The objective of this project is to develop a predictive model that can assist in the early detection of heart disease in individuals based on various health parameters and clinical test results

Lets import the important libraries

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

import warnings
warnings.filterwarnings("ignore")
```

Now lets read and understand the data

In [2]:
```
df = pd.read_csv("heart.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	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

Dataset Information

This dataset contains various attributes related to patients' health parameters and tests, aiming to predict the presence of heart disease.

- **age**: The age of the patient in years.
- **sex**: Binary variable denoting the sex of the patient (1 = male, 0 = female).
- **cp**: Type of chest pain experienced by the patient categorized as:
  - 1 = Typical angina
  - 2 = Atypical angina
  - 3 = Non-anginal pain
  - 4 = Asymptomatic
- **trestbps**: Resting blood pressure of the patient measured in mm Hg.
- **chol**: Serum cholesterol level of the patient measured in mg/dl.
- **fbs**: Fasting blood sugar level of the patient:
  - 1 = High
  - 0 = Low
- **restecg**: Resting electrocardiographic results classified as:
  - 0 = Normal
  - 1 = ST-T wave abnormality
  - 2 = Left ventricular hypertrophy
- **thalach**: Maximum heart rate achieved by the patient during exercise.
- **exang**: Presence of exercise-induced angina:
  - 1 = Yes
  - 0 = No
- **oldpeak**: ST depression induced by exercise relative to rest.
- **slope**: The slope of the ST segment during peak exercise:
  - 1 = Upsloping
  - 2 = Flat
  - 3 = Downsloping

- **ca**: The number of major vessels colored by fluoroscopy (ranging from 0 to 3).
- **thal**: The type of thallium scan performed on the patient:
  - 1 = Fixed defect
  - 2 = Reversible defect
  - 3 = Normal
- **target**: The presence of heart disease in the patient:
  - 0 = No disease
  - 1 = Disease present

In [4]:
```
#check null values in the df
df.isnull().sum()
```

Out[4]:
```
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
```

In [5]:
```
df.info()
```

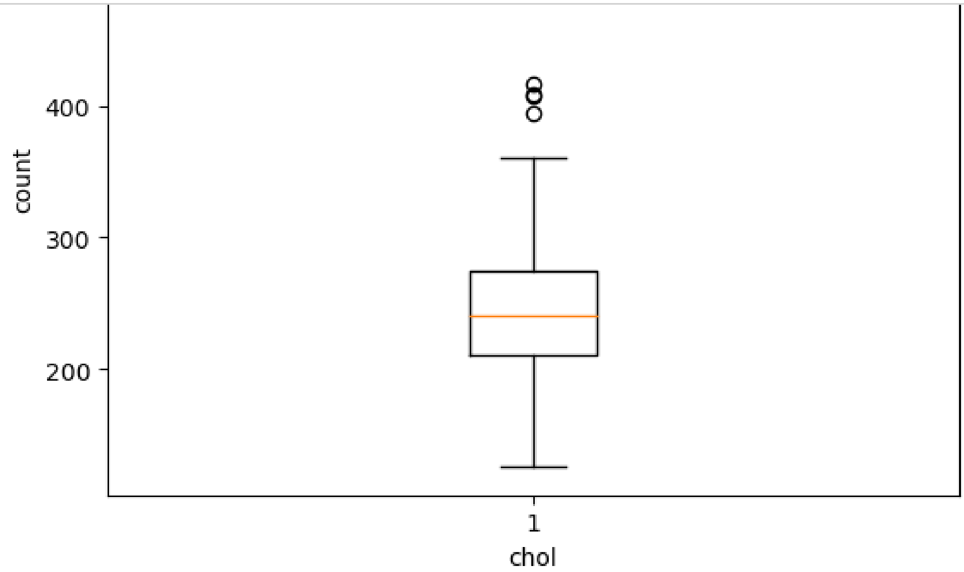
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
1
```

In [7]:
```
df.drop_duplicates(inplace=True)
```

In [8]:
```
#Lets check for the outliers in the data set using box plot
for i in df.columns:
    if ((df[i].dtype != object) & (i!='target')):
        plt.boxplot(df[i])
        plt.xlabel(i)
        plt.ylabel("count")
        plt.show()
```



In [9]:
```
outlier_list = ['trestbps', 'chol', 'thalach', 'oldpeak']  #this are the columns with outlier
```

The Interquartile Range (IQR) method is a way to detect and remove outliers from a dataset. Outliers are data points that significantly differ from other observations in a dataset and can negatively impact the performance of machine learning models. The IQR method involves the following steps to detect and remove outliers:

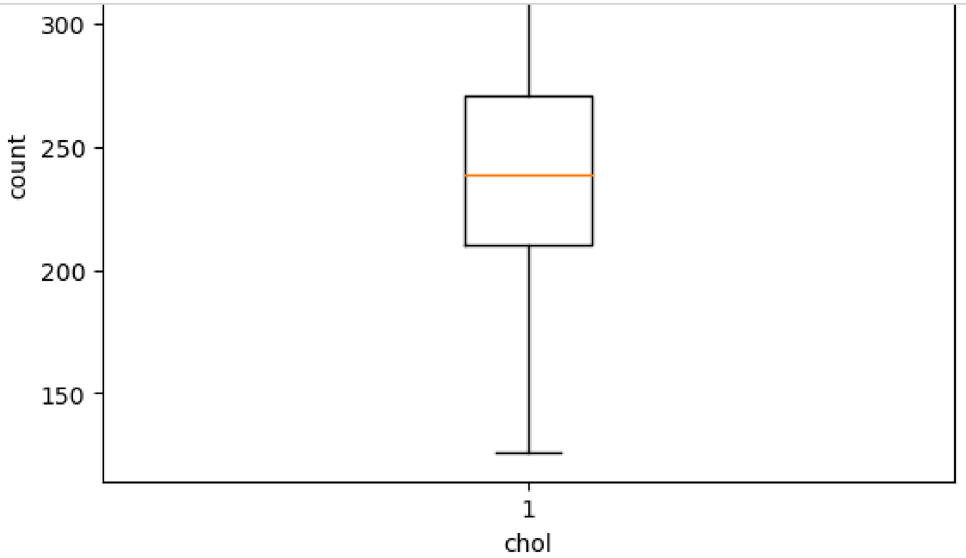
Calculate the IQR (Interquartile Range): The IQR is a measure of statistical dispersion and is calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. IQR = Q 3 − Q 1 IQR=Q3−Q1

Identify Outliers: Data points that fall below Q 1 − 1.5 × IQR Q1−1.5×IQR or above Q 3 + 1.5 × IQR Q3+1.5×IQR are considered outliers.

Remove Outliers: Remove these identified outliers from the dataset.

In [10]:
```
#removal of outliers

for i in outlier_list:

    Q1 = df[i].quantile(0.25) #Q1 is at 25%
    Q3 = df[i].quantile(0.75)

    IQR = Q3-Q1

    df = df[(df[i]<= Q3+1.5*IQR) & (df[i]>= Q1-1.5*IQR)]
```

In [11]:
```
#Lets check for the outliers in the data set using box plot and it should have been removed
for i in df.columns:
    if ((df[i].dtype != object) & (i!='target')):
        plt.boxplot(df[i])
        plt.xlabel(i)
        plt.ylabel("count")
        plt.show()
```



In [12]:
```
df.shape #outlier has been removed now the data size has changed from 303 -> 283
```

Out[12]: (283, 14)

In [13]:
```
#Lets build the model for the prediction
```

In [14]:
```
#split the data into X and y
X = df.iloc[:, :-1] #all rows and columns except target
y = df['target']
```

In [15]:
```
X
```

Out[15]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

283 rows × 13 columns

In [16]:
```
y
```

Out[16]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 283, dtype: int64
```

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=99)
```

In [19]:
```
x_train
```

Out[19]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
177	64	1	2	140	335	0	1	158	0	0.0	2	0	2
108	50	0	1	120	244	0	1	162	0	1.1	2	0	2
50	51	0	2	130	256	0	0	149	0	0.5	2	0	2
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2
143	67	0	0	106	223	0	1	142	0	0.3	2	2	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
210	57	1	2	128	229	0	0	150	0	0.4	1	1	3
175	40	1	0	110	167	0	0	114	1	2.0	1	0	3
192	54	1	0	120	188	0	1	113	0	1.4	1	1	3
37	54	1	2	150	232	0	0	165	0	1.6	2	0	3
135	49	0	0	130	269	0	1	163	0	0.0	2	0	2

198 rows × 13 columns

In [20]:
```
y_train
```

Out[20]:
```
177    0
108    1
50     1
17     1
143    1
      ..
210    0
175    0
192    0
37     1
135    1
Name: target, Length: 198, dtype: int64
```

In [21]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [22]:
```
dt = DecisionTreeClassifier(criterion='entropy', min_samples_split=2)
```

In [23]:
```
dt = dt.fit(x_train, y_train)
```

In [24]:
```
y_pred = dt.predict(x_test)
```

In [25]:
```
y_pred
```

Out[25]:
```
array([0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
      dtype=int64)
```

In [26]:
```
from sklearn.metrics import accuracy_score
```

In [27]:
```
accuracy_score(y_test, y_pred)
```

Out[27]:
```
0.7647058823529411
```

Lets check for the overfitting in the model

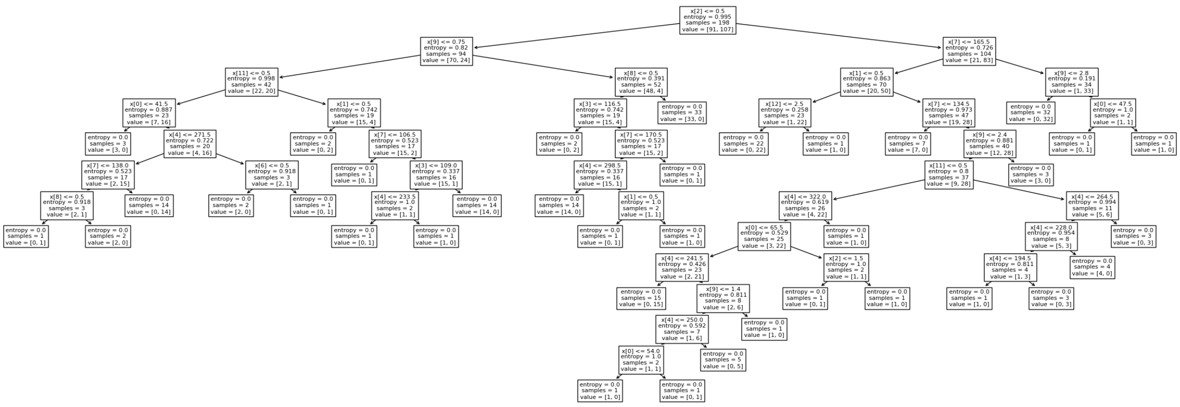
In [28]:
```
y_pred_train = dt.predict(x_train)
```

In [29]:
```
accuracy_score(y_train,y_pred_train)
```

Out[29]:
```
1.0
```

High accuracy on the training set (Y_train, y_pred_train) and significantly lower accuracy on the test set (Y_test, y_pred_test) can indicate overfitting.

In [30]:
```
#plotting the decision tree

from sklearn import tree
plt.figure(figsize=(30,10))
tree.plot_tree(dt)
plt.show()
plt.savefig('decision_tree.png')
```



```
<Figure size 640x480 with 0 Axes>
```

**Hyperparameter tuning is a crucial step in optimizing machine learning models like decision trees. Decision trees have several hyperparameters that can be adjusted to improve the model's performance. Some common hyperparameters in decision trees include:

**Maximum Depth (max_depth): It defines the maximum depth of the tree. A deeper tree may overfit the data, while a shallower tree might not capture all the patterns in the data.

**Minimum Samples Split (min_samples_split): This parameter defines the minimum number of samples required to split an internal node. A higher value can prevent overfitting.

**Minimum Samples Leaf (min_samples_leaf): It sets the minimum number of samples required to be at a leaf node. It helps in controlling the size of the tree and prevents overfitting.

**Maximum Features (max_features): It determines the number of features to consider when looking for the best split.

**Criterion: It defines the function to measure the quality of a split. Common criteria are 'gini' for the Gini impurity and 'entropy' for information gain.

**To tune these hyperparameters effectively, you can use techniques like Grid Search or Randomized Search, typically available in Python through libraries such as scikit-learn.

**Grid Search: Grid Search is an exhaustive search over a specified parameter grid. It tries every combination of hyperparameters in a grid to find the best performing combination.

In [31]:
```
#Hyperparamter tuning using GridSearchCV

from sklearn.model_selection import GridSearchCV
```

In [32]:
```
#Define the parameter grid
param_grid = {
    'criterion': ['gini', "entropy"],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1,2,4]
}
```

In [33]:
```
dt = DecisionTreeClassifier()
```

In [34]:
```
#GridSearchCV

grid_search = GridSearchCV(dt,param_grid, cv=5)
```

In [35]:
```
grid_search.fit(X, y)
```

Out[35]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [36]:
```
#best hyperparamters
best_params = grid_search.best_params_
best_params
```

Out[36]:
```
{'criterion': 'entropy',
 'max_depth': 5,
 'min_samples_leaf': 2,
 'min_samples_split': 2}
```

In [37]:
```
#best estimator
best_dt = grid_search.best_estimator_
best_dt
```

Out[37]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=2)
```

In [38]:
```
y_pred = best_dt.predict(x_test) #test data
```

In [39]:
```
accuracy_score(y_pred, y_test)
```

Out[39]:
```
0.9529411764705882
```

check if the model is still overfits or not

In [40]:

```
y_pred_train = best_dt.predict(x_train)
```

In [41]:

```
accuracy_score(y_train,y_pred_train)
```

Out[41]:

```
0.898989898989899
```

By tuning the hyperparameters we have got the better accuracy_score and overfitting problem has also been solved.