```
Objective:
    1) To analysis which factor is most affects for heart diseases
    2) To analyse which age group has more chances to get heart attack
```

In [3]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import precision_score,recall_score,f1_score
from sklearn.metrics import roc_curve,auc
```

In [4]:

```python
df= pd.read_csv("C:\\Users\\mahes\\Downloads\\heart.csv")
```

In [5]:

```python
df
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	

1025 rows × 14 columns

In [6]:

```python
#data first 5 row
df.head()
```

Out[6]:

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

In [7]:
```python
#data last 5 rows
df.tail()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	

In [8]:
```python
df.describe()
```

Out[8]:

	age	sex	cp	trestbps	chol	fbs	rest
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000

In [9]:
```python
#Let, we get some basic information about the dataset.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

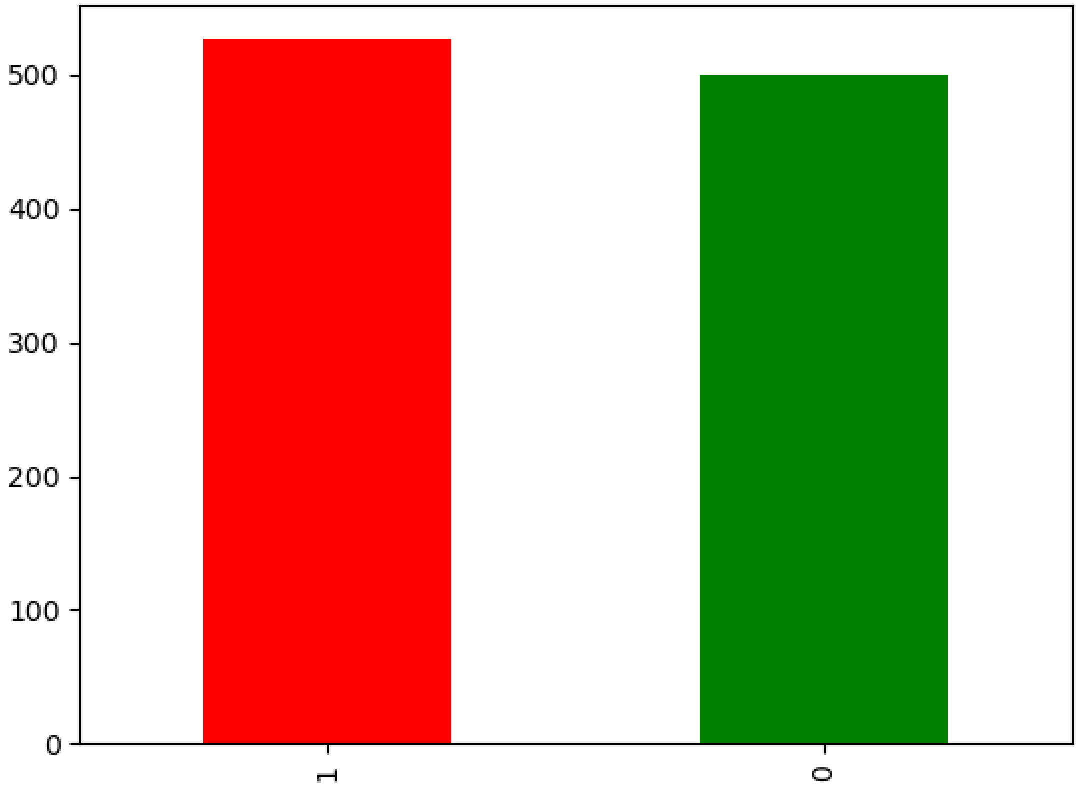
In [10]:
```
#no of row and no of column
df.shape
```

Out[10]:
```
(1025, 14)
```

In [11]:
```
df['target'].value_counts()
```

Out[11]:
```
1    526
0    499
Name: target, dtype: int64
```

In [12]:
```
df['target'].value_counts().plot(kind='bar', color=['red','green']);
```



In [13]:
```
# let us look at whether the dataset has null values or not.
df.isna().sum()
```

Out[13]:
```
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
```

From this output, our data does not contain null values and duplicates. So,the data is good which will be further analyzed.

In [14]:
```
df_dup =df.duplicated().any()
```

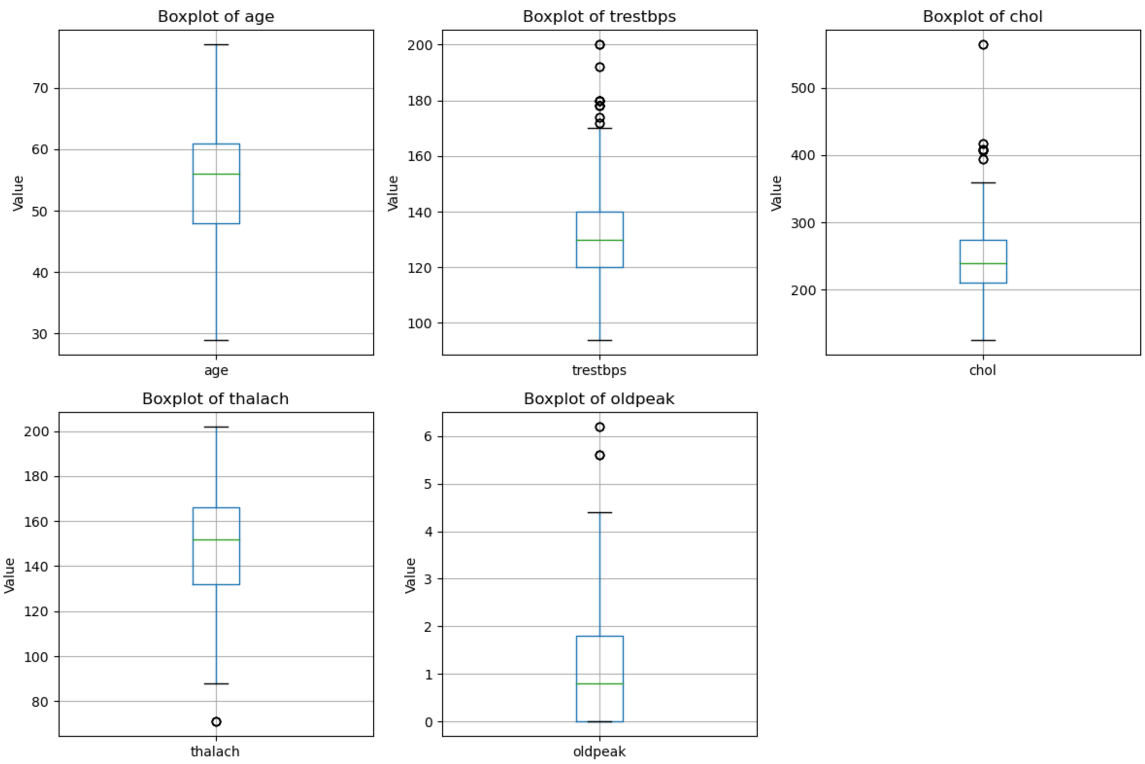
In [15]:
```
df_dup
```

Out[15]:
```
True
```

In [16]:
```
df.sex.value_counts()
```

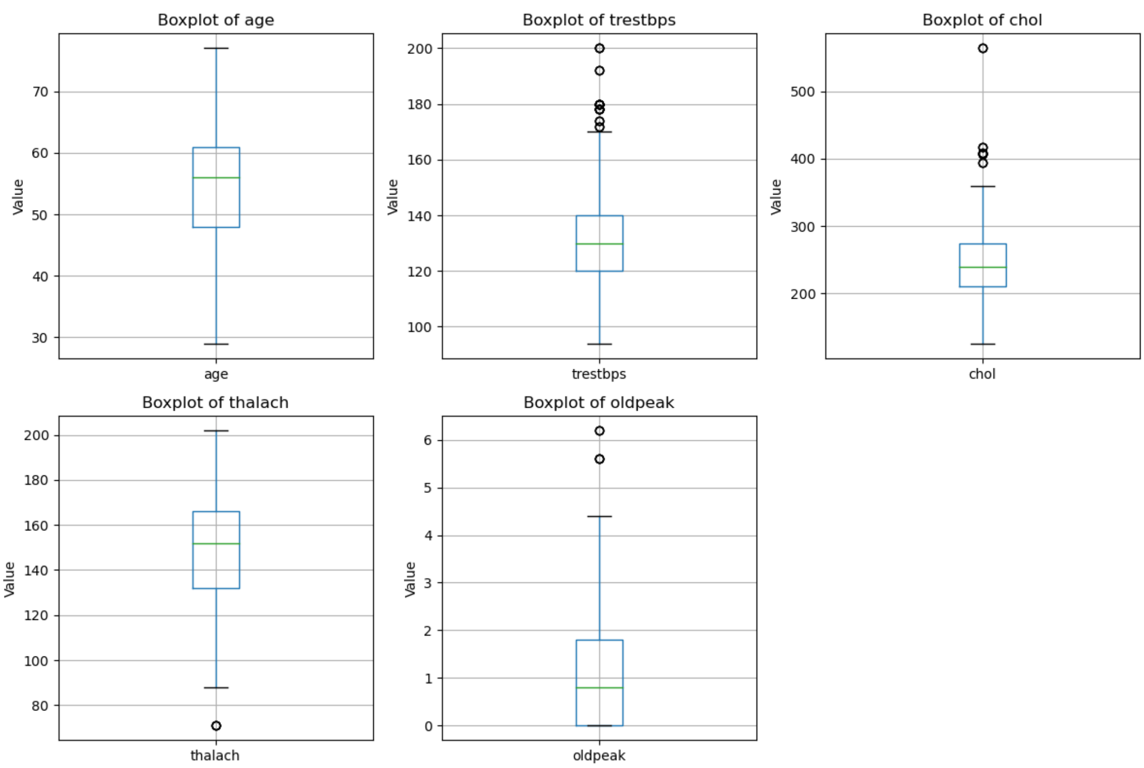
Out[16]:
```
1    713
0    312
Name: sex, dtype: int64
```

In [17]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'data' is your heart disease prediction dataset loaded into

# Select relevant features for visualization
features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

# Create boxplots for each feature to visualize outliers
plt.figure(figsize=(12, 8))
for i, feature in enumerate(features, start=1):
    plt.subplot(2, 3, i)
    df.boxplot(column=feature)
    plt.title(f'Boxplot of {feature}')
    plt.ylabel('Value')

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



In [18]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load your heart disease prediction dataset into a pandas DataFrame
# Assuming 'data' is your DataFrame containing the dataset

# Select relevant features for visualization
features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

# Create boxplots for each feature to visualize outliers
plt.figure(figsize=(12, 8))
for i, feature in enumerate(features, start=1):
    plt.subplot(2, 3, i)
    df.boxplot(column=feature)
    plt.title(f'Boxplot of {feature}')
    plt.ylabel('Value')

plt.tight_layout()
plt.show()

# After visual inspection, determine the range for each feature to filte
# For simplicity, let's assume a range for each feature based on boxplot

# Define the range for each feature based on your observations from boxp
# Example ranges (you need to adjust these based on your data and insigh
ranges = {
    'age': (20, 80),
    'trestbps': (80, 180),
    'chol': (100, 400),
    'thalach': (60, 220),
    'oldpeak': (0, 6)
}

# Apply the range filter to remove outliers from the dataset
for feature, (lower, upper) in ranges.items():
    data = df[(df[feature] >= lower) & (df[feature] <= upper)]
```



In [19]:

```
features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

# Calculate the IQR for each feature
Q1 = data[features].quantile(0.25)
Q3 = data[features].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Create a mask to identify outliers
outlier_mask = ~((data[features] < lower_bound) | (data[features] > uppe

# Filter the dataset to remove outliers
data_no_outliers = data[outlier_mask]

# Print the number of outliers removed
num_outliers_removed = len(data) - len(data_no_outliers)
print(f"Number of outliers removed: {num_outliers_removed}")
plt.show()
```

```
Number of outliers removed: 54
```

In [20]:

```
df
```

Out[20]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	

1025 rows × 14 columns

In [21]:
```
1 data_no_outliers
```

Out[21]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	ta
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	

968 rows × 14 columns

In [22]:
```
1 len(df)
```

Out[22]: 1025

In [23]:
```
1 713/1025
```

Out[23]: 0.6956097560975609

In [24]:
```
1 312/1025
```

Out[24]: 0.304390243902439

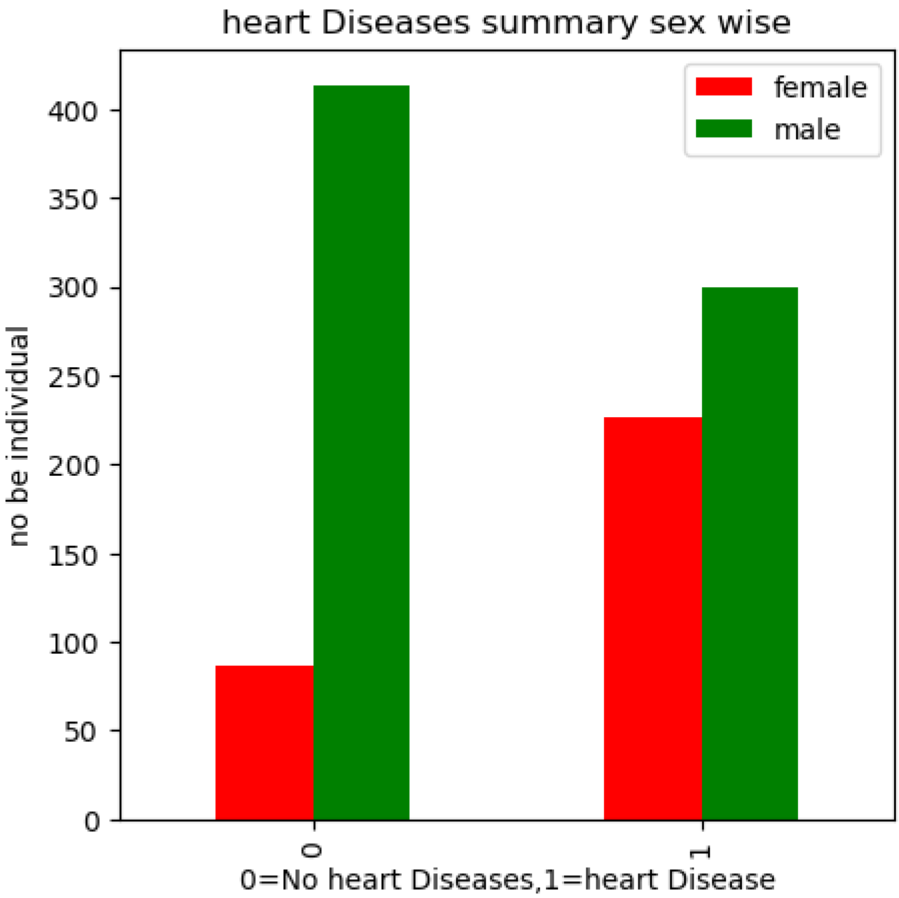
In [25]:
```
1 pd.crosstab(df.target,df.sex)
```

Out[25]:

sex	0	1
target		
0	86	413
1	226	300

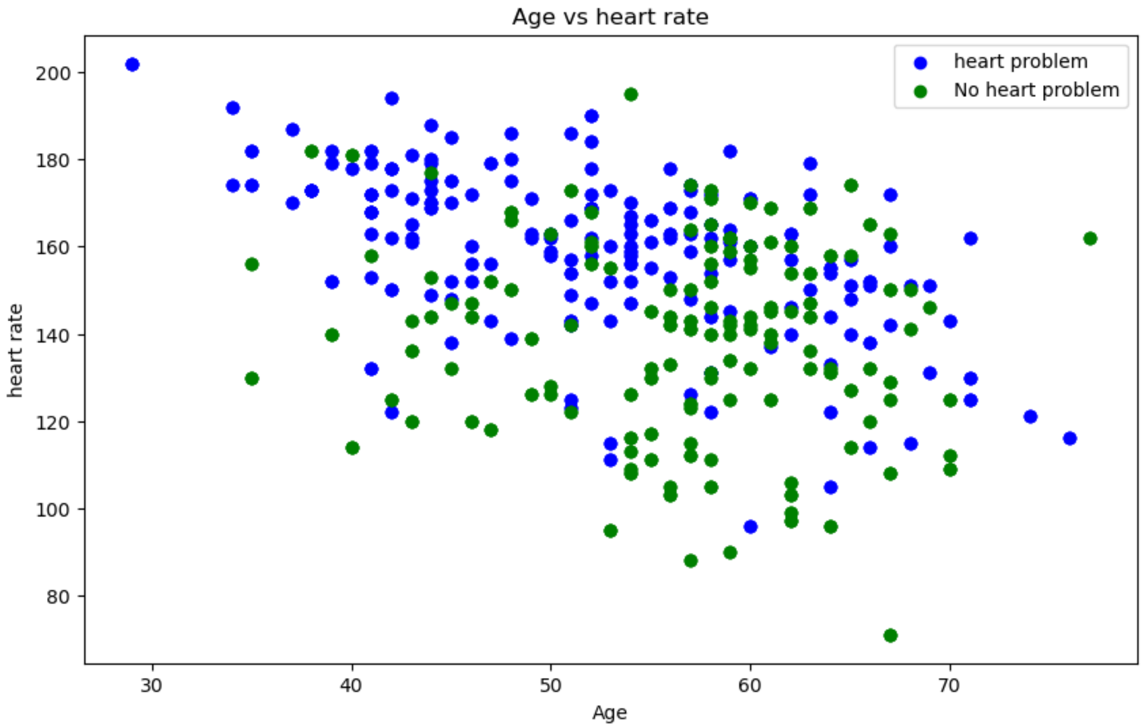
In [26]:
```
pd.crosstab(df.target,df.sex).plot(kind='bar',
                                   figsize=(5,5),
                                   color=['red','green'])
plt.title('heart Diseases summary sex wise')
plt.xlabel('0=No heart Diseases,1=heart Disease')
plt.ylabel('no be individual')
plt.legend(['female','male'])

```

Out[26]: <matplotlib.legend.Legend at 0x17f05817550>

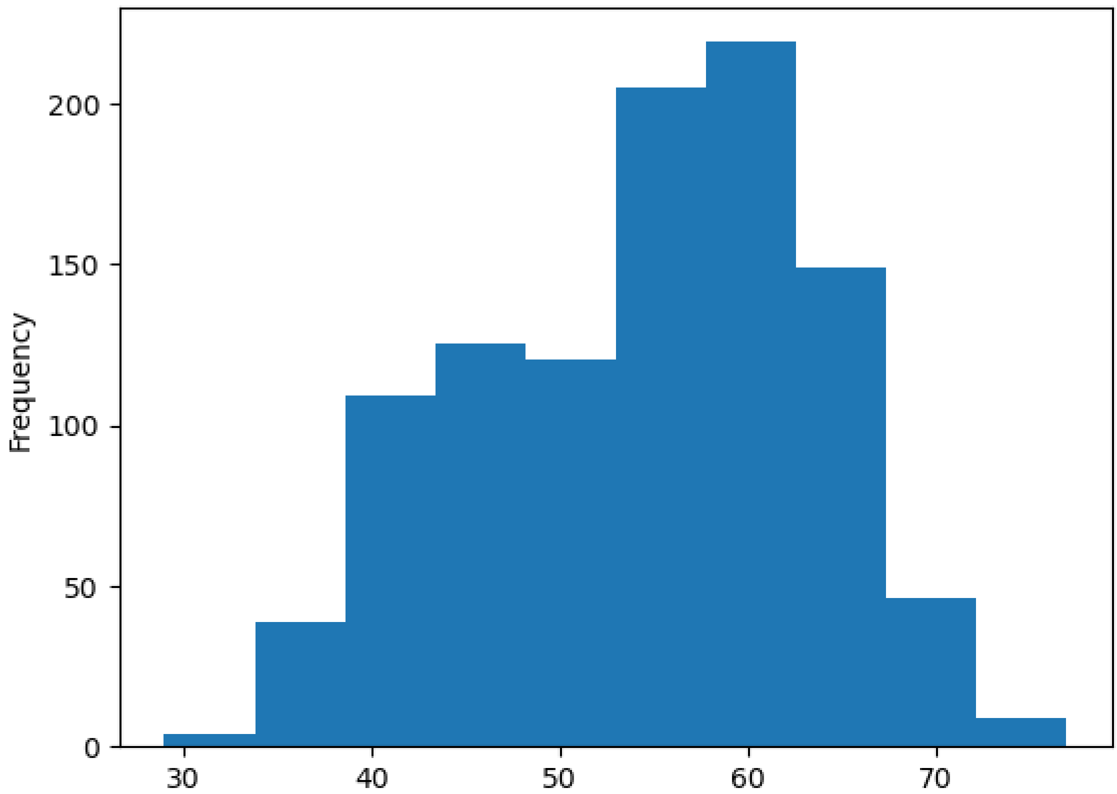


In [27]:
```
plt.figure(figsize=(10,6))
plt.scatter(df.age[df.target==1],df.thalach[df.target==1],c='blue')
plt.scatter(df.age[df.target==0],df.thalach[df.target==0],c='green')
plt.title('Age vs heart rate')
plt.xlabel('Age')
plt.ylabel('heart rate')
plt.legend(['heart problem','No heart problem'])
```

Out[27]: <matplotlib.legend.Legend at 0x17f05356f70>



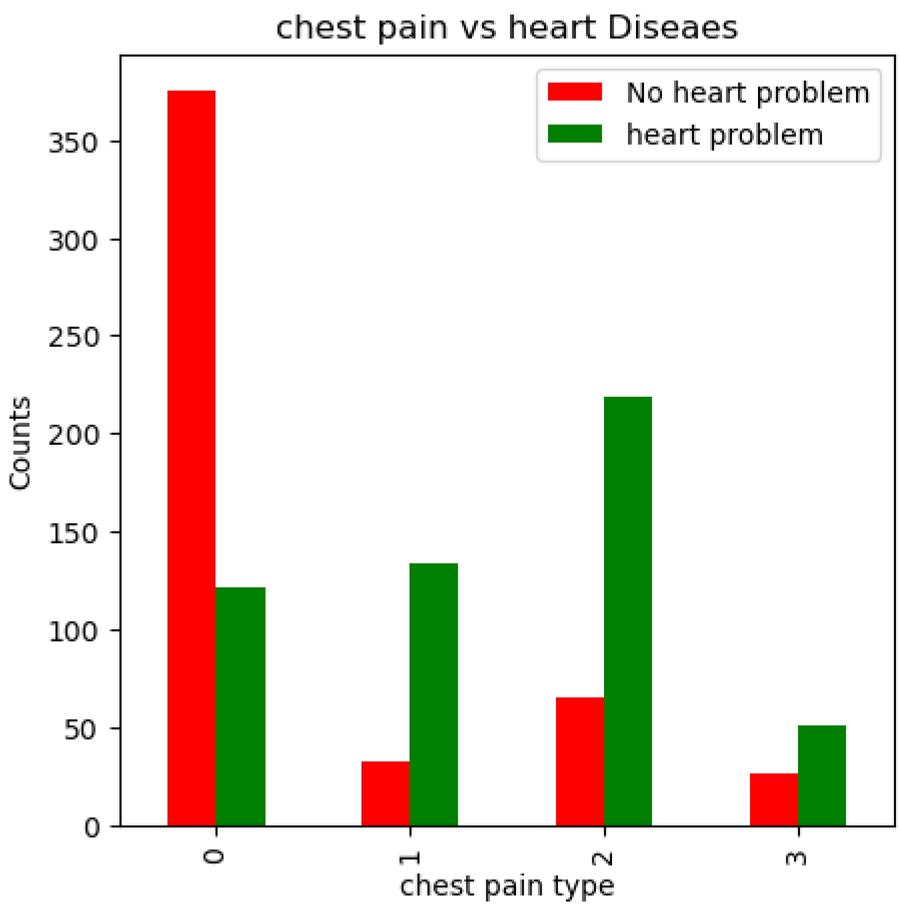
In [28]:
```
df.age.plot.hist();
```



In [29]:

```
pd.crosstab(df.cp,df.target).plot(kind='bar',
                                   figsize=(5,5),
                                   color=['red','green'])
plt.title('chest pain vs heart Diseaes')
plt.xlabel('chest pain type')
plt.ylabel('Counts')
plt.legend(['No heart problem','heart problem'])
```

Out[29]: <matplotlib.legend.Legend at 0x17f05437970>

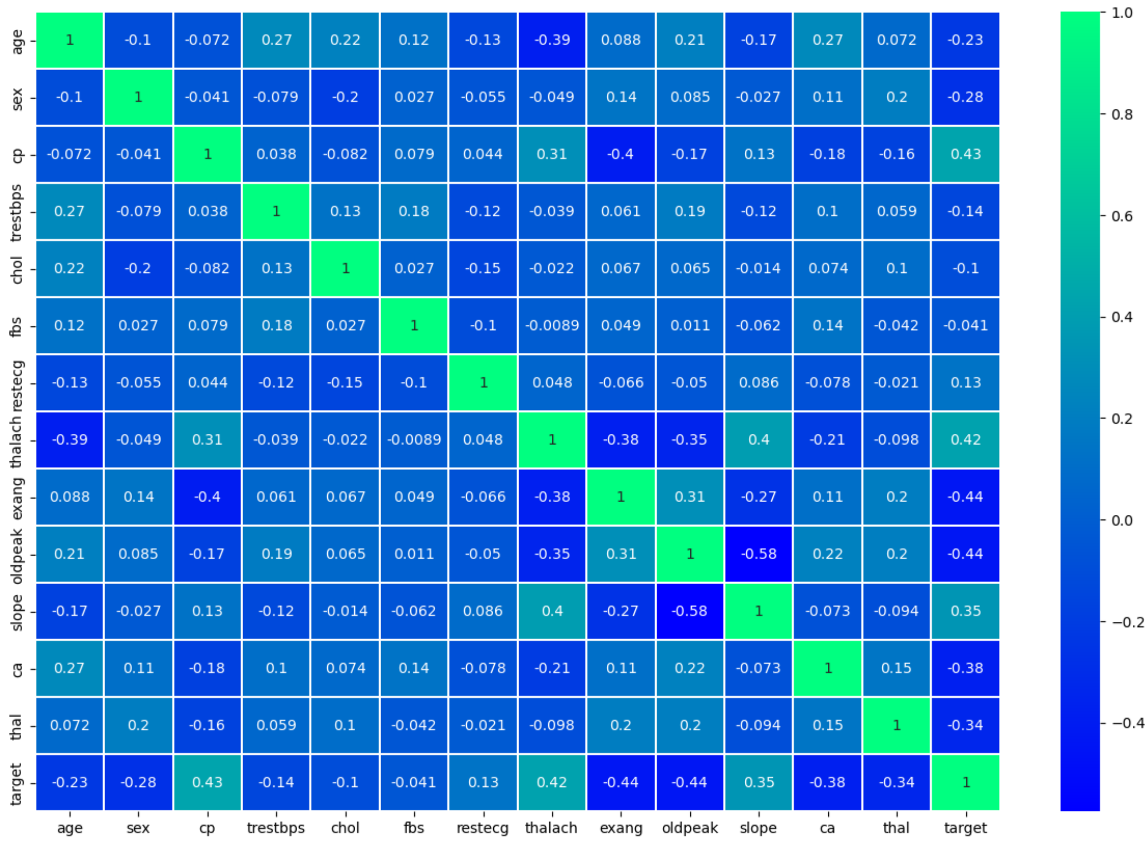


In [30]:
```
df.corr()
```

Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243	-0.132696	-0.390227
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200	-0.055117	-0.049365
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294	0.043581	0.306839
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767	-0.123794	-0.039264
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917	-0.147410	-0.021772
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000	-0.104051	-0.008866
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051	1.000000	0.048411
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866	0.048411	1.000000
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261	-0.065606	-0.380281
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859	-0.050114	-0.349796
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902	0.086086	0.395308
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156	-0.078072	-0.207888
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177	-0.020504	-0.098068
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164	0.134468	0.422895

In [31]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),linewidth=.01,annot=True,cmap="winter")
plt.show()
plt.savefig('correlationfigure')
```



<Figure size 640x480 with 0 Axes>

we can understand that Chest pain(cp) and target have a positive correlation. It means that whose has a large risk of chest pain results in a greater chance to have heart disease. In addition to chest pain, thalach, slope,and resting have a positive correlation with the target. exang and the target have a negative correlation which means when we exercise, the heart requires more blood, but narrowed arteries slow down the blood flow. In addition to ca, old peak, thal have a negative correlation with the target.

Splitting the dataset into the Training Set and Test Set

In [32]:

```
x=df.drop('target', axis=1)
y=df['target']
```

In [33]:

```
x
```

Out[33]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3

1025 rows × 13 columns

In [34]:

```
y
```

Out[34]:

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

In [35]:
```
df.corr()
```

Out[35]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
age	1.000000	-0.103240	-0.071966	0.271121	0.219823	0.121243	-0.132696	-0.390227
sex	-0.103240	1.000000	-0.041119	-0.078974	-0.198258	0.027200	-0.055117	-0.049365
cp	-0.071966	-0.041119	1.000000	0.038177	-0.081641	0.079294	0.043581	0.306839
trestbps	0.271121	-0.078974	0.038177	1.000000	0.127977	0.181767	-0.123794	-0.039264
chol	0.219823	-0.198258	-0.081641	0.127977	1.000000	0.026917	-0.147410	-0.021772
fbs	0.121243	0.027200	0.079294	0.181767	0.026917	1.000000	-0.104051	-0.008866
restecg	-0.132696	-0.055117	0.043581	-0.123794	-0.147410	-0.104051	1.000000	0.048411
thalach	-0.390227	-0.049365	0.306839	-0.039264	-0.021772	-0.008866	0.048411	1.000000
exang	0.088163	0.139157	-0.401513	0.061197	0.067382	0.049261	-0.065606	-0.380281
oldpeak	0.208137	0.084687	-0.174733	0.187434	0.064880	0.010859	-0.050114	-0.349796
slope	-0.169105	-0.026666	0.131633	-0.120445	-0.014248	-0.061902	0.086086	0.395308
ca	0.271551	0.111729	-0.176206	0.104554	0.074259	0.137156	-0.078072	-0.207888
thal	0.072297	0.198424	-0.163341	0.059276	0.100244	-0.042177	-0.020504	-0.098068
target	-0.229324	-0.279501	0.434854	-0.138772	-0.099966	-0.041164	0.134468	0.422895

In [36]:
```
dataset=pd.get_dummies(df,columns=["sex","cp","fbs","restecg","exang","
```

In [37]:
```
from sklearn .model_selection import train_test_split
from sklearn .preprocessing import StandardScaler
standardscalar = StandardScaler()
Columns_to_Scale =["age","trestbps","chol","thalach","oldpeak"]

```

In [38]:
```
dataset.head()
```

Out[38]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0
0	52	125	212	168	1.0	0	0	1	1	0	...	1	0
1	53	140	203	155	3.1	0	0	1	1	0	...	0	1
2	70	145	174	125	2.6	0	0	1	1	0	...	0	1
3	61	148	203	161	0.0	0	0	1	1	0	...	1	0
4	62	138	294	106	1.9	0	1	0	1	0	...	0	0

5 rows × 31 columns

In [39]:
```
x=df.drop('target', axis=1)
y=df['target']
```

In [40]:
```
x
```

Out[40]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3

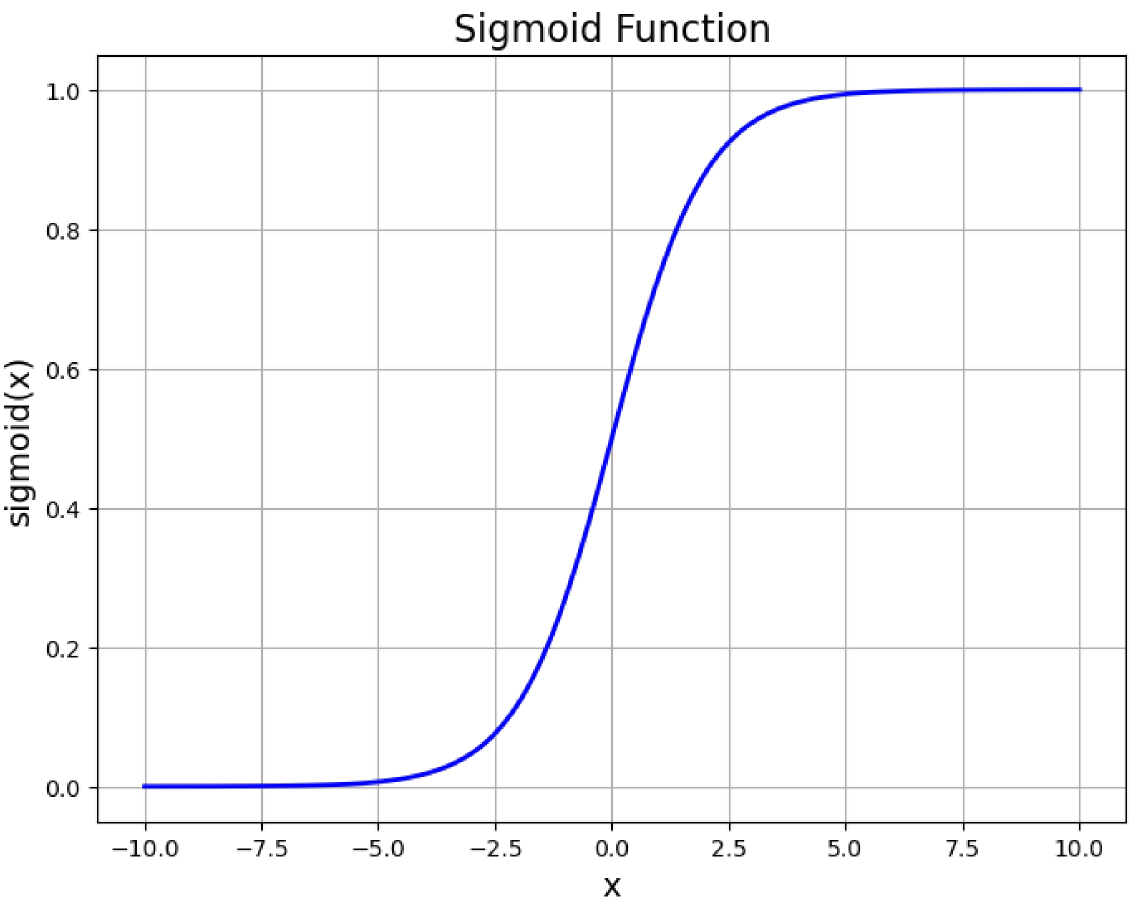
1025 rows × 13 columns

In [41]:
```
np.random.seed(7)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

logistic Regression

In [42]:

```
import numpy as np
import matplotlib.pyplot as plt

# Define the sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Generate values for x
x = np.linspace(-10, 10, 100)

# Compute the corresponding y values using the sigmoid function
y = sigmoid(x)

# Plot the sigmoid function
plt.figure(figsize=(8, 6))
plt.plot(x, y, color='blue', linewidth=2)

# Add labels and title
plt.title('Sigmoid Function', fontsize=16)
plt.xlabel('x', fontsize=14)
plt.ylabel('sigmoid(x)', fontsize=14)
plt.grid(True)

# Show the plot
plt.show()
```



In [43]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confu
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Initialize and train the logistic regression model
log_reg_model = LogisticRegression()
log_reg_model.fit(x_train, y_train)

# Predict on the test set
y_pred = log_reg_model.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Classification report
print(classification_report(y_test, y_pred))

# Confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

```

```
Accuracy: 0.8682926829268293
              precision    recall  f1-score   support

           0       0.84      0.87      0.85        89
           1       0.89      0.87      0.88       116

    accuracy                           0.87       205
   macro avg       0.87      0.87      0.87       205
weighted avg       0.87      0.87      0.87       205

Confusion Matrix:
[[ 77  12]
 [ 15 101]]
```

Decision tree

In [44]:
```
from sklearn import tree
```

In [45]:
```
learner = tree.DecisionTreeClassifier(random_state=0)
learner.fit(x_train,y_train)
```

Out[45]:
```
DecisionTreeClassifier(random_state=0)
```

In [46]:
```
y_train_pred = learner.predict(x_train)
```

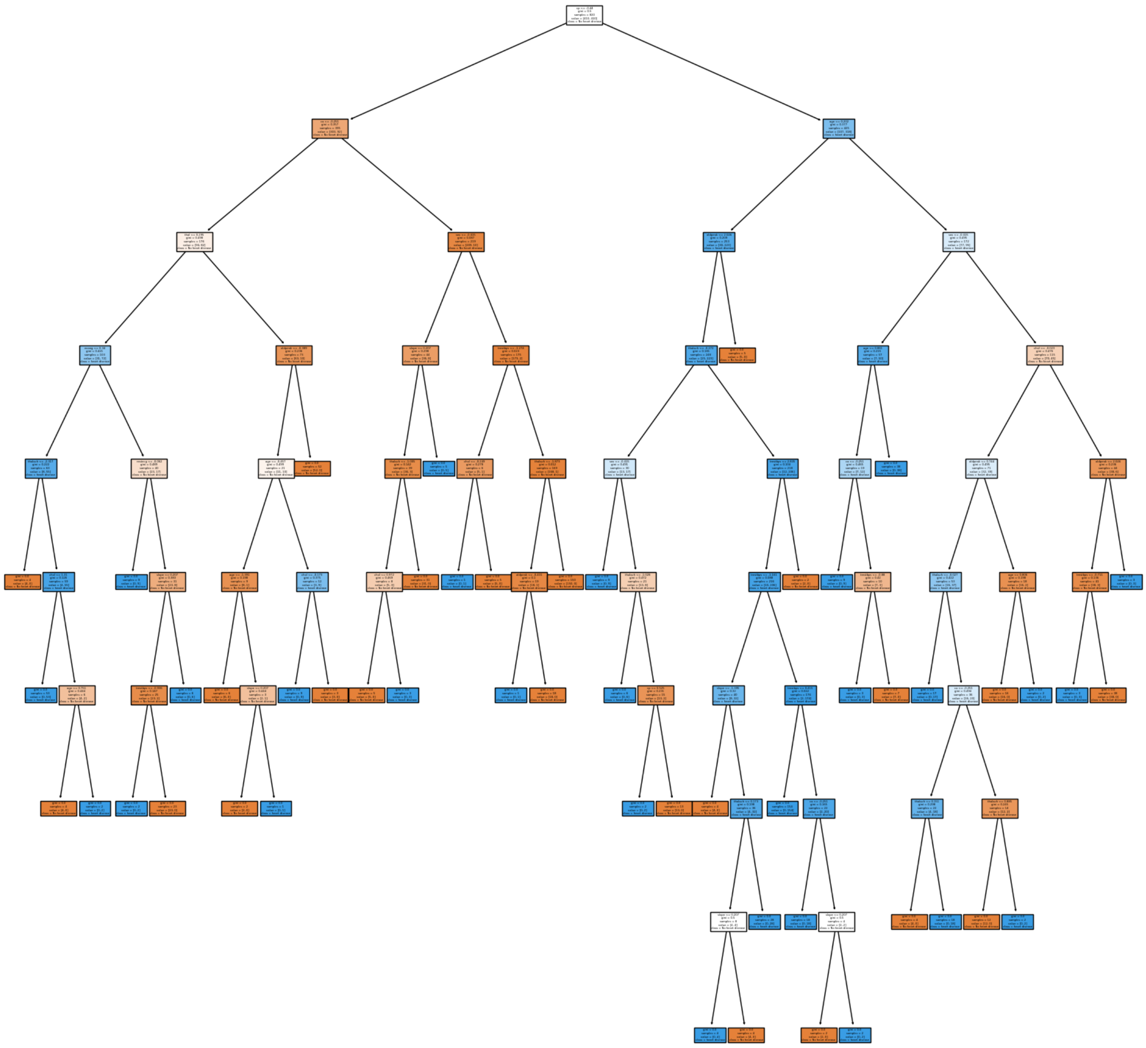
In [47]:
```
#prediction for New data
y_test_pred = learner.predict(x_test)
```

In [48]:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [49]:

```
#visualization the decision tree
plt.figure(figsize=(20,20))
features=df.columns
classes = ['No heart disease','heart disease']
tree.plot_tree(learner,feature_names=features,class_names=classes,fille
plt.show()
```



In [50]:

```
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [51]:

```
print(f'training accurancy{accuracy_score(y_train_pred,y_train)}')
```

```
training accurancy1.0
```

In [52]:

```
print(f'Test accurancy{accuracy_score(y_test_pred,y_test)}')
```

```
Test accurancy1.0
```

In [53]:

```
#post proning - Cost complexity proning approach
path= learner.cost_complexity_pruning_path(x_train,y_train)
#path variable gives two values:ccp_alphas and impurities
ccp_alphas,impurities=path.ccp_alphas,path.impurities
print("ccp alpha provides list of values:",ccp_alphas)
print("Impurities in Decision Tree:",impurities)
```

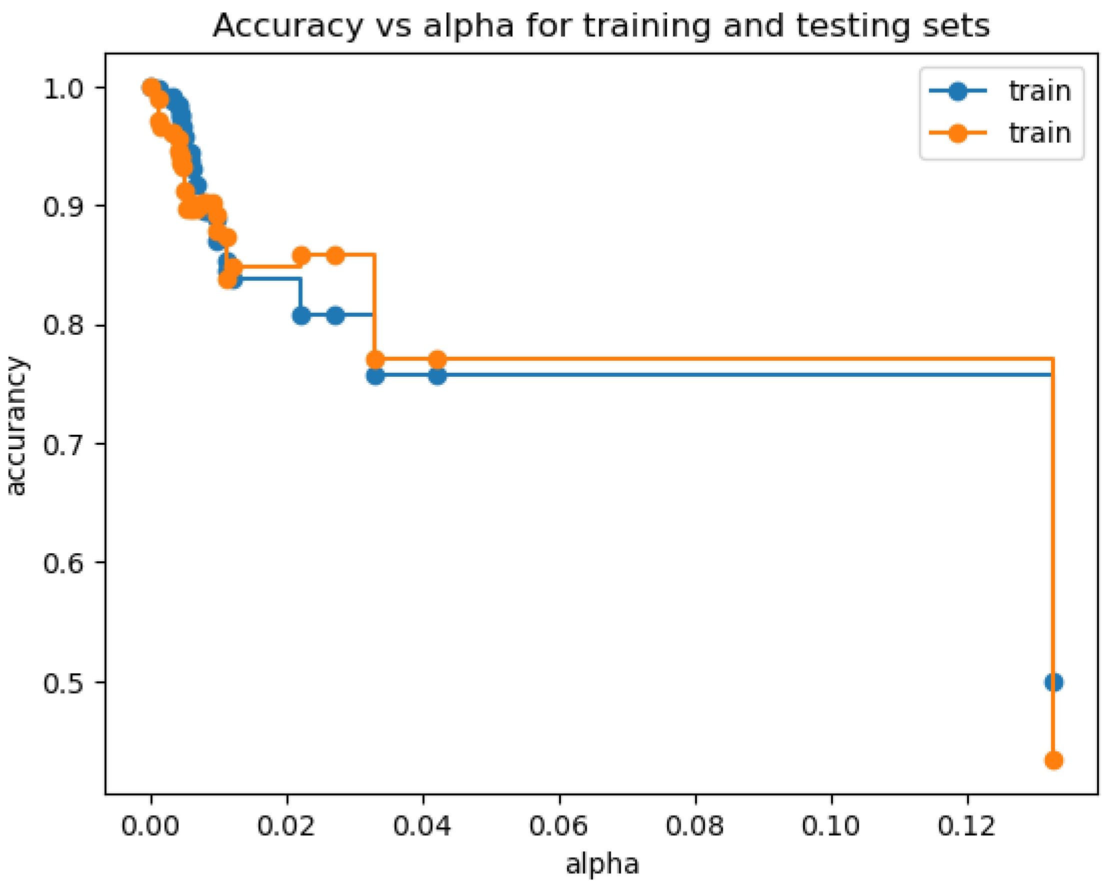
```
ccp alpha provides list of values: [0.         0.00108401 0.00120557 0.0016
0754 0.00325203 0.00337711
 0.00418118 0.00422764 0.00433604 0.00433604 0.0044878  0.00472083
 0.00499216 0.00530391 0.00584263 0.00585001 0.00631929 0.00686992
 0.00768742 0.00793979 0.00827327 0.0092042  0.00966549 0.00967683
 0.01129238 0.01129761 0.0119775  0.02194491 0.02704736 0.03291565
 0.04203248 0.13260164]
Impurities in Decision Tree: [0.         0.00216802 0.00699032 0.01181294
0.01506497 0.02181919
 0.02600038 0.03022802 0.03890011 0.04323615 0.04772395 0.06188644
 0.07686291 0.08747073 0.09331337 0.10501338 0.11765196 0.1313918
 0.15445405 0.16239384 0.17066712 0.17987132 0.1895368  0.20889046
 0.22018284 0.23148045 0.24345796 0.26540287 0.29245023 0.32536588
 0.36739836 0.5       ]
```

In [54]:

```
All_learners=[]
for ccp_alpha in ccp_alphas:
    learner=tree.DecisionTreeClassifier(random_state=0,ccp_alpha=ccp_al
    learner.fit(x_train,y_train)
    All_learners.append(learner)
print("Last node in Decision tree is {} and ccp_alpha for last node is 
```

```
Last node in Decision tree is <sklearn.tree._tree.Tree object at 0x0000017F
05C559D0> and ccp_alpha for last node is 0.13260163812360393
```

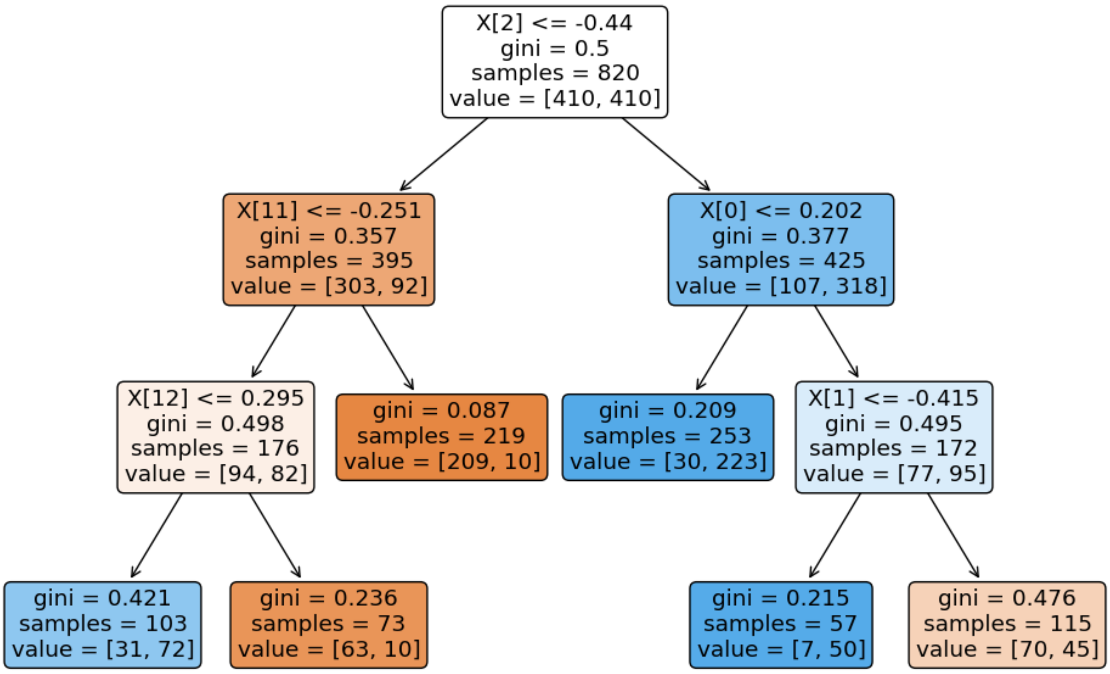
In [55]:
```
#visulizing the accurancy for train and test set
train_scores = [learner.score(x_train,y_train) for learner in All_learne
test_scores = [learner.score(x_test,y_test) for learner in All_learners]
fig, ax= plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accurancy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas,train_scores,marker='o',label='train',drawstyle='ste
ax.plot(ccp_alphas,test_scores,marker='o',label='train',drawstyle='steps
ax.legend()
plt.show()
```



The plot displays the accuracy of the model on the training set and the testing set as a function of the complexity parameter (ccp_alpha). As the complexity parameter increases (meaning more aggressive pruning), the accuracy of the model on the training set tends to decrease. This is because overly aggressive pruning may lead to underfitting the training data. On the other hand, the accuracy on the testing set may initially increase with increasing ccp_alpha as the model generalizes better, but after a certain point, it may start to decrease due to excessive pruning, leading to overfitting. The goal is to find the optimal value of ccp_alpha that maximizes the accuracy on the testing set without sacrificing too much accuracy on the training set, thus achieving good generalization performance.

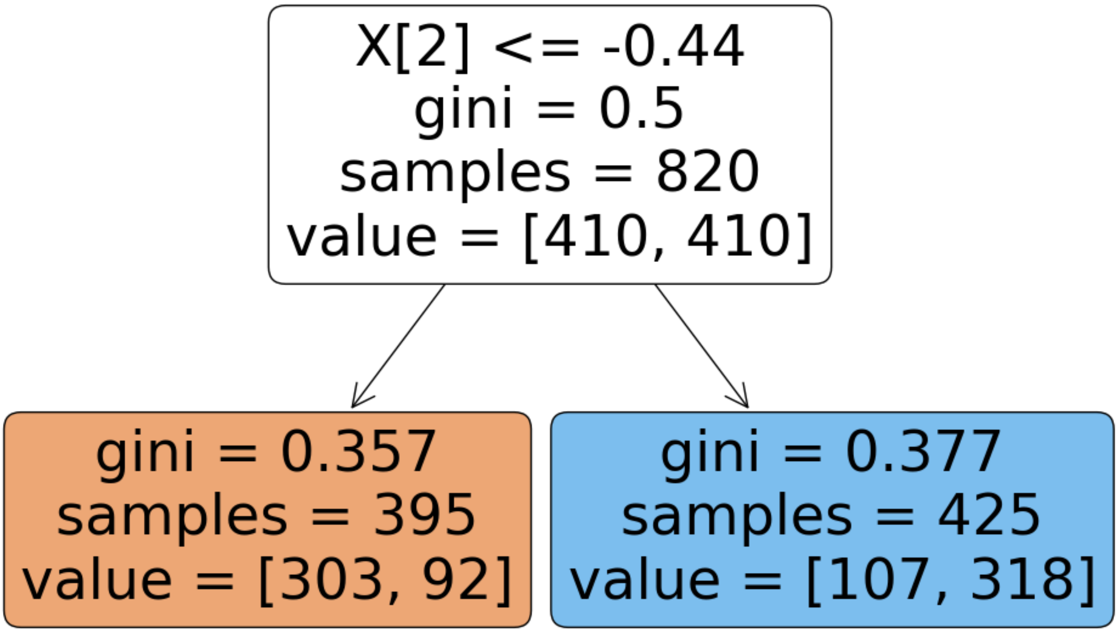
In [56]:

```
#choose the point with law bias(Low traning error) and Low variance (Lo
learner=tree.DecisionTreeClassifier (random_state=0,ccp_alpha=0.02)
learner.fit(x_train,y_train)
plt.figure(figsize=(12,8))
tree.plot_tree(learner,rounded=True,filled=True)
plt.show()

print(accuracy_score(y_test,learner.predict(x_test)))
```



0.848780487804878

In [57]:
```
learner=tree.DecisionTreeClassifier (random_state=0,ccp_alpha=0.12)
learner.fit(x_train,y_train)
plt.figure(figsize=(12,8))
tree.plot_tree(learner,rounded=True,filled=True)
plt.show()

print(accuracy_score(y_test,learner.predict(x_test)))
```




```
0.7707317073170732
```

In [58]:
```
learner=tree.DecisionTreeClassifier (random_state=0)
learner.fit(x_train,y_train)

```

Out[58]:
```
DecisionTreeClassifier(random_state=0)
```

In [59]:
```
from sklearn.model_selection import GridSearchCV
```

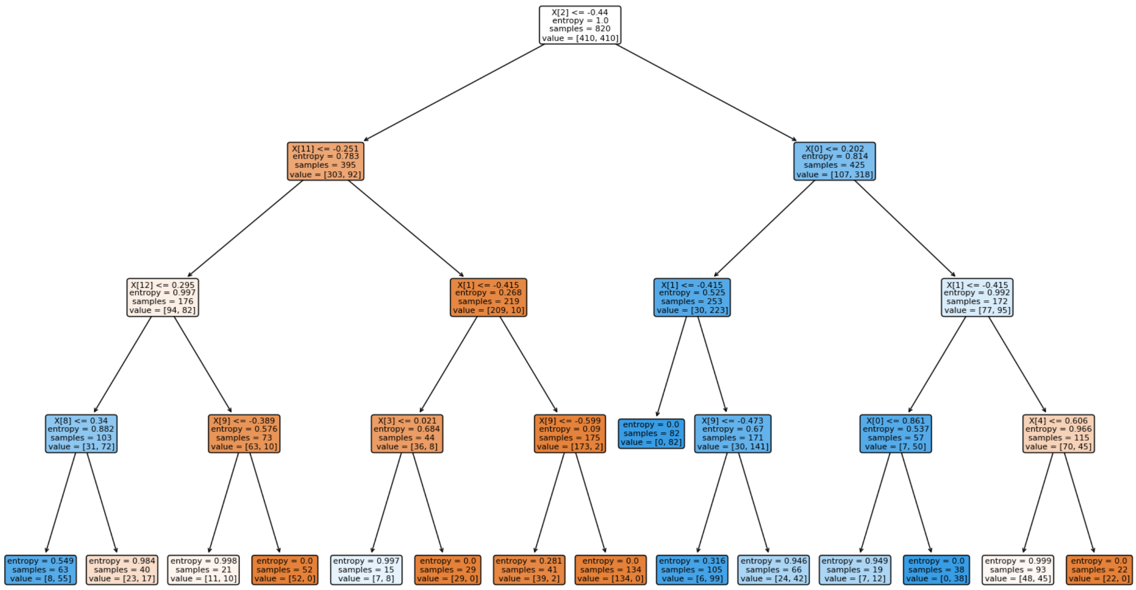
In [60]:
```
#PRE-pruning
grid_param ={"criterion":["gini","entropy"],
            "splitter":["best","random"],
            "max_depth":range(2,5,1),
             "min_samples_leaf":range(1,15,1),
            "min_samples_split":range(2,20,1)}

grid_search=GridSearchCV(estimator=learner,param_grid=grid_param,cv=5,n_
grid_search.fit(x_train,y_train)
```

Out[60]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0), n_jobs
=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 5),
                         'min_samples_leaf': range(1, 15),
                         'min_samples_split': range(2, 20),
                         'splitter': ['best', 'random']})
```

In [61]:
```
#Extract best parameters
print(grid_search.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_s
plit': 2, 'splitter': 'best'}
```

In [62]:
```
learner=tree.DecisionTreeClassifier(criterion= 'entropy', max_depth= 4,
learner.fit(x_train,y_train)
plt.figure(figsize=(20,12))
tree.plot_tree(learner, rounded=True, filled=True)
plt.show()
```



In [65]:

```
x = df.drop('target', axis=1) # Features
y = df['target'] # Target variable

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Initialize and train the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(x_train, y_train)

# Predict on the test set
y_pred = dt_classifier.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Classification report
print(classification_report(y_test, y_pred))

# Confusion matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9853658536585366
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       102
           1       1.00      0.97      0.99       103

    accuracy                           0.99       205
   macro avg       0.99      0.99      0.99       205
weighted avg       0.99      0.99      0.99       205

Confusion Matrix:
[[102   0]
 [  3 100]]
```

Random forest

In [70]:

```
from sklearn.naive_bayes import GaussianNB
x = df.drop('target', axis=1)  # Features
y = df['target']  # Labels

# 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

# Initialize and train the Naive Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(x_train, y_train)
y_pred = nb_classifier.predict(x_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.71      0.78       102
           1       0.75      0.89      0.82       103

    accuracy                           0.80       205
   macro avg       0.81      0.80      0.80       205
weighted avg       0.81      0.80      0.80       205
```

In [ ]:

```

```