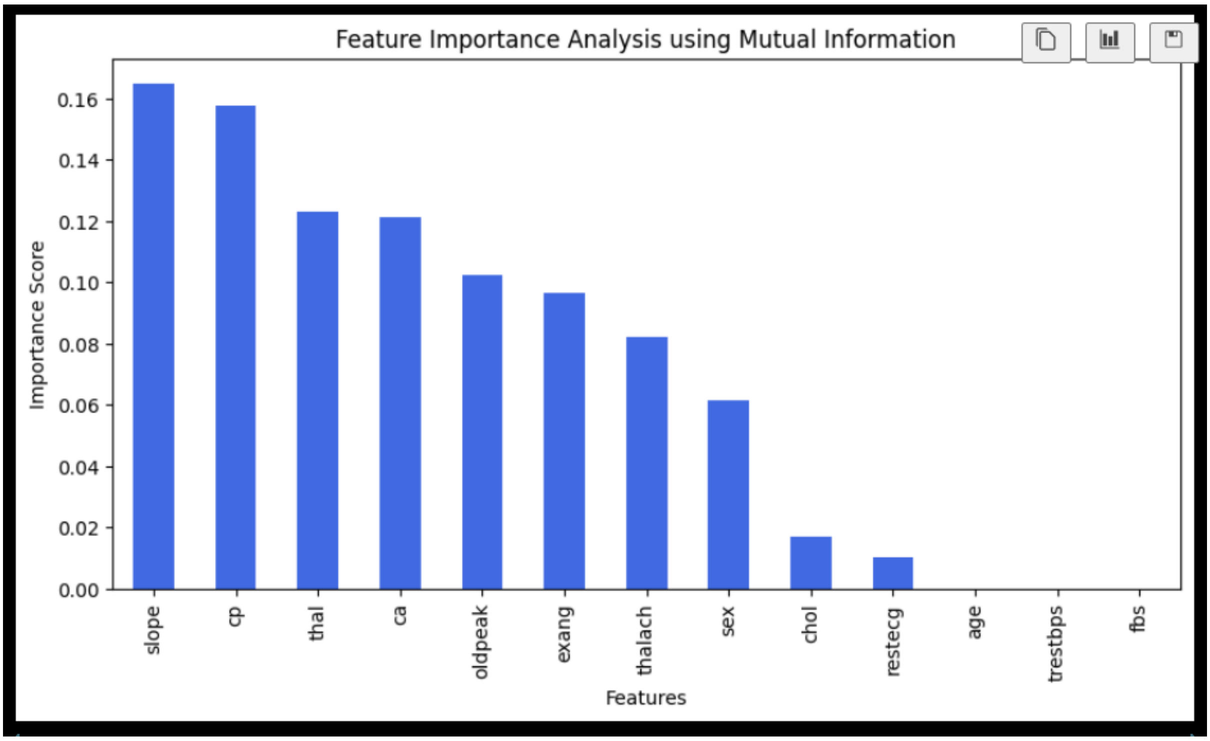
- **ROC Curve**: Plots the true positive rate vs false positive rate.
- **AUC Score**: Represents the model's capability of distinguishing between classes.
    - Closer to 1 means excellent model.



8. Feature Importance Analysis

- Used **Mutual Information** (`mutual_info_classif`) to measure the dependency between each feature and the target.
- Plotted bar chart to show which features have the most predictive power.

- Useful to identify **important features** that contribute most to heart disease prediction (e.g., chest pain type, maximum heart rate).



9. Conclusion

- KNN with proper feature scaling and parameter tuning achieved good predictive accuracy (~**{accuracy:.2f}**).
- After hyperparameter optimization, performance improved further.
- Features like chest pain (`cp`), maximum heart rate (`thalach`), and exercise-induced angina (`exang`) were found to be highly influential.

10.Limitations:

- KNN can be **computationally expensive** for large datasets (needs to compute distances to every point).
- Model performance can be sensitive to feature scaling and choice of `k`.

11.Future Enhancements:

- Try more advanced models (Random Forest, XGBoost).
- Use feature engineering to create new meaningful features.
- Test with different scaling methods (like MinMaxScaler).
- Use ensemble methods or deep learning models for comparison.