K-Nearest Neighbors Classifier

September 7, 2023

0.1 Algorithms - K- Nearest Neighbors Classifier: Predicting Heart Disease

The World Health Organization (WHO) estimates that 17.9 million people die every year because of cardiovascular diseases (CVDs).

There are multiple risk factors that could contribute to CVD in an individual such as unhealthy diet, lack of physical activity or mental illnesses. Being able to identify these risk factors in individuals early on could help prevent a lot of premature deaths.

In this project, I will use the Kaggle dataset (https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction) and build a K-Nearest Neighbors classifier to accurately predict the likelihood of a patient having a heart disease in the future.

0.2 EDA: Descriptive Statistics

- Age: age of the patient (years)
- Sex: sex of the patient (M: Male, F: Female)
- ChestPainType: chest pain type (TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic)
- RestingBP: resting blood pressure (mm Hg)
- Cholesterol: serum cholesterol (mm/dl)
- FastingBS: fasting blood sugar (1: if FastingBS > 120 mg/dl, 0: otherwise)
- RestingECG: resting electrocardiogram results (Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria)
- MaxHR: maximum heart rate achieved (Numeric value between 60 and 202)
- ExerciseAngina: exercise-induced angina (Y: Yes, N: No)
- Oldpeak: oldpeak = ST (Numeric value measured in depression)
- ST_Slope: the slope of the peak exercise ST segment (Up: upsloping, Flat: flat, Down: downsloping)

- HeartDisease: output class (1: heart disease, 0: Normal)

[1]:
```
import pandas as pd
import numpy as np

df = pd.read_csv("heart_disease_prediction.csv")
```

[2]:
```
df.head()
```

[2]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

[3]:
```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

There aren't any missing values in the dataset.

[4]:
```
print(df.dtypes)
df.dtypes.value_counts()
```

```
Age                 int64
Sex                object
ChestPainType      object
```

```
RestingBP            int64
Cholesterol          int64
FastingBS            int64
RestingECG          object
MaxHR                int64
ExerciseAngina      object
Oldpeak            float64
ST_Slope            object
HeartDisease         int64
dtype: object
```

[4]:
```
int64      6
object     5
float64    1
dtype: int64
```

7 features in total are numerical while 5 are categorical. However, two of the numerical features, FastingBS and HeartDisease are categorical as well.

I will focus on the numerical variables first.

[5]:
```python
data_summary = df.describe()
print(data_summary)
```

```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000
mean    53.510893  132.396514   198.799564    0.233115  136.809368
std      9.432617   18.514154   109.384145    0.423046   25.460334
min     28.000000    0.000000     0.000000    0.000000   60.000000
25%     47.000000  120.000000   173.250000    0.000000  120.000000
50%     54.000000  130.000000   223.000000    0.000000  138.000000
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

From the table above, I can observe that:

- The average age of patients is ~53 years.
- The median for Cholesterol is higher than its mean by roughly 25 mm/dl, indicating that it could be a left-skewed distribution with a possibility of outliers skewing the distribution.
- RestingBP and Cholesterol have a minimum value of zero.

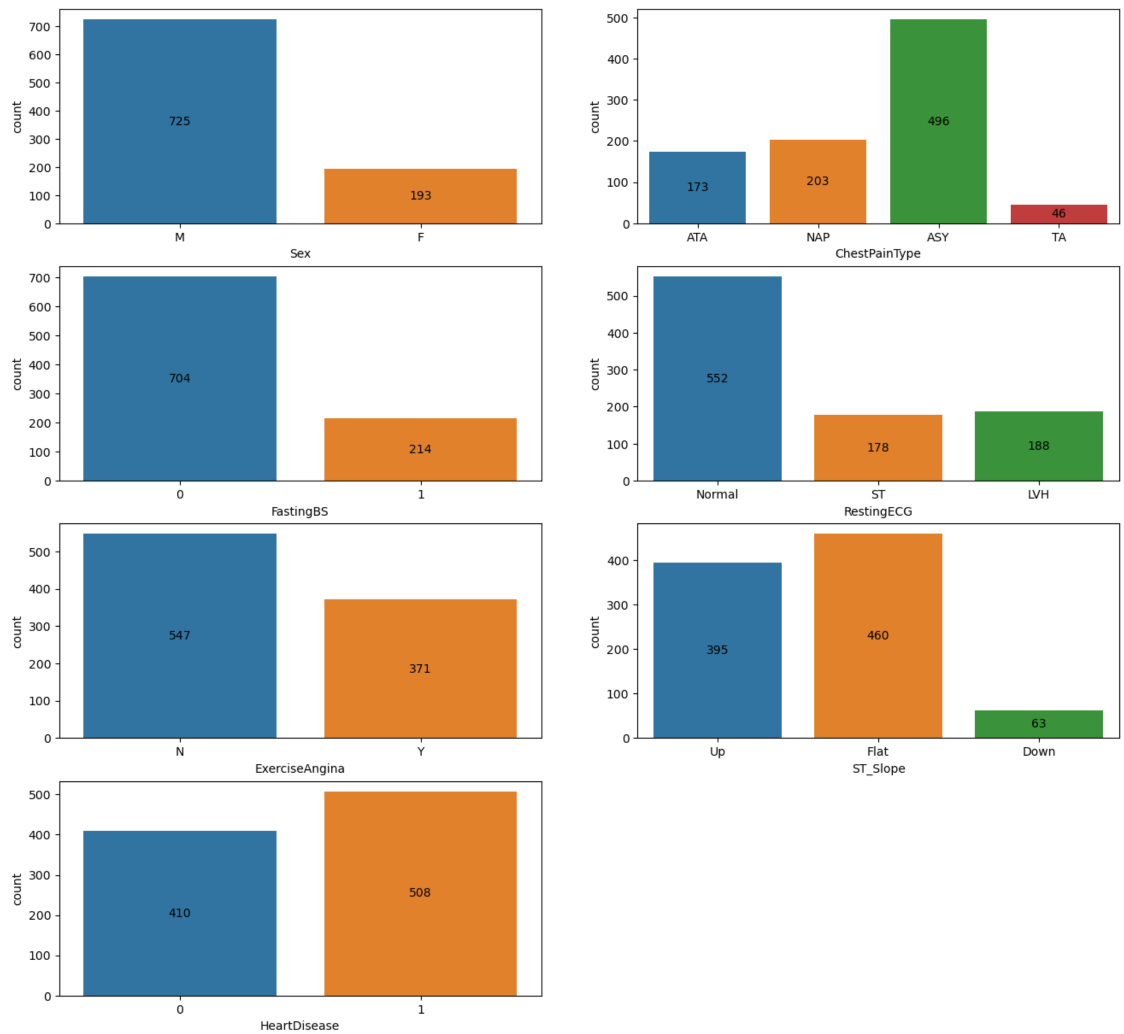
- There aren't any missing values in these columns,

RestingBP can't be 0. And, as per the American Heart Association, serum cholesterol is a composite of different measurements. So, it is unlikely that Cholesterol would be 0 as well. I will have to clean both of these up later.

0.3 EDA: Categorical Data

Next, I will look at the categorical variables using grouped bar plots.

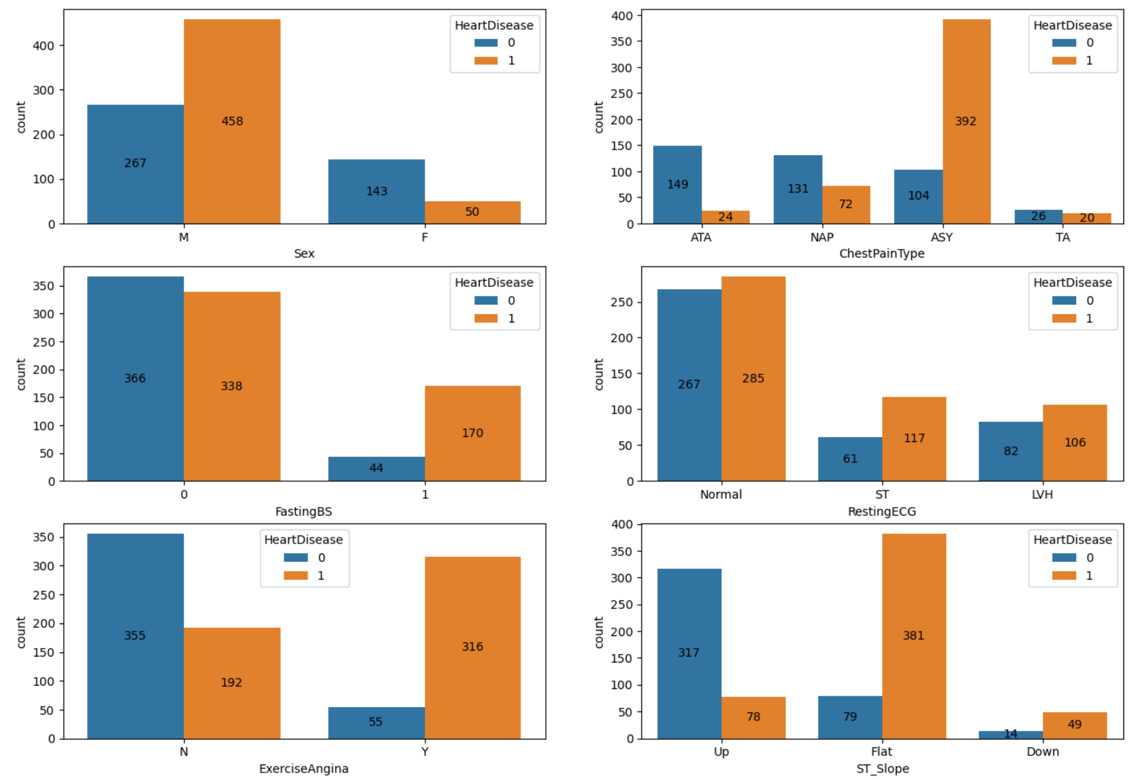
[6]:
```
import matplotlib.pyplot as plt
import seaborn as sns


categorical_cols = ["Sex", "ChestPainType", "FastingBS", "RestingECG",␣
↪"ExerciseAngina", "ST_Slope", "HeartDisease"]

fig = plt.figure(figsize=(16,15))

for idx, col in enumerate(categorical_cols):
    ax = plt.subplot(4, 2, idx+1)
    sns.countplot(x=df[col], ax=ax)
    # add data labels to each bar
    for container in ax.containers:
        ax.bar_label(container, label_type="center")
```



The dataset is highly skewed towards male patients. There are 725 male patients and 193 female patients. This could potentially induce a bias in the model. - 496 patients had ASY (asymptotic) chest pain type. - 552 patients had a normal restin ECG. - 704 patients had blood sugar lower than 120 mg/dl

Grouping these by HeartDisease will give me a better idea about the data distribution.

[7]:
```
fig = plt.figure(figsize=(16,15))

for idx, col in enumerate(categorical_cols[:-1]):
    ax = plt.subplot(4, 2, idx+1)
    # group by HeartDisease
    sns.countplot(x=df[col], hue=df["HeartDisease"], ax=ax)
    # add data labels to each bar
    for container in ax.containers:
        ax.bar_label(container, label_type="center")
```



I can further notice how skewed the dataset is towards male patients. Only 50 female patients in the dataset have been diagnosed with heart disease.

A significant number of patients, 392, diagnosed with heart disease have asymptomatic (ASY) chest pain. While chest pain could be a relevant feature for the model, asymptomatic implies that those patients who had a heart disease did not have chest pain as a symptom.

A high number (170) of patients with blood sugar greater than 120 mg/dl were diagnosed with heart disease in relation to those who were not diagnosed as such.

Out of all patients who had an exercise-induced angina, 316 were diagnosed with a heart disease.

Out of all patients with a flat ST slope, 381 were diagnosed with a heart disease.

Looking at the data distribution from the above plots, I can start to identify some features that could be relevant. I will clean up the dataset a bit first before narrowing down on the features.

[8]:
```
sex_percentage = df['Sex'].value_counts(normalize=True) * 100
print(sex_percentage)
```

```
M    78.976035
F    21.023965
Name: Sex, dtype: float64
```

I can see that the data consists of 78% males and 21% females.

0.4 Data Cleaning

I identified that there are no missing values. However, as noticed earlier, a couple of columns have 0 values which don't make sense. I will replace these 0 values with the median of their corresponding columns.

[9]:
```
# Replace 0 values in the 'RestingBP' column with the median of the column
median_resting_bp = df['RestingBP'].median()
df['RestingBP'].replace(0, median_resting_bp, inplace=True)

# Replace 0 values in the 'Cholesterol' column with the median of the column
median_cholesterol = df['Cholesterol'].median()
df['Cholesterol'].replace(0, median_cholesterol, inplace=True)
```

[10]:
```
df[["Cholesterol", "RestingBP"]].describe()
```

[10]:

	Cholesterol	RestingBP
count	918.000000	918.000000
mean	240.581699	132.538126
std	53.982967	17.990127
min	85.000000	80.000000
25%	214.000000	120.000000
50%	223.000000	130.000000
75%	267.000000	140.000000
max	603.000000	200.000000

The minimum values for both have changed. There are no more zero values in either of those.

0.5 Feature Selection

Now that I have cleaned the data, I will select features for the model. Thanks to our EDA and a general understanding of the features, I can identify some of the features that to start with:

- Age
- Sex
- ChestPainType
- Cholesterol
- FastingBS

I will explore how the columns correlate to one another. Before attempting that, I will convert the categorical columns into dummy variables.

[11]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

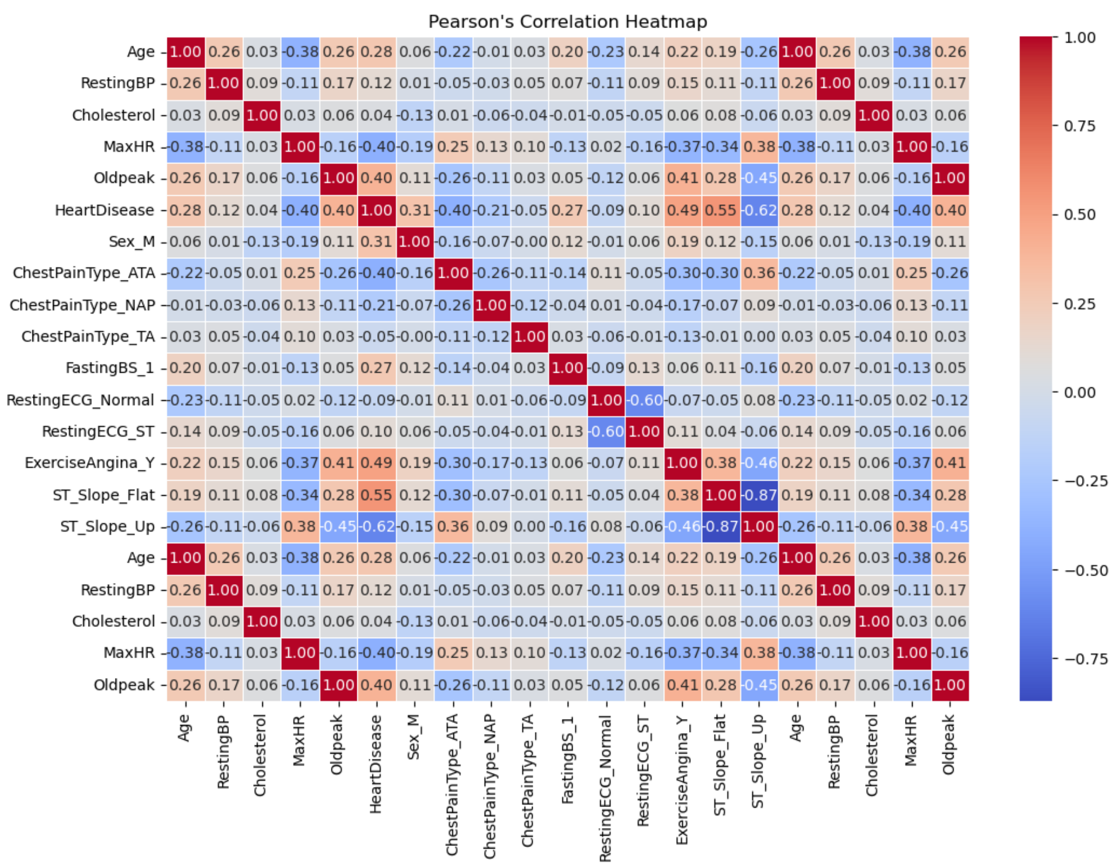
```python
from sklearn.metrics import accuracy_score, classification_report
```

[12]:

```python
categorical_cols = ["Sex", "ChestPainType", "FastingBS", "RestingECG",␣
 ↪"ExerciseAngina", "ST_Slope"]
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# Numeric columns
numeric_cols = ["Age", "RestingBP", "Cholesterol", "MaxHR", "Oldpeak"]

# Combine numeric and dummy variables into a single DataFrame
df_clean = pd.concat([df_encoded, df[numeric_cols]], axis=1)
```

[13]:

```python
# Calculate the correlation matrix
correlation_matrix = df_clean.corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
 ↪linewidths=0.5)
plt.title("Pearson's Correlation Heatmap")
plt.show()
```



From the correlation heatmap, I can identify the following features to be positively correlated (correlation coefficient greater than 0.3) to HeartDisease:

- Oldpeak
- MaxHR
- ChestPainType_ATA
- ExerciseAngina_Y
- ST_Slope_Flat
- ST_Slope_Up

The correlation coefficient threshold was chosen arbitrarily. Surprisingly, Cholesterol is not strongly correlated to HeartDisease. I will consider ignoring the feature for now.

Given everything I have attempted so far, I can narrow down the features to the following:

- Oldpeak
- Sex_M

Note: Sex_M has a relatively low value for the coefficient, but given what I observed in EDA, I will also take it into account. - ExerciseAngina_Y - ST_Slope_Flat - ST_Slope_Up

0.6 Building a Classifier with Multiple Features

[14]:
```
# Define your feature columns and target variable
selected_features = ["Oldpeak", "Sex_M", "ExerciseAngina_Y", "ST_Slope_Flat",␣
↪"ST_Slope_Up"]
X = df_clean[selected_features]
y = df_clean["HeartDisease"]

# 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)
```

[15]:
```
# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[16]:
```
from sklearn.model_selection import GridSearchCV

knn = KNeighborsClassifier()
param_grid = {"n_neighbors": range(1, 21)}
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

best_k = grid_search.best_params_["n_neighbors"]
```

[17]:
```
# Create and train the k-NN classifier using the best value of k
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)
```

[17]:
```
KNeighborsClassifier(n_neighbors=12)
```

[18]:
```
# Make predictions on the test set
y_pred = knn.predict(X_test_scaled)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

```
Accuracy: 0.78
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.79      0.75        77
           1       0.84      0.78      0.81       107

    accuracy                           0.78       184
   macro avg       0.78      0.78      0.78       184
weighted avg       0.79      0.78      0.78       184
```

The overall accuracy of this model was 78%. Now, I will try to improve this rate.

0.7 Hyperparameter Optimization: Trying Different Distance Metrics

[19]:
```
# Define a list of distance metrics to try
distance_metrics = ['euclidean', 'manhattan', 'chebyshev', 'minkowski']

# Initialize variables to store the best accuracy and corresponding metric
best_accuracy = 0
best_metric = None

# Iterate over the distance metrics
for metric in distance_metrics:
    # Create and train the k-NN classifier with the current metric
    knn = KNeighborsClassifier(n_neighbors=best_k, metric=metric)
    knn.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    y_pred = knn.predict(X_test_scaled)
```

```
    # Calculate the accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy for the current metric
    print(f"Accuracy with {metric} metric: {accuracy:.2f}")

    # Update the best accuracy and metric if the current metric performs better
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_metric = metric

# Print the best metric and accuracy
print(f"Best Metric: {best_metric}")
print(f"Best Accuracy: {best_accuracy:.2f}")
```

```
Accuracy with euclidean metric: 0.78
Accuracy with manhattan metric: 0.78
Accuracy with chebyshev metric: 0.78
Accuracy with minkowski metric: 0.78
Best Metric: euclidean
Best Accuracy: 0.78
```

Trying different distance metric didn't change the accuracy at all. This could be because the numerical columns were scaled.

0.8 Trying Min-Max Scaling to Increase Accuracy

[20]:
```
from sklearn.preprocessing import MinMaxScaler

# Feature Scaling with Min-Max Scaling
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[21]:
```
# Create and Train the k-NN Classifier
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test_scaled)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
```

```python
print(report)
```

```
Accuracy: 0.77
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.87      0.76        77
           1       0.88      0.70      0.78       107

    accuracy                           0.77       184
   macro avg       0.78      0.79      0.77       184
weighted avg       0.80      0.77      0.77       184
```

The accuracy decreased to 77% with the min-max scaler.

0.9 Trying to Change K (Number of Neighbors)

[22]:
```python
# Define a list of distance metrics to try
distance_metrics = ['euclidean']

# Initialize variables to store the best accuracy and corresponding metric
best_accuracy = 0
best_metric = None

# Iterate over the distance metrics
for metric in distance_metrics:
    # Create and train the k-NN classifier with the current metric
    knn = KNeighborsClassifier(n_neighbors=17, metric=metric)
    knn.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    y_pred = knn.predict(X_test_scaled)

    # Calculate the accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy for the current metric
    print(f"Accuracy with {metric} metric: {accuracy:.2f}")

    # Update the best accuracy and metric if the current metric performs better
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_metric = metric

# Print the best metric and accuracy
print(f"Metric: {best_metric}")
print(f"Accuracy: {best_accuracy:.2f}")
```

```
Accuracy with euclidean metric: 0.81
Metric: euclidean
Accuracy: 0.81
```

This method was successful in increasing the model's accuracy. I used the Euclidean distance metric and tried different values of K. With K = 17, I was able to get an accuracy of 81%.

0.10 Summary

My model got an accuracy of ~81%. This means that the model is likely to correctly predict whether a patient is at risk for a heart disease ~81% of the time.

I found that the above datasets have a significantly higher number of male patients than female ones. This could present a bias because of this imbalance in the dataset and can see its potential impacts on the model. If the test dataset doesn't have that many female patients and the model was trained on a dataset with more male patients, then it is understandable it has better accuracy on the test set. Of course, there could be other factors contributing to this discrepancy.

My final model was trained using the following features:

- Oldpeak
- Sex_M
- ExerciseAngina_Y
- ST_Slope_Flat
- ST_Slope_Up

and had a test set accuracy of 81%. However, given the limitations of the data this accuracy might not be indicative of a well performing model.

There are quite a few things can be done next to get better results:

- Try out different features.
- Expand the grid search parameters to identify more optimal hyperparameters.
- Explore other algorithms that might perform better than k-NN.
- Try and collect more data.

[ ]:
```

```