Binary Classifier Model

A Machine Learning Model For Heart Disease using Decision Tree and K nearest Neighbour

```
# Print dataset name and describe the healthcare problem
print("Dataset Name: Heart Disease Dataset")
print("Description: This dataset contains 14 attributes related to
heart disease. The goal is to predict the presence of heart disease.")
```

```
Dataset Name: Heart Disease Dataset
Description: This dataset contains 14 attributes related to heart
disease. The goal is to predict the presence of heart disease.
```

Importing Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Dataset

CP= Chest Paint Type Trestbps= Resting Blood Pressure Chol = Serum Cholestorel fbs = Fasting Blood Sugar restecg = Resting Electrocardiographic Result thlach = Max Heart Rate Achived exang = Exercise Induced Angina oldpeak = ST Depression Included by exercise relative to rest slope = Slope of the peak exercise ST segment ca = Number of major vessels (0-3) colored by fluoroscopy thal = Thalassemia Target = Diagonosis of Heart Disease

```
# Load the dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/heart-
disease/processed.cleveland.data"
columns = [
    'age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
'thalach', 'exang',
    'oldpeak', 'slope', 'ca', 'thal', 'target'
]
```

```
data = pd.read_csv(url, header=None, names=columns)
data.head()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 303,\n \"fields\": [\n   {\n   \"column\": \"age\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 9.038662442446746,\n   \"min\": 29.0,\n   \"max\": 77.0,\n   \"num_unique_values\": 41,\n   \"samples\": [\n   61.0,\n   64.0,\n   44.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sex\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.46729882777012993,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.0,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cp\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9601256119600138,\n   \"min\": 1.0,\n   \"max\": 4.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   4.0,\n   2.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"trestbps\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 17.59974772958769,\n   \"min\": 94.0,\n   \"max\": 200.0,\n   \"num_unique_values\": 50,\n   \"samples\": [\n   124.0,\n   192.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"chol\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 51.77691754263704,\n   \"min\": 126.0,\n   \"max\": 564.0,\n   \"num_unique_values\": 152,\n   \"samples\": [\n   321.0,\n   187.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"fbs\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.35619787492797644,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.0,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"restecg\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.9949712915251782,\n   \"min\": 0.0,\n   \"max\": 2.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   2.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"thalach\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 22.875003276980376,\n   \"min\": 71.0,\n   \"max\": 202.0,\n   \"num_unique_values\": 91,\n   \"samples\": [\n   170.0,\n   114.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"exang\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.46979446452231655,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n

```
1.0,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"oldpeak\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 1.1610750220686348,\n        \"min\": 0.0,\n
\"max\": 6.2,\n        \"num_unique_values\": 40,\n
\"samples\": [\n          2.4,\n          0.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"slope\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 0.6162261453459619,\
n        \"min\": 1.0,\n        \"max\": 3.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3.0,\n
2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ca\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"3.0\",\
n          \"?\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"thal\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          \"3.0\",\
n          \"?\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"target\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 4,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          2,\n
4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}
```


Dataset Structures

```python
# Examine the dataset structure
print("\nFirst 5 rows of the dataset:")
print(data.head())
print("\nDataset shape:")
print(data.shape)
print("\nDataset info:")
print(data.info())
print("\nDataset description:")
print(data.describe())
```

```
First 5 rows of the dataset:
    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

Dataset shape:
(303, 14)

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        303 non-null    object
 12  thal      303 non-null    object
 13  target    303 non-null    int64
dtypes: float64(11), int64(1), object(2)
memory usage: 33.3+ KB
None

Dataset description:
              age         sex          cp    trestbps        chol
fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
303.000000
mean    54.438944    0.679868    3.158416  131.689769  246.693069
0.148515
std      9.038662    0.467299    0.960126   17.599748   51.776918
0.356198
```

```
min     29.000000    0.000000    1.000000   94.000000  126.000000
0.000000
25%     48.000000    0.000000    3.000000  120.000000  211.000000
0.000000
50%     56.000000    1.000000    3.000000  130.000000  241.000000
0.000000
75%     61.000000    1.000000    4.000000  140.000000  275.000000
0.000000
max     77.000000    1.000000    4.000000  200.000000  564.000000
1.000000

          restecg     thalach       exang     oldpeak       slope
target
count  303.000000  303.000000  303.000000  303.000000  303.000000
303.000000
mean     0.990099  149.607261    0.326733    1.039604    1.600660
0.937294
std      0.994971   22.875003    0.469794    1.161075    0.616226
1.228536
min      0.000000   71.000000    0.000000    0.000000    1.000000
0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000
0.000000
50%      1.000000  153.000000    0.000000    0.800000    2.000000
0.000000
75%      2.000000  166.000000    1.000000    1.600000    2.000000
2.000000
max      2.000000  202.000000    1.000000    6.200000    3.000000
4.000000
```

Handling Missing Values

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

# Convert 'ca' and 'thal' columns to numeric, setting errors='coerce'
to handle '?' entries
data['ca'] = pd.to_numeric(data['ca'], errors='coerce')
data['thal'] = pd.to_numeric(data['thal'], errors='coerce')

# Handle missing values by filling with the median of the column
data['ca'].fillna(data['ca'].median(), inplace=True)
data['thal'].fillna(data['thal'].median(), inplace=True)

# Drop rows with any remaining missing values (if any)
data.dropna(inplace=True)
```

Spliting Datasets and Training Sets

```
# Split the dataset into training and testing sets
test_size = (40 % 10) + 5  # Given the last 3 digits of ID are 040
X = data.drop(columns=['target'])  # Features
y = data['target']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size/100, random_state=42)
```

Model Initializing and Training

```
# Initialize the models
dt_model = DecisionTreeClassifier(random_state=42)
knn_model = KNeighborsClassifier()

# Train the models
dt_model.fit(X_train, y_train)
knn_model.fit(X_train, y_train)

# Make predictions
dt_predictions = dt_model.predict(X_test)
knn_predictions = knn_model.predict(X_test)
```

Model Evaluation

```
# Evaluate the models
metrics = {
    'Accuracy': accuracy_score,
    'Precision': precision_score,
    'Recall': recall_score,
    'F1 Score': f1_score
}

print("\nDecision Tree Performance:")
for metric_name, metric in metrics.items():
    print(f"{metric_name}: {metric(y_test, dt_predictions):.4f}")

print("\nK-Nearest Neighbors Performance:")
for metric_name, metric in metrics.items():
    print(f"{metric_name}: {metric(y_test, knn_predictions):.4f}")

# Determine which model performs better in terms of F1-score
dt_f1 = f1_score(y_test, dt_predictions)
knn_f1 = f1_score(y_test, knn_predictions)
better_model = "Decision Tree" if dt_f1 > knn_f1 else "K-Nearest
```

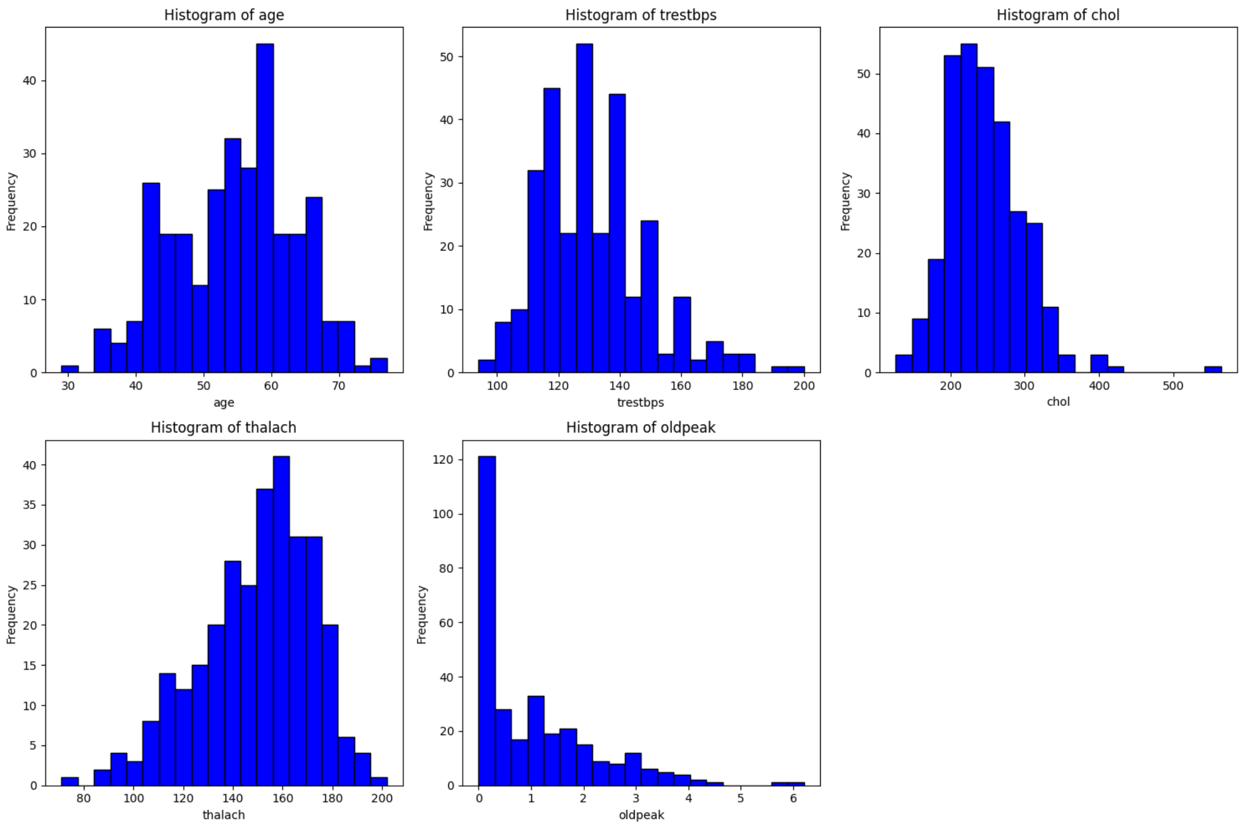
```
Neighbors"
print(f"\nBetter model in terms of F1-score: {better_model}")
```

```
Decision Tree Performance:
Accuracy: 0.8125
Precision: 0.9000
Recall: 0.8182
F1 Score: 0.8571

K-Nearest Neighbors Performance:
Accuracy: 0.7500
Precision: 0.8889
Recall: 0.7273
F1 Score: 0.8000

Better model in terms of F1-score: Decision Tree
```

Visualization of Confusion Matrix

```
# Visualization: Confusion Matrix
# Confusion matrix for Decision Tree
"""
dt_cm = confusion_matrix(y_test, dt_predictions)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.heatmap(dt_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disease', 'Disease'], yticklabels=['No
Disease', 'Disease'])
plt.title('Decision Tree Confusion Matrix')

# Confusion matrix for K-Nearest Neighbors
knn_cm = confusion_matrix(y_test, knn_predictions)
plt.subplot(1, 2, 2)
sns.heatmap(knn_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Disease', 'Disease'], yticklabels=['No
Disease', 'Disease'])
plt.title('K-Nearest Neighbors Confusion Matrix')

plt.show()"""

# Histogram Dataset
# List of continuous features in the dataset
continuous_features = ['age', 'trestbps', 'chol', 'thalach',
'oldpeak']

# Set up the figure for subplots
plt.figure(figsize=(15, 10))
```

```python

# Loop through each continuous feature and plot the histogram
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(2, 3, i)
    plt.hist(data[feature], bins=20, color='blue', edgecolor='black')
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

# Show the plots
plt.tight_layout()
plt.show()

"""
#Bar Plot
# List of categorical features in the dataset
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang',
'slope', 'ca', 'thal']

# Set up the figure for subplots
plt.figure(figsize=(15, 10))

# Loop through each categorical feature and plot a bar chart
for i, feature in enumerate(categorical_features, 1):
    plt.subplot(3, 3, i)
    sns.countplot(x=feature, data=data, palette='Set2')
    plt.title(f'Bar Plot of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')

# Show the plots
plt.tight_layout()
plt.show()"""
```


Histogram of age
Frequency
age
Histogram of trestbps
Frequency
trestbps
Histogram of chol
Frequency
chol
Histogram of thalach
Frequency
thalach
Histogram of oldpeak
Frequency
oldpeak

{"type":"string"}