Heart Disease Analysis

Introduction

Heart disease, also known as cardiovascular disease, is a leading cause of morbidity and mortality worldwide. It encompasses a range of conditions affecting the heart and blood vessels, including coronary artery disease, heart failure, arrhythmias, and valvular heart diseases. Heart disease affects people of all ages and backgrounds, and its prevalence has been steadily increasing due to various risk factors, including sedentary lifestyles, poor dietary habits, smoking, and stress.

Understanding the underlying factors and risk predictors associated with heart disease is crucial for early detection, prevention, and effective management. Analyzing and interpreting relevant data can provide valuable insights into the prevalence, patterns, and potential interventions to combat this significant public health concern.

Data Source

The dataset used in this article is the Cleveland Heart Disease dataset taken from the UCI repository. Link: http://archive.ics.uci.edu/dataset/45/heart+disease

Feature Description:

1. **age** : age in years
2. **sex** : (1 = male; 0 = female)
3. **cp** : chest pain type
   - Value 1: typical angina
   - Value 2: atypical angina
   - Value 3: non-anginal pain
   - Value 4: asymptomatic
4. **trestbps**: displays the resting blood pressure value of an individual in mmHg (unit)
5. **chol** : serum cholestoral
6. **fbs** : fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
7. **restecg**: resting electrocardiographic results
   - Value 0: normal
   - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. **thalach**: maximum heart rate achieved
9. **exang** : exercise induced angina (1 = yes; 0 = no)
10. **oldpeak** : ST depression induced by exercise relative to rest
11. **slope** : the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. **ca** : number of major vessels (0-3) colored by flourosopy
13. **thal** : 3 = normal; 6 = fixed defect; 7 = reversable defect
14. **presence** : the presence of heart disease in the patient, 0 = absence, 1 = present

In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from warnings import simplefilter
```

In [2]:
```python
# reading csv files
df =  pd.read_csv('processed.cleveland.data',header = None)
df.columns = ["age", "sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","slope","ca","thal","presence"]
df
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	presence
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	2
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	3
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	?	3.0	0

303 rows × 14 columns

Check missing values

There are no missing values in the dataframe, however, at row 302, there is a "?" as value from column ca, and the Dtype for column ca is an object, we assume that "?" is a replacement for a missing value, same goes for columm thal. We can replace the missing values by replacing it with the mean of the column.

In [3]:
```python
df.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  presence  303 non-null    int64
dtypes: float64(11), int64(1), object(2)
memory usage: 33.3+ KB
```

In [4]:
```python
df['thal'] = df['thal'].replace('?', np.nan).astype(float)
df['thal'] = df['thal'].fillna(df['thal'].mean()).round(1)
df['ca'] = df['ca'].replace('?', np.nan).astype(float)
df['ca'] = df['ca'].fillna(df['ca'].mean()).round(1)
```

Now the missing values are placement with the mean of the column.

In [5]:
```python
df
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	presence
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	2
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	3
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	0.7	3.0	0

303 rows × 14 columns

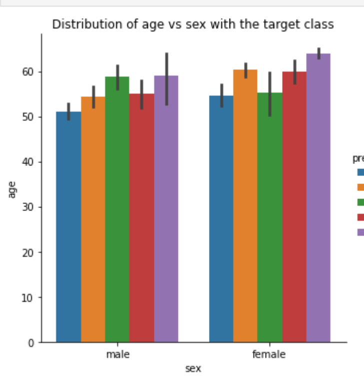
In [6]:
```python
df.info()
```

```
<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    float64
 12  thal      303 non-null    float64
 13  presence  303 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 33.3 KB
```

Age vs Sex

In [7]:
```python
# barplot of age vs sex
df['sex'] = df['sex'].map({0: 'female', 1: 'male'})


sns.catplot(kind='bar', data=df, y='age', x='sex', hue='presence')
plt.title('Distribution of age vs sex with the target class')
plt.show()
```



Chest Pain Types vs Heart Disease Presence

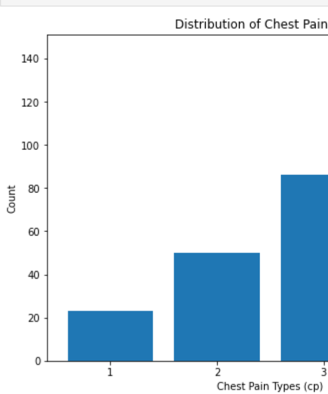
From the following visualization of distribution of chest pain types and heart disease presence. We can see that most patient has chest pain type of value 4 = asymptomatic, which means no chest pain, and most patient has heart disease presence of value 0, meaning no heart disease presence. We can assume that patient has no chest pain also has no heart disease presence.

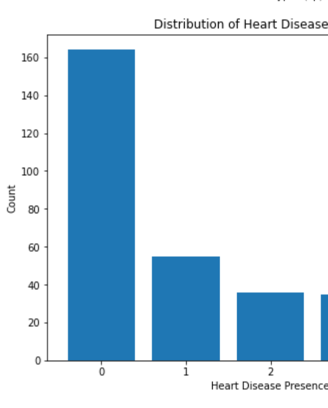
In [8]:
```python
cp_counts = df['cp'].value_counts()

# Create a bar chart for the distribution of chest pain types
plt.figure(figsize=(8, 6))
plt.bar(cp_counts.index, cp_counts.values)
plt.xlabel('Chest Pain Types (cp)')
plt.ylabel('Count')
plt.title('Distribution of Chest Pain Types')
plt.xticks(cp_counts.index)
plt.show()

# Count the occurrences of heart disease presence (presence)
presence_counts = df['presence'].value_counts()

# Create a bar chart for the distribution of heart disease presence
plt.figure(figsize=(8, 6))
plt.bar(presence_counts.index, presence_counts.values)
plt.xlabel('Heart Disease Presence')
plt.ylabel('Count')
plt.title('Distribution of Heart Disease Presence')
plt.xticks(presence_counts.index)
plt.show()
```





Data Preprocessing

In [9]:
```python
#we made all presence level 1-4 to value 1 to indicate that there is a heart disease
df['presence'] = df.presence.map({0: 0, 1: 1, 2: 1, 3: 1, 4: 1})
```

In [10]:
```python
#convert to numercial
df['sex'] = df.sex.map({'female': 0, 'male': 1})
```

In [11]:
```python
#set features = X and target = y
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [12]:
```python
#split the data, 80% training and 20% testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

SVM

In [20]:
```python
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler

# Initialize SVM classifier with a smaller C value
classifier = SVC(kernel='rbf', C=0.1)

# Scale the features to [0, 1] range
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform feature selection using SelectKBest
selector = SelectKBest(chi2, k=13)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Perform k-fold cross-validation on the selected features
cv_accuracy = cross_val_score(classifier, X_train_selected, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy: {:.2f}%".format(cv_accuracy.mean() * 100))

# Fit the model on the training data with the selected features
classifier.fit(X_train_selected, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test_selected)

# Confusion matrix for the test set
from sklearn.metrics import confusion_matrix
cm_test = confusion_matrix(y_pred, y_test)

# Calculate accuracy for training and test sets
y_pred_train = classifier.predict(X_train_selected)
cm_train = confusion_matrix(y_pred_train, y_train)
print()
print('Accuracy for training set for SVM = {:.2f}%'.format((cm_train[0][0] + cm_train[1][1]) / len(y_train) * 100))
print('Accuracy for test set for SVM = {:.2f}%'.format((cm_test[0][0] + cm_test[1][1]) / len(y_test) * 100))
```

```
Cross-Validation Accuracy: 83.47%

Accuracy for training set for SVM = 85.54%
Accuracy for test set for SVM = 83.61%
```

Logistic Regression

In [18]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

# Initialize the Logistic Regression classifier with L2 regularization
# Set C to a smaller value (e.g., 0.1) to increase regularization strength
classifier = LogisticRegression(solver='lbfgs', max_iter=1000, C=0.1)

# Scale the features to [0, 1] range
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform k-fold cross-validation on the scaled features
cv_accuracy = cross_val_score(classifier, X_train_scaled, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy: {:.2f}%".format(cv_accuracy.mean() * 100))

# Fit the model on the scaled training data
classifier.fit(X_train_scaled, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test_scaled)

# Confusion matrix for the test set
cm_test = confusion_matrix(y_pred, y_test)

# Calculate accuracy for training and test sets
y_pred_train = classifier.predict(X_train_scaled)
cm_train = confusion_matrix(y_pred_train, y_train)
print()
print('Accuracy for training set for Logistic Regression = {:.2f}%'.format((cm_train[0][0] + cm_train[1][1]) / len(y_train) * 100))
print('Accuracy for test set for Logistic Regression = {:.2f}%'.format((cm_test[0][0] + cm_test[1][1]) / len(y_test) * 100))
```

```
Cross-Validation Accuracy: 83.48%

Accuracy for training set for Logistic Regression = 85.95%
Accuracy for test set for Logistic Regression = 81.97%
```

Naive Bayes

In [14]:
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest, chi2

# Initialize the Naive Bayes classifier
classifier = GaussianNB()

# Scale the features to [0, 1] range
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform feature selection using SelectKBest
selector = SelectKBest(chi2, k=9)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Perform k-fold cross-validation on the selected features
cv_accuracy = cross_val_score(classifier, X_train_selected, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy: {:.2f}%".format(cv_accuracy.mean() * 100))

# Fit the model on the scaled and selected training data
classifier.fit(X_train_selected, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test_selected)

# Confusion matrix for the test set
cm_test = confusion_matrix(y_pred, y_test)

# Calculate accuracy for training and test sets
y_pred_train = classifier.predict(X_train_selected)
cm_train = confusion_matrix(y_pred_train, y_train)
print()
print('Accuracy for training set for Naive Bayes = {:.2f}%'.format((cm_train[0][0] + cm_train[1][1]) / len(y_train) * 100))
print('Accuracy for test set for Naive Bayes = {:.2f}%'.format((cm_test[0][0] + cm_test[1][1]) / len(y_test) * 100))
```

```
Cross-Validation Accuracy: 84.73%

Accuracy for training set for Naive Bayes = 85.54%
Accuracy for test set for Naive Bayes = 81.97%
```

Decision Tree

In [15]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler

# Initialize the Decision Tree classifier with constraints
classifier = DecisionTreeClassifier(max_depth=3, min_samples_split=5)

# Scale the features to [0, 1] range
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform feature selection using SelectKBest
selector = SelectKBest(chi2, k=9)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Perform k-fold cross-validation on the selected features
cv_accuracy = cross_val_score(classifier, X_train_selected, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy: {:.2f}%".format(cv_accuracy.mean() * 100))

# Fit the model on the scaled and selected training data
classifier.fit(X_train_selected, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test_selected)

# Confusion matrix for the test set
cm_test = confusion_matrix(y_pred, y_test)

# Calculate accuracy for training and test sets
y_pred_train = classifier.predict(X_train_selected)
cm_train = confusion_matrix(y_pred_train, y_train)
print()
print('Accuracy for training set for Decision Tree = {:.2f}%'.format((cm_train[0][0] + cm_train[1][1]) / len(y_train) * 100))
print('Accuracy for test set for Decision Tree = {:.2f}%'.format((cm_test[0][0] + cm_test[1][1]) / len(y_test) * 100))
```

```
Cross-Validation Accuracy: 79.35%

Accuracy for training set for Decision Tree = 85.54%
Accuracy for test set for Decision Tree = 78.69%
```

Random Forest

In [16]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

# Initialize the Random Forest classifier with more estimators and limited tree depth
classifier = RandomForestClassifier(n_estimators=100, max_depth=5, min_samples_leaf=5)

# Scale the features to [0, 1] range
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Perform k-fold cross-validation on the Random Forest classifier
cv_accuracy = cross_val_score(classifier, X_train_scaled, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy: {:.2f}%".format(cv_accuracy.mean() * 100))

# Fit the model on the scaled training data
classifier.fit(X_train_scaled, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test_scaled)

# Confusion matrix for the test set
cm_test = confusion_matrix(y_pred, y_test)

# Calculate accuracy for training and test sets
y_pred_train = classifier.predict(X_train_scaled)
cm_train = confusion_matrix(y_pred_train, y_train)
print()
print('Accuracy for training set for Random Forest = {:.2f}%'.format((cm_train[0][0] + cm_train[1][1]) / len(y_train) * 100))
print('Accuracy for test set for Random Forest = {:.2f}%'.format((cm_test[0][0] + cm_test[1][1]) / len(y_test) * 100))
```

```
Cross-Validation Accuracy: 83.48%

Accuracy for training set for Random Forest = 90.50%
Accuracy for test set for Random Forest = 80.33%
```