Heart_Disease_Dataset_Analysis

July 23, 2024

HEART DISEASE DATASET ANALYSIS Attribute Information:

age sex chest pain type (4 values) resting blood pressure serum cholestoral in mg/dl fasting blood sugar > 120 mg/dl resting electrocardiographic results (values 0,1,2) maximum heart rate achieved exercise induced angina oldpeak = ST depression induced by exercise relative to rest the slope of the peak exercise ST segment number of major vessels (0-3) colored by flourosopy thal: 0 = normal; 1 = fixed defect; 2 = reversable defect Import libraries:

[1]:
```
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.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Load and Explore the Dataset

[2]:
```
# Load the dataset
df = pd.read_csv('Heart_Disease_Dataset.csv')

# Display the first few rows of the dataset
df.head()
```

[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

```
4   3     2       0
```

[3]:
```
# Check for missing values
print(df.isnull().sum())
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[4]:
```
# Summary statistics
print(df.describe())
```

```
                age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

             slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
```

```
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

[5]:
```
# Data types of each column
print(df.dtypes)
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

Data Preprocessing

[6]:
```
# Drop rows with missing values (if any)
df.dropna(inplace=True)
```

[7]:
```
# Feature Selection: Select features and target variable
X = df.drop(columns=['thalach'])
y = df['thalach']
```

[8]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[9]:
```
# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Development

[11]:
```
#Training different models
# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
y_pred_lr = lr.predict(X_test)
```

[12]:
```
# Decision Tree Regressor
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

[42]:
```
# Neural Network Regressor
mlp = MLPRegressor(hidden_layer_sizes=(100,), max_iter=5000)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
```

Model Evaluation

[31]:
```
# Function to evaluate models
def evaluate_model(y_true, y_pred):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    return mae, rmse, r2
```

[32]:
```
# Evaluate Linear Regression
mae_lr, rmse_lr, r2_lr = evaluate_model(y_test, y_pred_lr)
print(f'Linear Regression - MAE: {mae_lr}, RMSE: {rmse_lr}, R²: {r2_lr}')
```

```
Linear Regression - MAE: 14.127731819192395, RMSE: 18.54418175994919, R²:
0.36606111310330236
```

[33]:
```
# Evaluate Decision Tree Regressor
mae_dt, rmse_dt, r2_dt = evaluate_model(y_test, y_pred_dt)
print(f'Decision Tree Regressor - MAE: {mae_dt}, RMSE: {rmse_dt}, R²: {r2_dt}')
```

```
Decision Tree Regressor - MAE: 0.8048780487804879, RMSE: 5.576518714718866, R²:
0.942673111521245
```

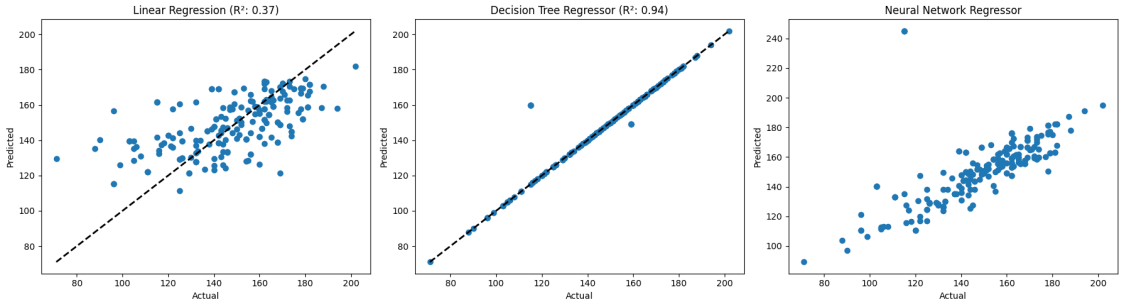
[43]:
```
# Evaluate Neural Network Regressor
mae_mlp, rmse_mlp, r2_mlp = evaluate_model(y_test, y_pred_mlp)
print(f'Neural Network Regressor - MAE: {mae_mlp}, RMSE: {rmse_mlp}, R²:␣
 ↪{r2_mlp}')
```

```
Neural Network Regressor - MAE: 9.486713369080174, RMSE: 18.87497007565954, R²:
0.34324318841945145
```

Visualization

[45]:
```
plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
plt.scatter(y_test, y_pred_lr)
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title(f'Linear Regression (R²: {r2_lr:.2f})')

plt.subplot(1, 3, 2)
plt.scatter(y_test, y_pred_dt)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',␣
 ↪lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title(f'Decision Tree Regressor (R²: {r2_dt:.2f})')

plt.subplot(1, 3, 3)
plt.scatter(y_test, y_pred_mlp)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Neural Network Regressor')

plt.tight_layout()
plt.show()
```



Conclusion

[50]:
```
# Conclusion based on evaluation metrics
print(f"Linear Regression - R²: {r2_lr:.2f}, MAE: {mae_lr:.2f}, RMSE: {rmse_lr:.
 ↪2f}")
print(f"Decision Tree Regressor - R²: {r2_dt:.2f}, MAE: {mae_dt:.2f}, RMSE:␣
 ↪{rmse_dt:.2f}")
print(f"Neural Network Regressor - R²: {r2_mlp:.2f}, MAE: {mae_mlp:.2f}, RMSE:␣
 ↪{rmse_mlp:.2f}")
```

```
Linear Regression - R²: 0.37, MAE: 14.13, RMSE: 18.54
Decision Tree Regressor - R²: 0.94, MAE: 0.80, RMSE: 5.58
Neural Network Regressor - R²: 0.34, MAE: 9.49, RMSE: 18.87
```

[48]:
```
accuracy_percentage_lr = r2_lr * 100
accuracy_percentage_dt = r2_dt * 100
accuracy_percentage_mlp = r2_mlp * 100

print(f'Linear Regression Accuracy: {accuracy_percentage_lr:.2f}%')
print(f'Decision Tree Regressor Accuracy: {accuracy_percentage_dt:.2f}%')
print(f'Decision Tree Regressor Accuracy: {accuracy_percentage_mlp:.2f}%')
```

```
Linear Regression Accuracy: 36.61%
Decision Tree Regressor Accuracy: 94.27%
Decision Tree Regressor Accuracy: 34.32%
```

[49]:
```
# Final thoughts
if r2_mlp > r2_lr and r2_mlp > r2_dt:
    print("The Neural Network Regressor performed the best in terms of R²␣
 ↪accuracy.")
elif r2_lr > r2_mlp and r2_lr > r2_dt:
    print("The Linear Regression performed the best in terms of R² accuracy.")
else:
    print("The Decision Tree Regressor performed the best in terms of R²␣
 ↪accuracy.")
```

```
The Decision Tree Regressor performed the best in terms of R² accuracy.
```

[ ]:
```

```