In [153]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that ge
# You can also write temporary files to /kaggle/temp/, but they won't be saved
```

```
/kaggle/input/heart-failure-prediction/heart.csv
```

In [154]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

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

In [155]:
```

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/heart-failure-prediction/heart.csv
```

In [156]:
```
df = pd.read_csv('/kaggle/input/heart-failure-prediction/heart.csv')
```

In [157]:
```
df.head()
```

Out[157]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
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	

In [158]:
```
df.shape
```

Out[158]:
```
(918, 12)
```

In [159]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [160]:
```
df.isnull().sum()/len(df)*100
```

Out[160]:
```
Age               0.0
Sex               0.0
ChestPainType     0.0
RestingBP         0.0
Cholesterol       0.0
FastingBS         0.0
RestingECG        0.0
MaxHR             0.0
ExerciseAngina    0.0
Oldpeak           0.0
ST_Slope          0.0
HeartDisease      0.0
dtype: float64
```

In [161]:
```
df_dup = df.duplicated().any()
print(df_dup)
```

```
False
```

In [162]:
```
df.describe()
```

Out[162]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.553377
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.497414
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.000000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.000000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.000000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.000000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.000000

In [ ]:
```
df['Sex'].value_counts()
```

In [164]:
```
df['Sex'] = df['Sex'].astype('category')
df['Sex'] = df['Sex'].cat.codes
```

In [320]:
```
df['ChestPainType'].value_counts()
```

Out[320]:
```
0    496
2    203
1    173
3     46
Name: ChestPainType, dtype: int64
```

In [166]:
```
df['ChestPainType'] = df['ChestPainType'].astype('category')
df['ChestPainType'] = df['ChestPainType'].cat.codes
```

In [319]:
```
df['RestingECG'].value_counts()
```

Out[319]:
```
1    552
0    188
2    178
Name: RestingECG, dtype: int64
```

In [168]:
```
df['RestingECG'] = df['RestingECG'].astype('category')
df['RestingECG'] = df['RestingECG'].cat.codes
```

In [318]:
```
df['ExerciseAngina'].value_counts()
```

Out[318]:
```
0    547
1    371
Name: ExerciseAngina, dtype: int64
```

In [170]:
```
df['ExerciseAngina'] = df['ExerciseAngina'].astype('category')
df['ExerciseAngina'] = df['ExerciseAngina'].cat.codes
```

In [317]:
```
df['ST_Slope'].value_counts()
```

Out[317]:
```
1    460
2    395
0     63
Name: ST_Slope, dtype: int64
```

In [172]:
```
df['ST_Slope'] = df['ST_Slope'].astype('category')
df['ST_Slope'] = df['ST_Slope'].cat.codes
```

In [173]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    int8   
 2   ChestPainType   918 non-null    int8   
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    int8   
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    int8   
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    int8   
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), int8(5)
memory usage: 54.8 KB
```

In [174]:
```
df.head()
```

Out[174]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	

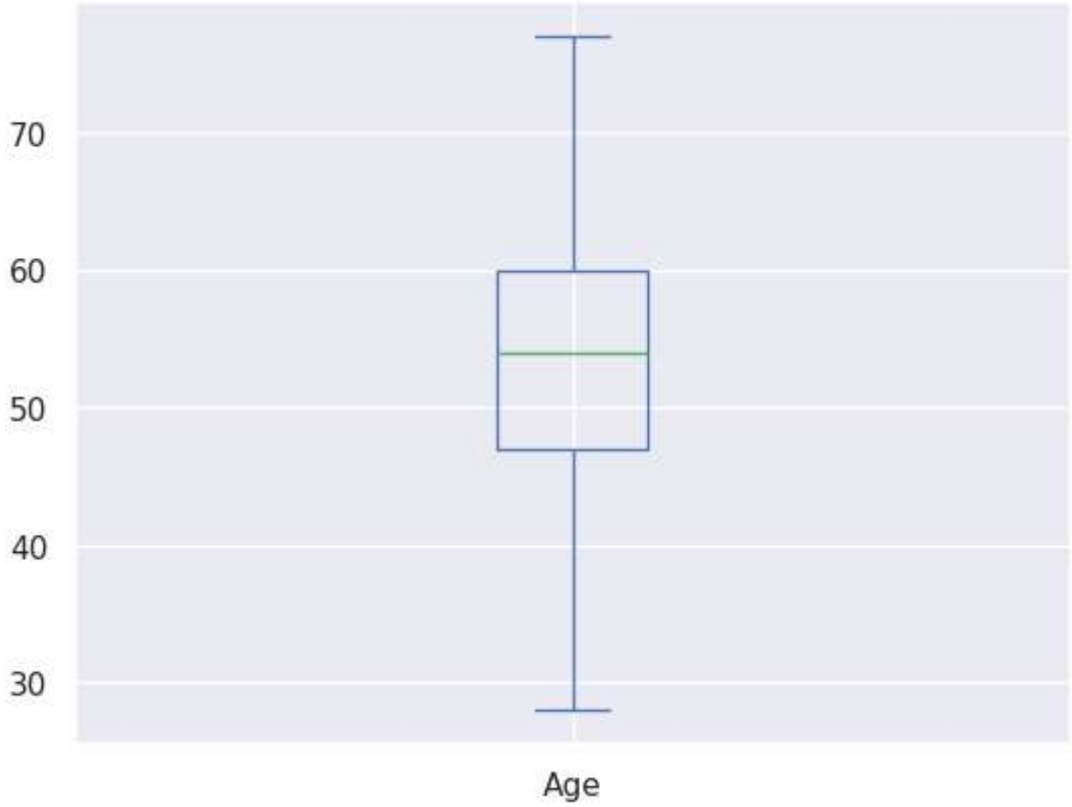
In [175]:
```
df.describe()
```

Out[175]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000
mean	53.510893	0.789760	0.781046	132.396514	198.799564	0.233115	0.989107
std	9.432617	0.407701	0.956519	18.514154	109.384145	0.423046	0.631671
min	28.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.000000	1.000000	0.000000	120.000000	173.250000	0.000000	1.000000
50%	54.000000	1.000000	0.000000	130.000000	223.000000	0.000000	1.000000
75%	60.000000	1.000000	2.000000	140.000000	267.000000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	603.000000	1.000000	2.000000

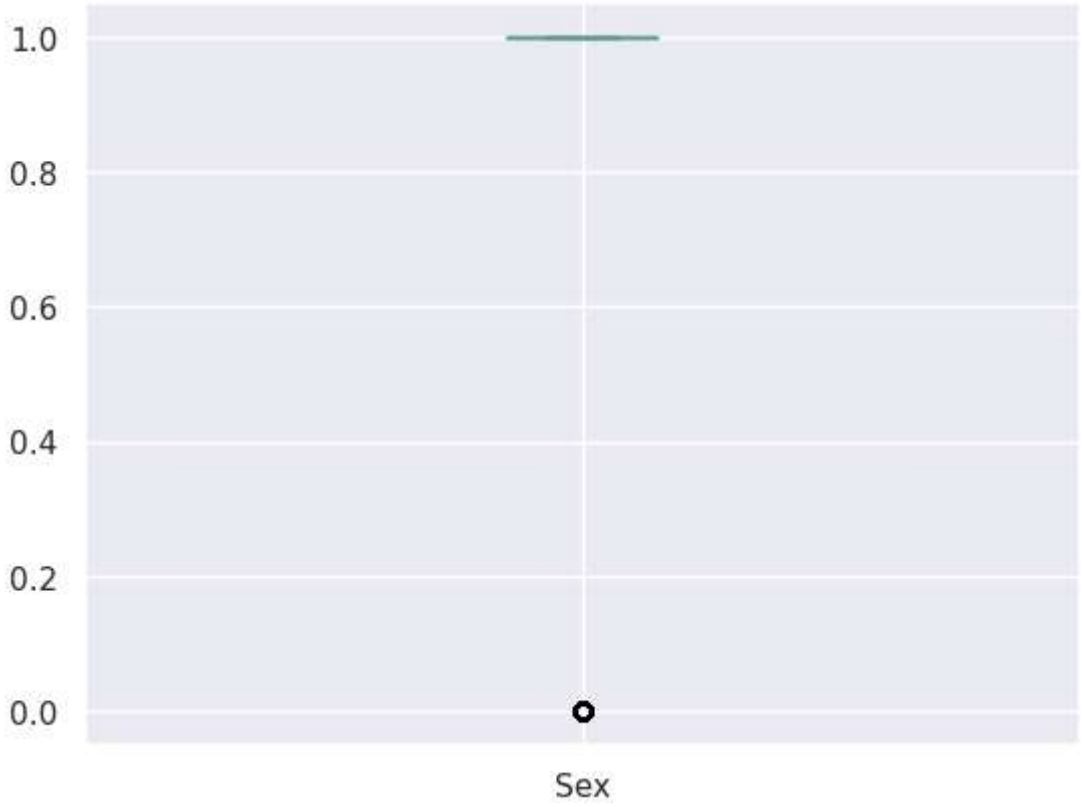
In [178]:
```
df['Age'].plot(kind='box')
```

Out[178]: <Axes: >



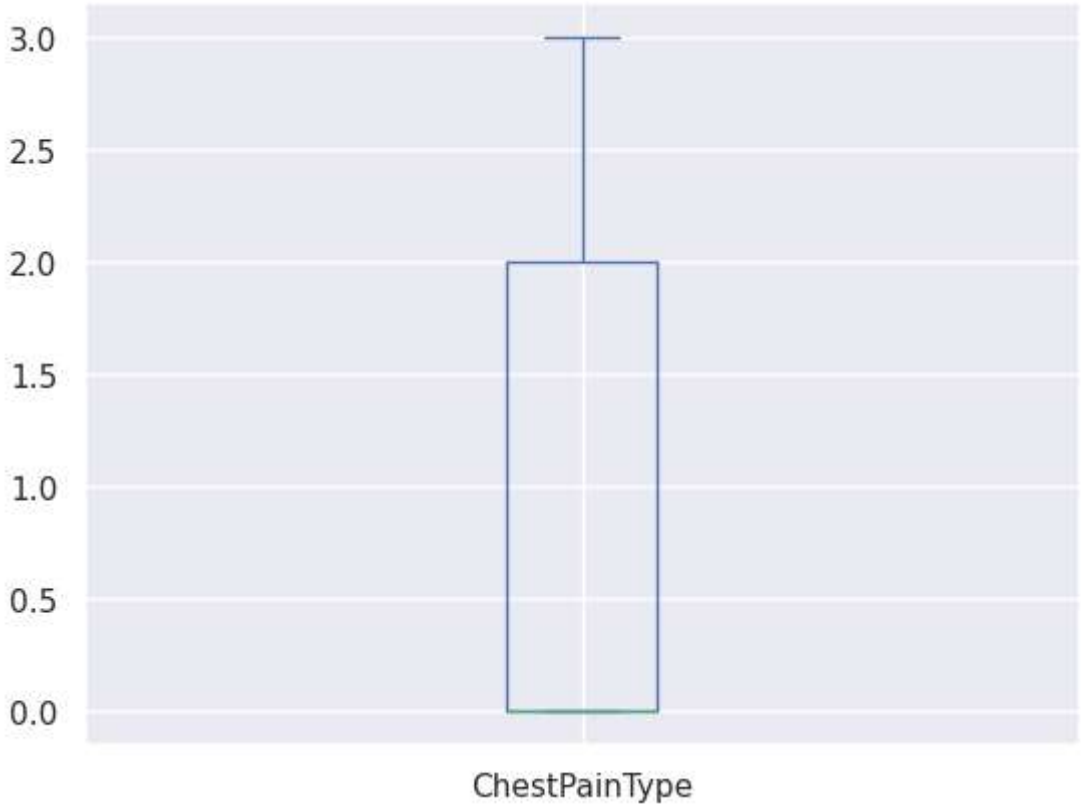
In [179]:
```
df['Sex'].plot(kind='box')
```

Out[179]: <Axes: >



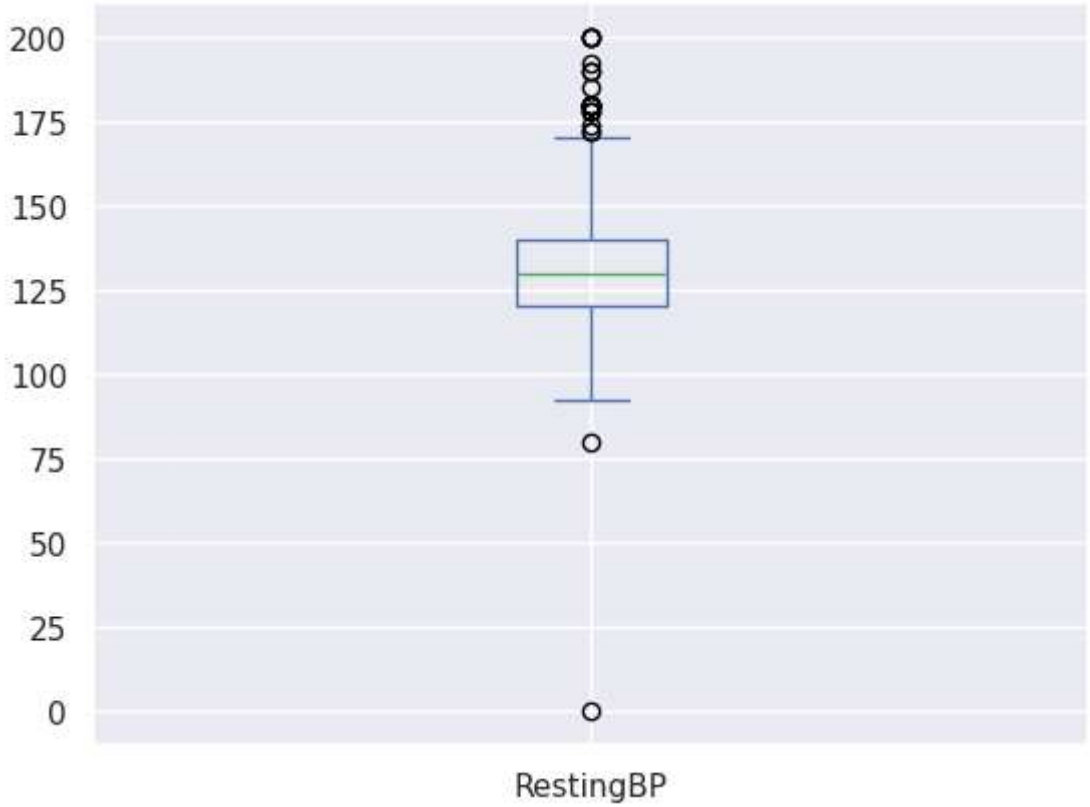
In [180]:
```
df['ChestPainType'].plot(kind='box')
```

Out[180]: <Axes: >



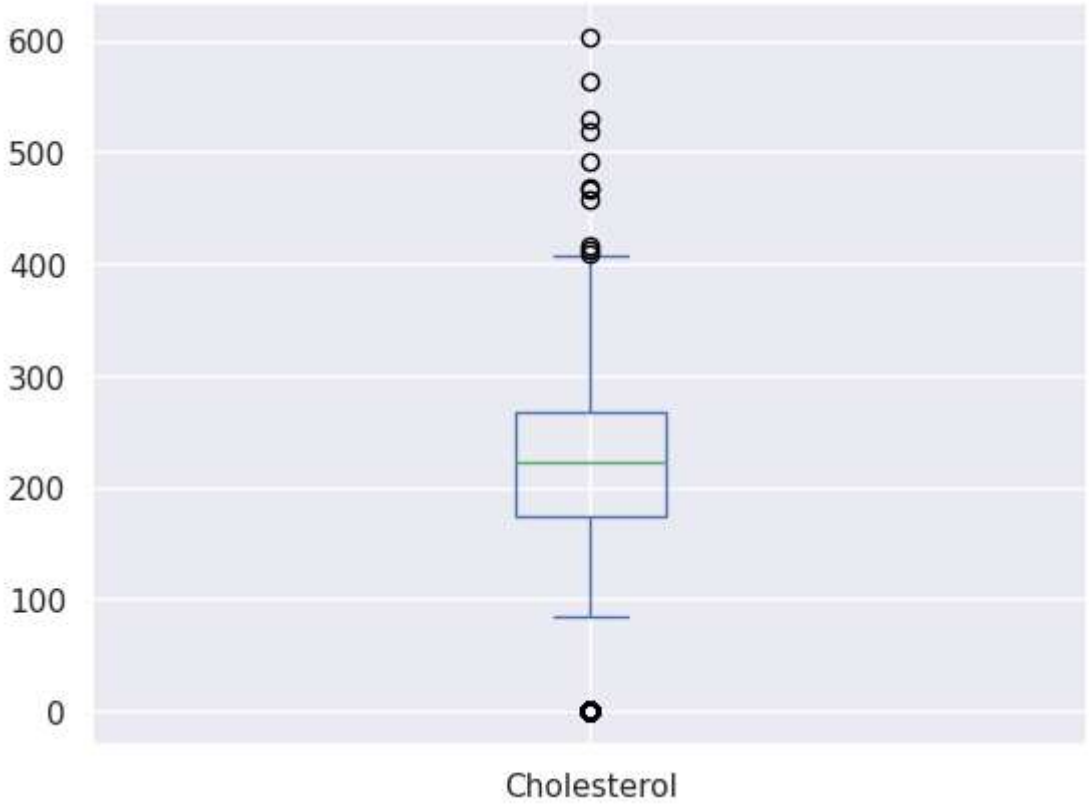
In [181]:
```
df['RestingBP'].plot(kind='box')
```

Out[181]: <Axes: >



In [182]:
```
df['Cholesterol'].plot(kind='box')
```

Out[182]: <Axes: >

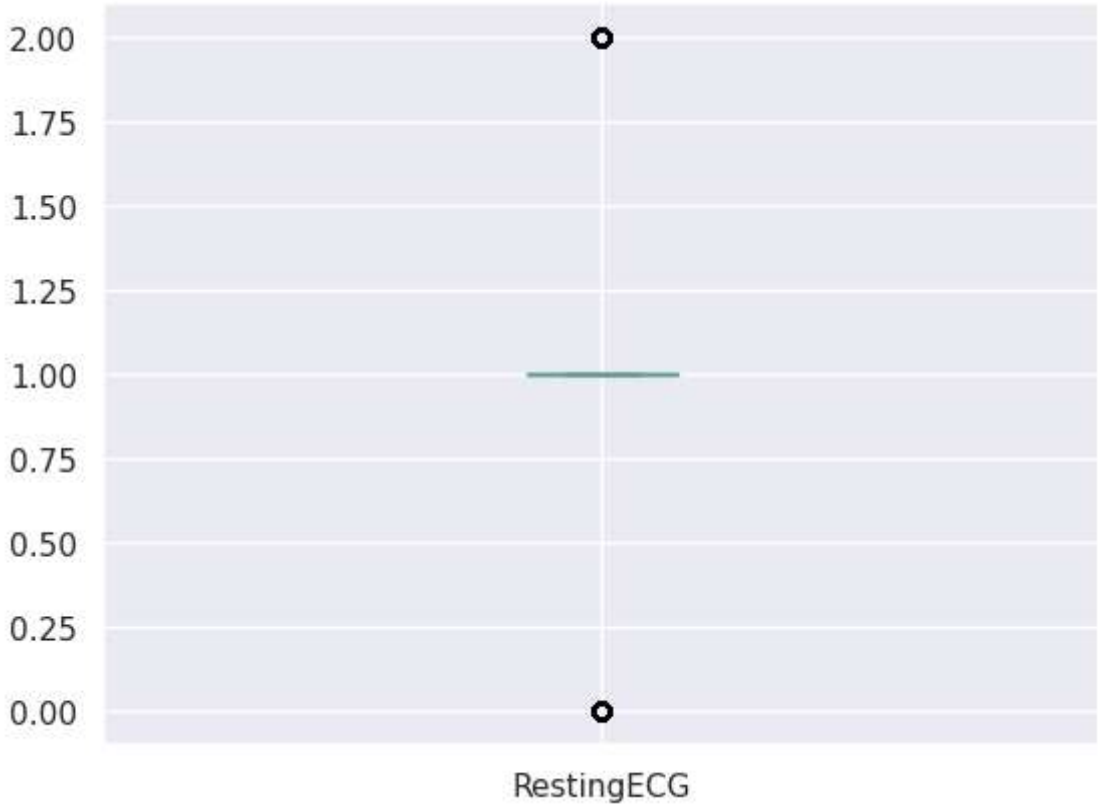


In [183]:
```
df['FastingBS'].plot(kind='box')
```

Out[183]: <Axes: >

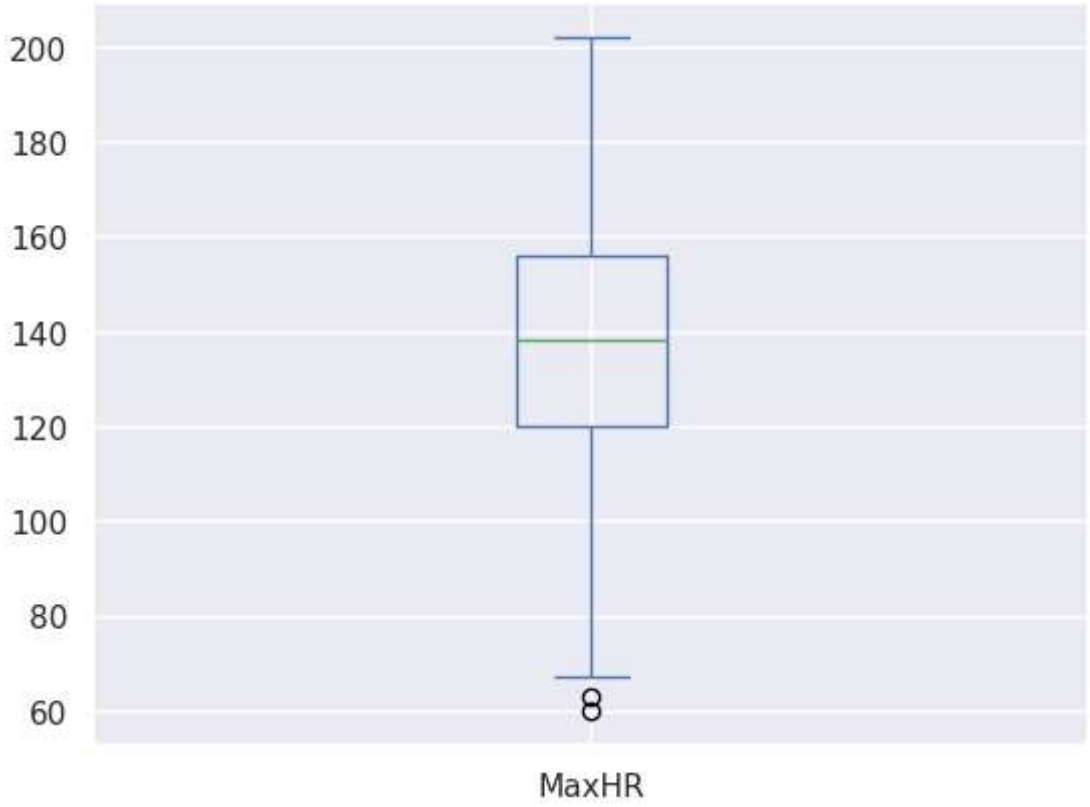
In [184]:
```
df['RestingECG'].plot(kind='box')
```

Out[184]: <Axes: >



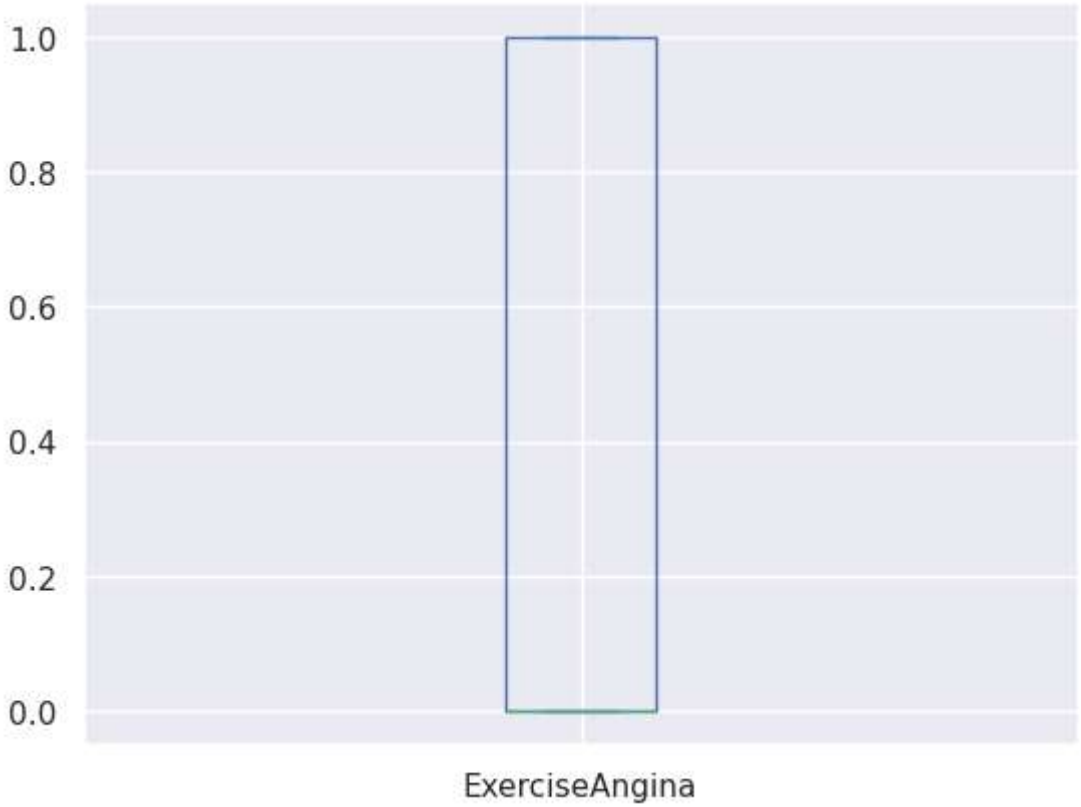
In [185]:
```
df['MaxHR'].plot(kind='box')
```

Out[185]: <Axes: >



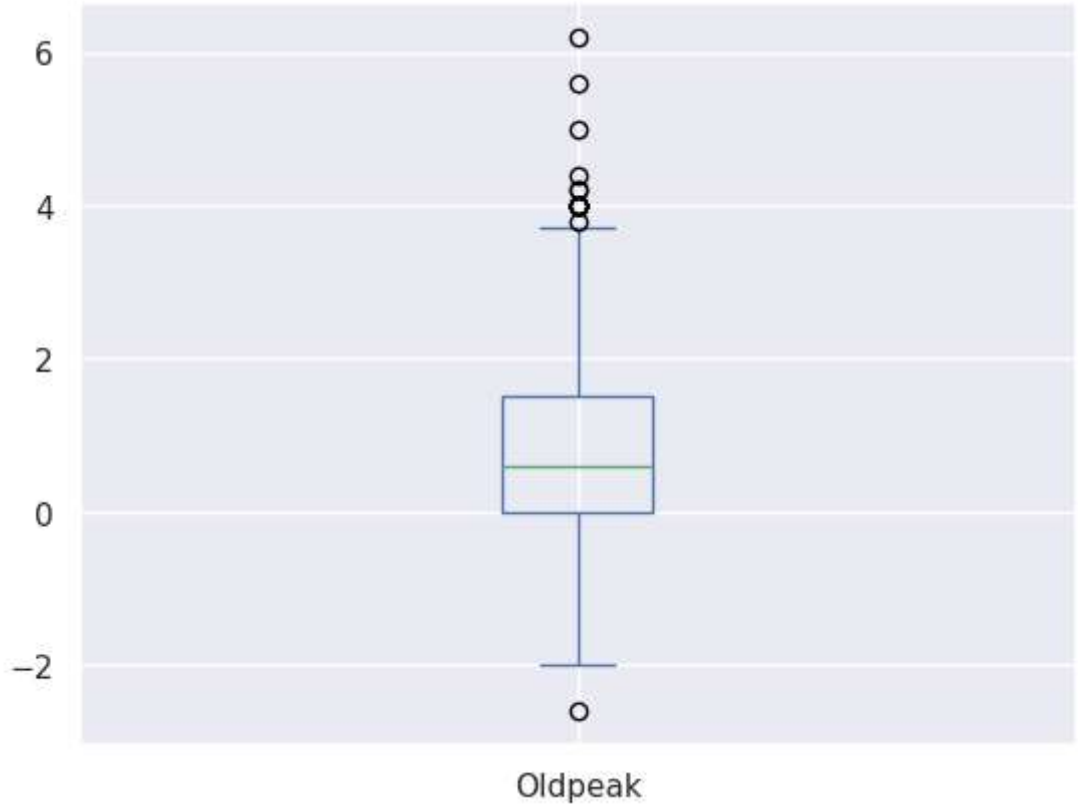
In [186]:
```
df['ExerciseAngina'].plot(kind='box')
```

Out[186]: <Axes: >



In [187]:
```
df['Oldpeak'].plot(kind='box')
```

Out[187]: <Axes: >

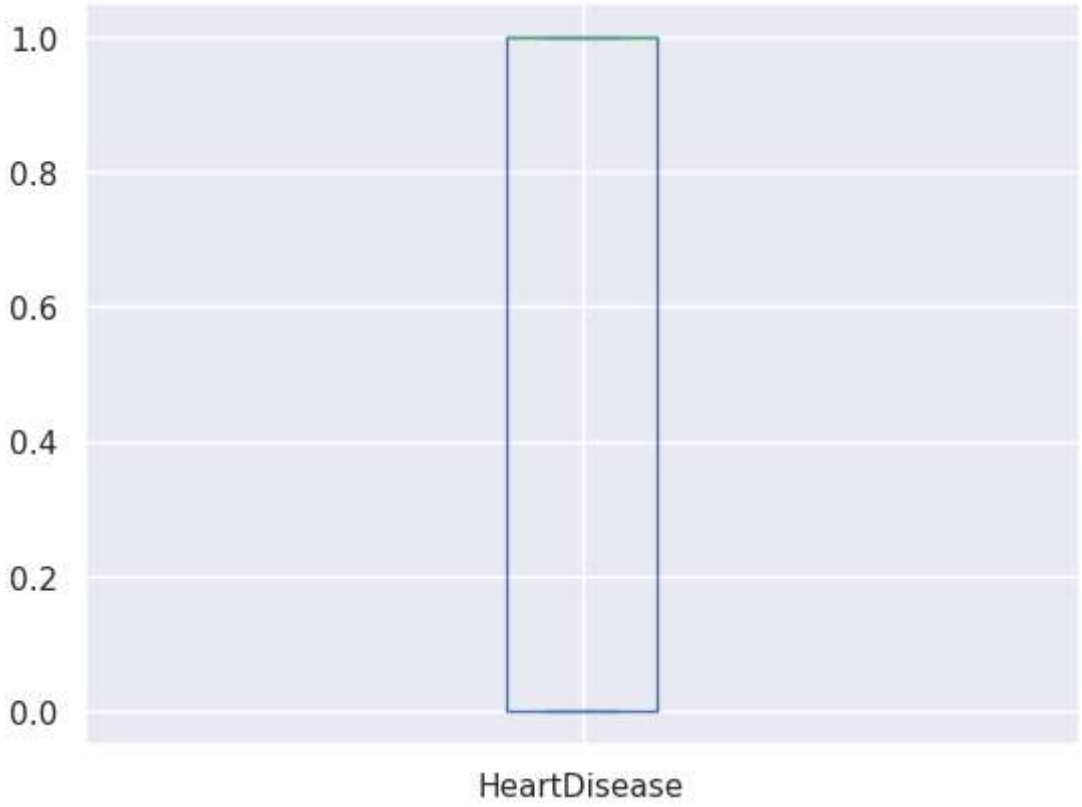


In [188]:
```
df['ST_Slope'].plot(kind='box')
```

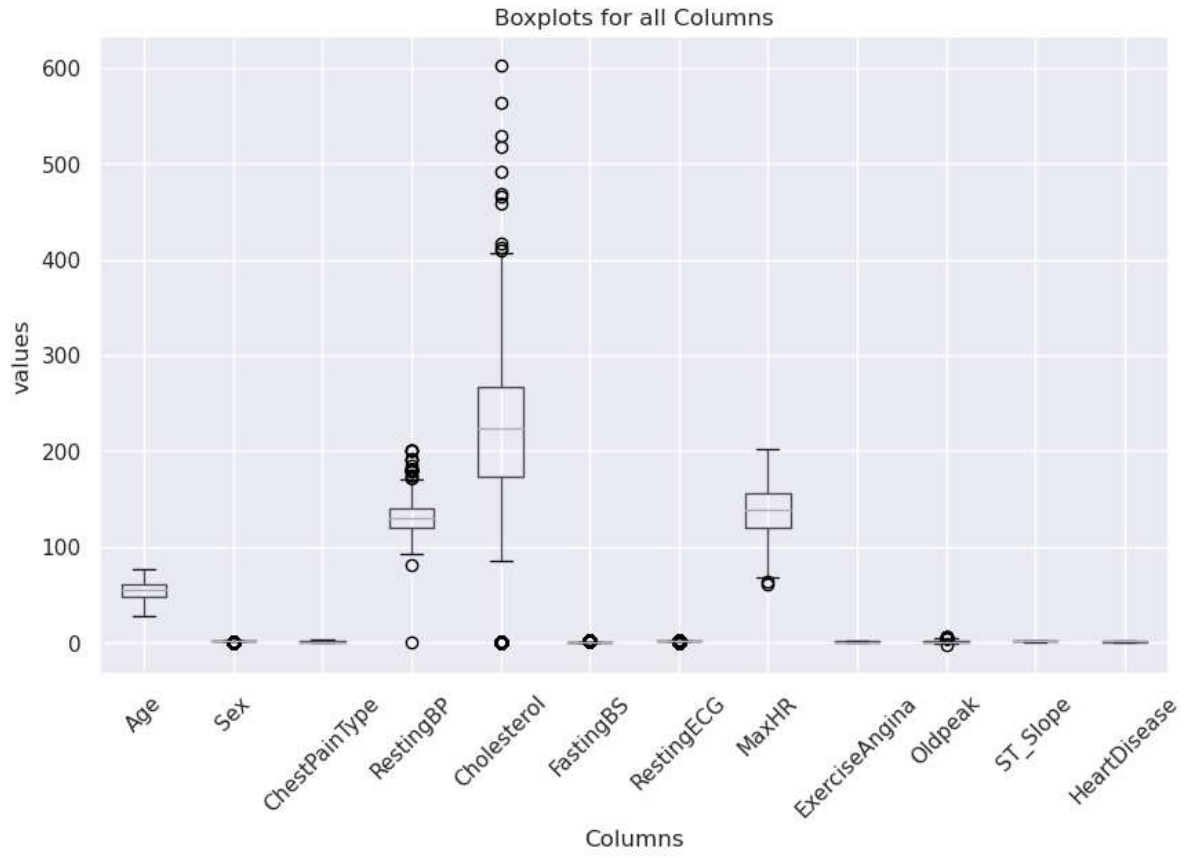
Out[188]: <Axes: >

In [189]:
```
df['HeartDisease'].plot(kind='box')
```

Out[189]: <Axes: >



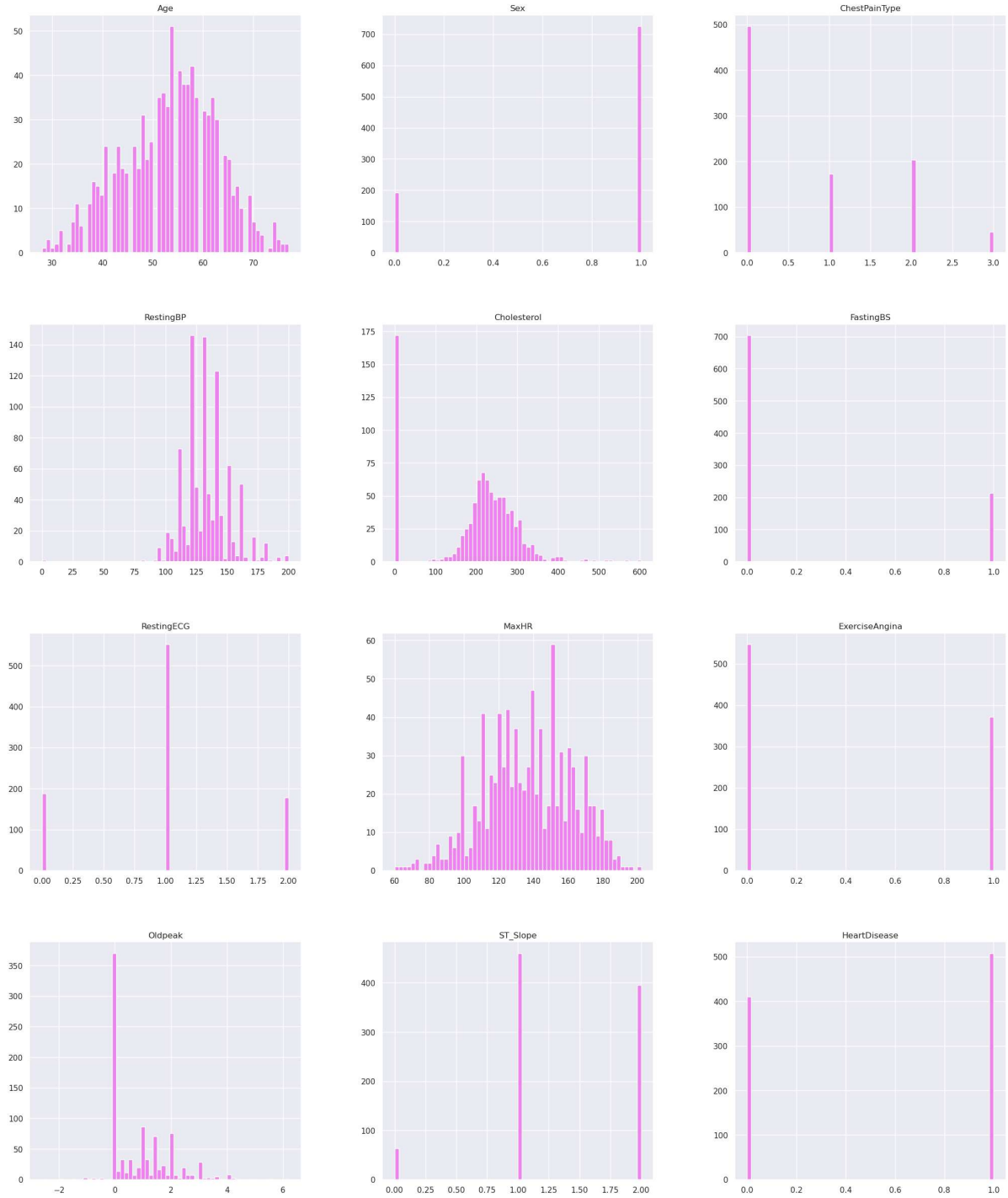
In [190]:
```
num_col = df.select_dtypes(exclude='object').columns.tolist()
plt.figure(figsize = (10,6))
df.boxplot(column = num_col)
plt.title('Boxplots for all Columns')
plt.xticks(rotation = 45)
plt.xlabel('Columns')
plt.ylabel('values')
plt.grid(True)
plt.show()
```



In [191]:
```
df.columns
```

Out[191]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingB
S',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

In [192]:
```
df.hist(bins=60,figsize=(25,30),color='violet')
plt.show()
```



Find the uniqe values

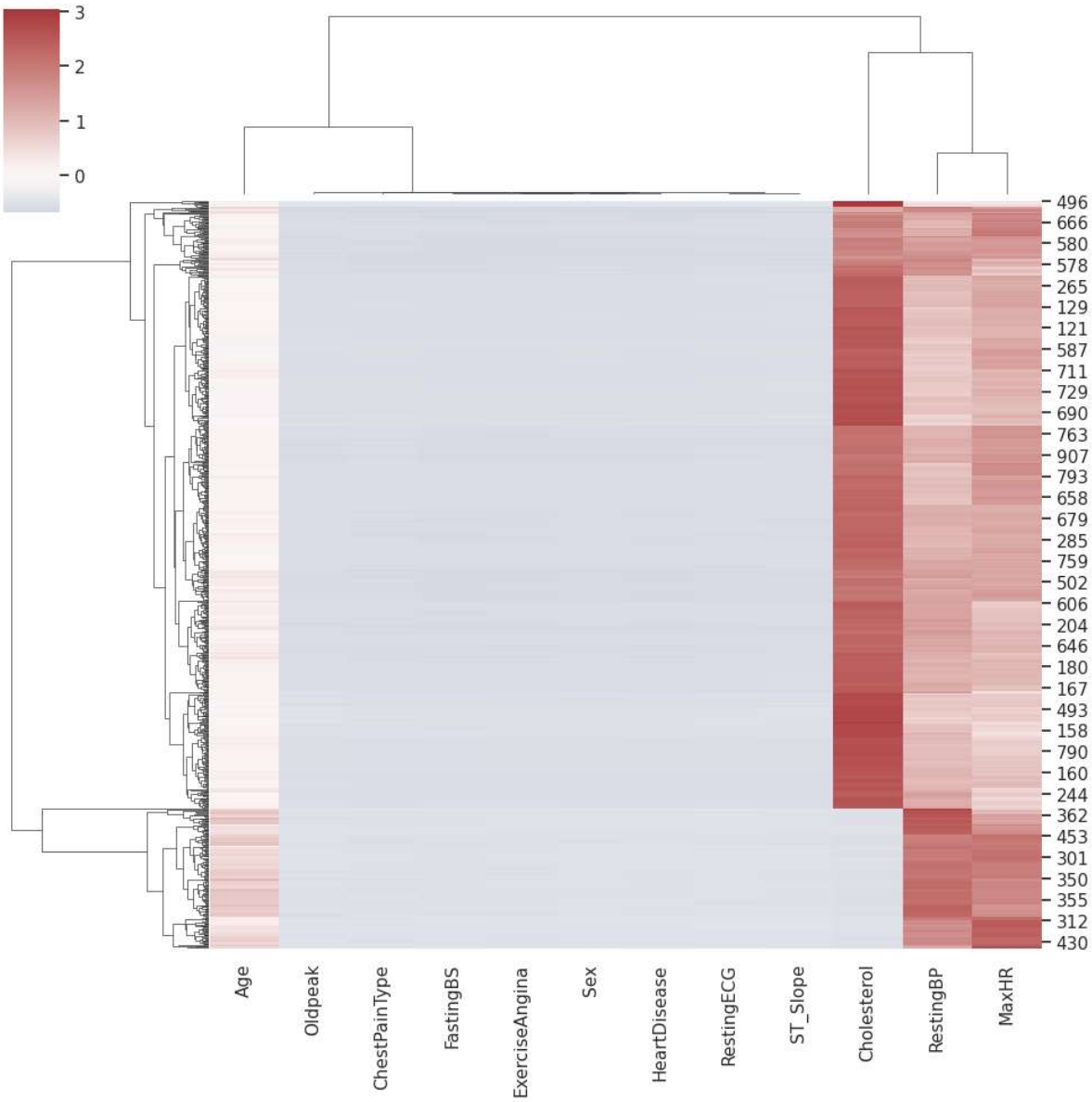
In [193]:
```
for i in df.columns:
    print("**************************************************",i,
          "**************************************************")
    print()
    print(set(df[i].tolist()))
    print()
```

```
********************************************************** Age ***************
************************************

{28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 6
6, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77}

********************************************************** Sex ***************
************************************

{0, 1}

********************************************************** ChestPainType *****
***********************************************

{0, 1, 2, 3}

********************************************************** RestingBP *********
*******************************************

{128, 0, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 14
3, 144, 145, 146, 129, 148, 150, 152, 154, 155, 156, 158, 160, 164, 165, 170,
172, 174, 178, 180, 185, 190, 192, 200, 80, 92, 94, 95, 96, 98, 100, 101, 10
2, 104, 105, 106, 108, 110, 112, 113, 114, 115, 116, 117, 118, 120, 122, 123,
124, 125, 126, 127}

********************************************************** Cholesterol *******
***********************************************

{0, 518, 529, 564, 85, 603, 100, 110, 113, 117, 123, 126, 129, 131, 132, 139,
141, 142, 147, 149, 152, 153, 156, 157, 159, 160, 161, 163, 164, 165, 166, 16
7, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
183, 184, 185, 186, 187, 188, 190, 192, 193, 194, 195, 196, 197, 198, 199, 20
0, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215,
216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 23
1, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 26
2, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277,
278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 29
3, 294, 295, 297, 298, 299, 300, 302, 303, 304, 305, 306, 307, 308, 309, 310,
311, 312, 313, 315, 316, 318, 319, 320, 321, 322, 325, 326, 327, 328, 329, 33
0, 331, 333, 335, 336, 337, 338, 339, 340, 341, 342, 344, 347, 349, 353, 354,
355, 358, 360, 365, 369, 384, 385, 388, 392, 393, 394, 404, 407, 409, 412, 41
7, 458, 466, 468, 491}

********************************************************** FastingBS *********
*********************************************

{0, 1}

********************************************************** RestingECG ********
*********************************************

{0, 1, 2}

********************************************************** MaxHR *************
*************************************
```

```
{60, 63, 67, 69, 70, 71, 72, 73, 77, 78, 80, 82, 83, 84, 86, 87, 88, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100, 102, 103, 104, 105, 106, 107, 108, 109,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 12
5, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140,
141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 15
6, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171,
172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 184, 185, 186, 187, 18
8, 190, 192, 194, 195, 202}

****************************************************** ExerciseAngina ****
************************************************

{0, 1}

******************************************************** Oldpeak ***********
*****************************************

{0.0, 1.0, 2.0, 3.0, 1.5, 4.0, 0.5, 2.5, 5.0, 0.8, 1.4, 2.1, 3.7, -0.5, 5.6,
6.2, 3.5, 4.4, -0.9, 0.6, 1.6, 1.1, 2.6, 3.1, 0.7, -0.8, 3.6, 0.2, 1.7, 1.2,
0.1, -0.1, 2.2, 2.8, 0.3, 2.3, 3.2, -0.7, 3.8, 4.2, 1.3, 1.8, -1.1, 1.9, 0.4,
0.9, 2.4, 2.9, 3.4, -1.0, -2.0, -2.6, -1.5}

******************************************************** ST_Slope **********
*****************************************

{0, 1, 2}

****************************************************** HeartDisease ******
***********************************************

{0, 1}
```

In [194]:
```
sns.clustermap(df, z_score=0, cmap="vlag", center=0)
```

Out[194]: <seaborn.matrix.ClusterGrid at 0x7e615ca59ab0>



In [195]:
```
df.head()
```

Out[195]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	

In [196]:
```
df.columns
```

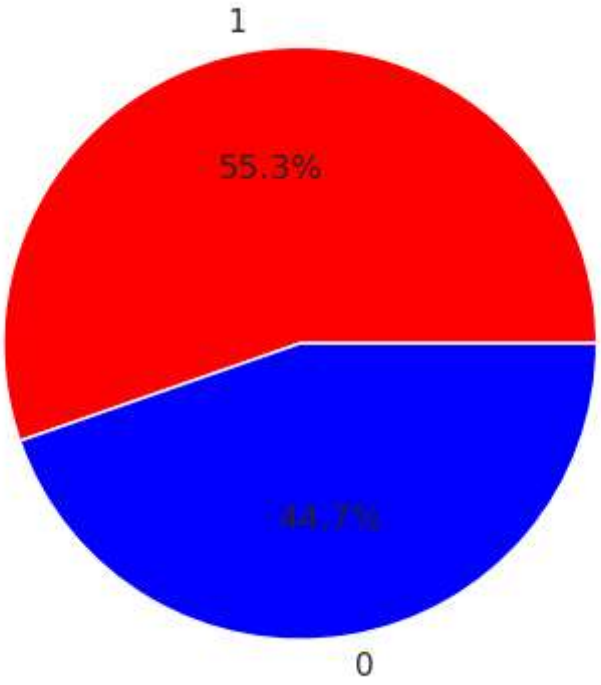
Out[196]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingB
S',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

Check HeartDisease column

In [197]:
```
df['HeartDisease'].value_counts()
```

Out[197]:
```
1    508
0    410
Name: HeartDisease, dtype: int64
```

In [198]:
```
HeartDisease = [1,0]
quantity = [508,410]
plt.pie(quantity,labels=HeartDisease,autopct='%0.1f%%',colors=['red','blue'])
plt.show()
```

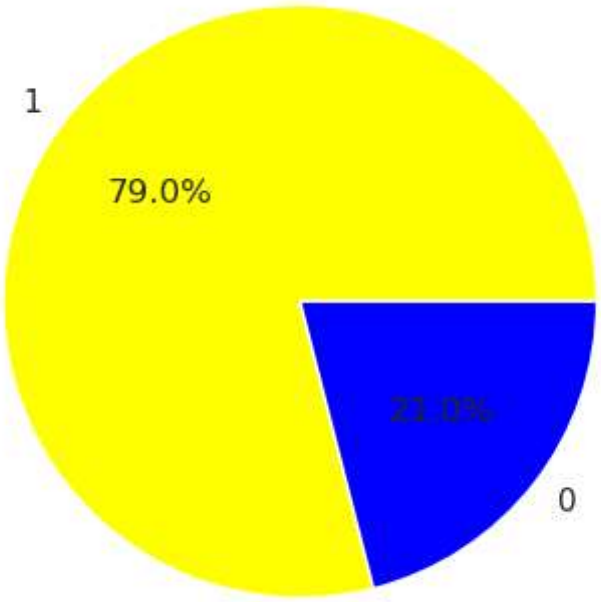


Check Sex column

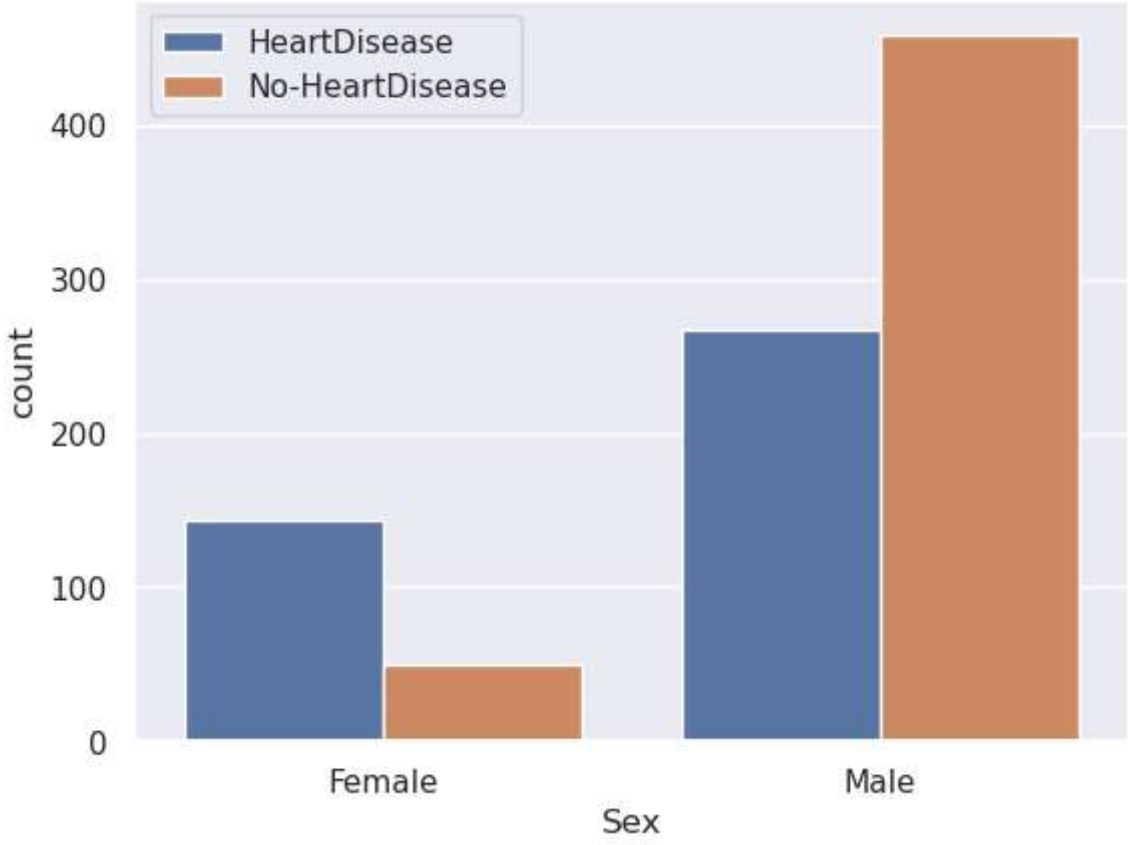
In [199]:
```
df['Sex'].value_counts()
```

Out[199]:
```
1    725
0    193
Name: Sex, dtype: int64
```

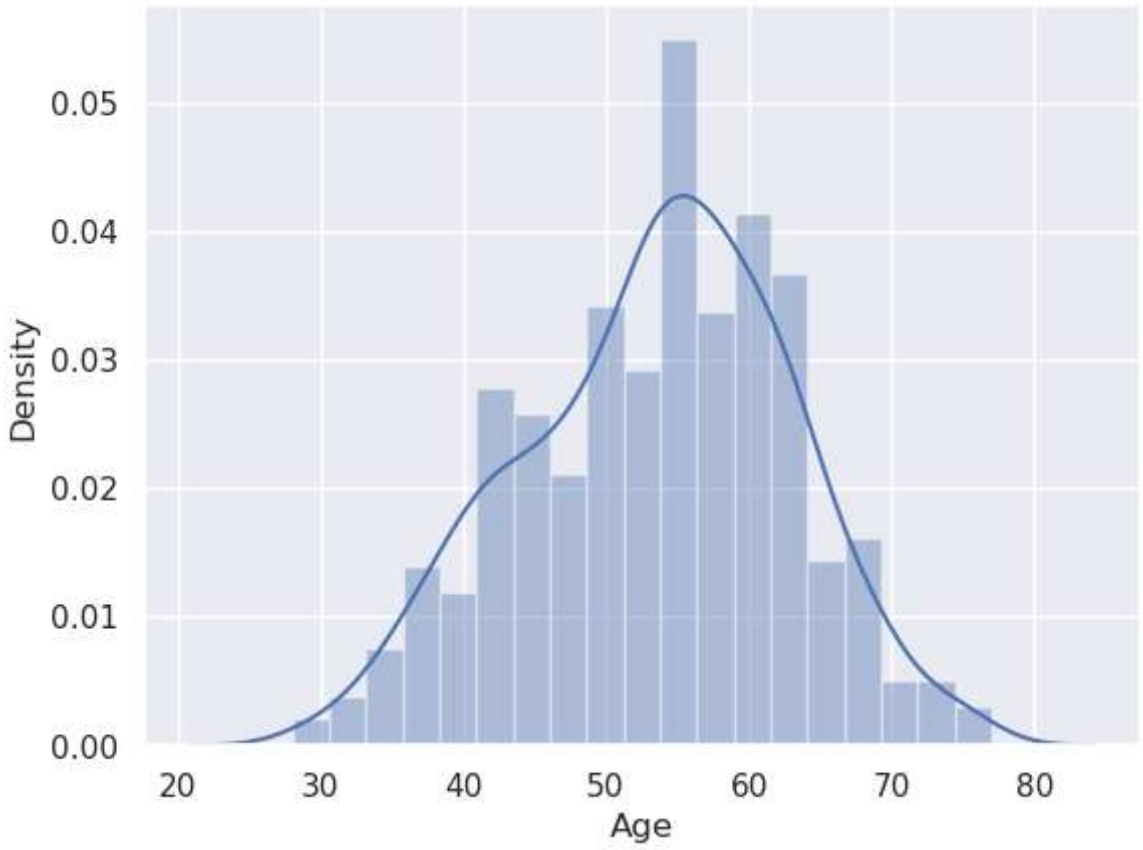
In [200]:
```
Sex = [1,0]
quantity = [725,193]
plt.pie(quantity,labels=Sex,autopct='%0.1f%%',colors=['yellow','blue'])
plt.show()
```



In [201]:
```
sns.countplot(x="Sex",hue="HeartDisease",data=df)
plt.xticks([1,0],['Male','Female'])
plt.legend(labels=['HeartDisease','No-HeartDisease'])
plt.show()
```



In [202]:
```
sns.distplot(df['Age'])
plt.show()
```

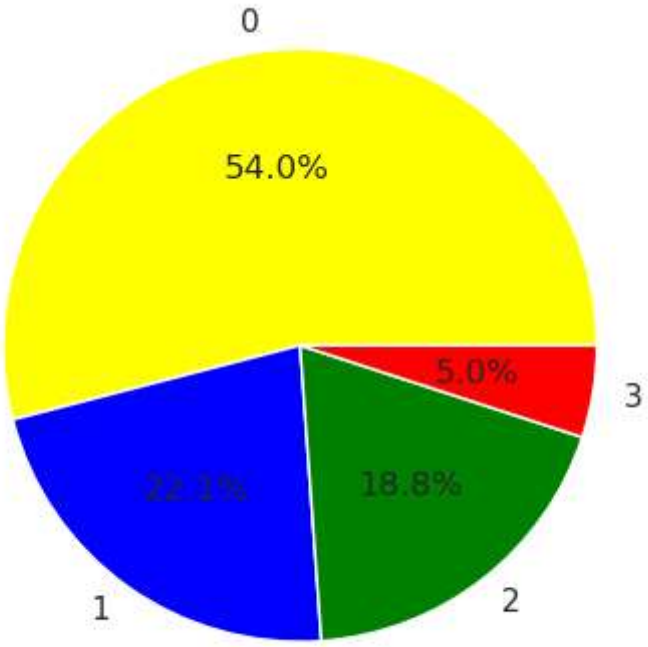


Check pain Type

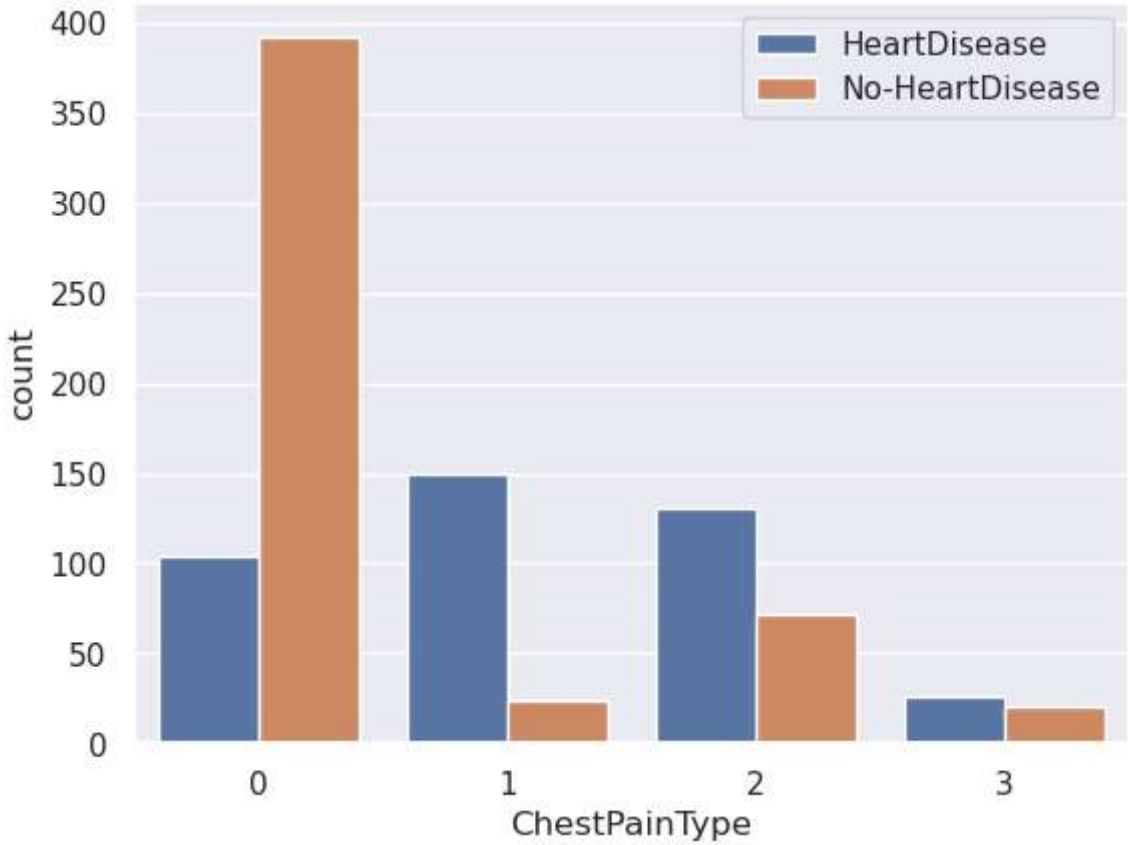
In [203]:
```
df['ChestPainType'].value_counts()
```

Out[203]:
```
0    496
2    203
1    173
3     46
Name: ChestPainType, dtype: int64
```

In [204]:
```
ChestPainType = [0,1,2,3]
quantity = [496,203,173,46]
plt.pie(quantity,labels=ChestPainType,autopct='%0.1f%%',colors=['yellow','blue
plt.show()
```



In [205]:
```
sns.countplot(x="ChestPainType",hue="HeartDisease",data=df)
plt.xticks([0,1,2,3])
plt.legend(labels=['HeartDisease','No-HeartDisease'])
plt.show()
```



In [206]:
```
df.columns
```

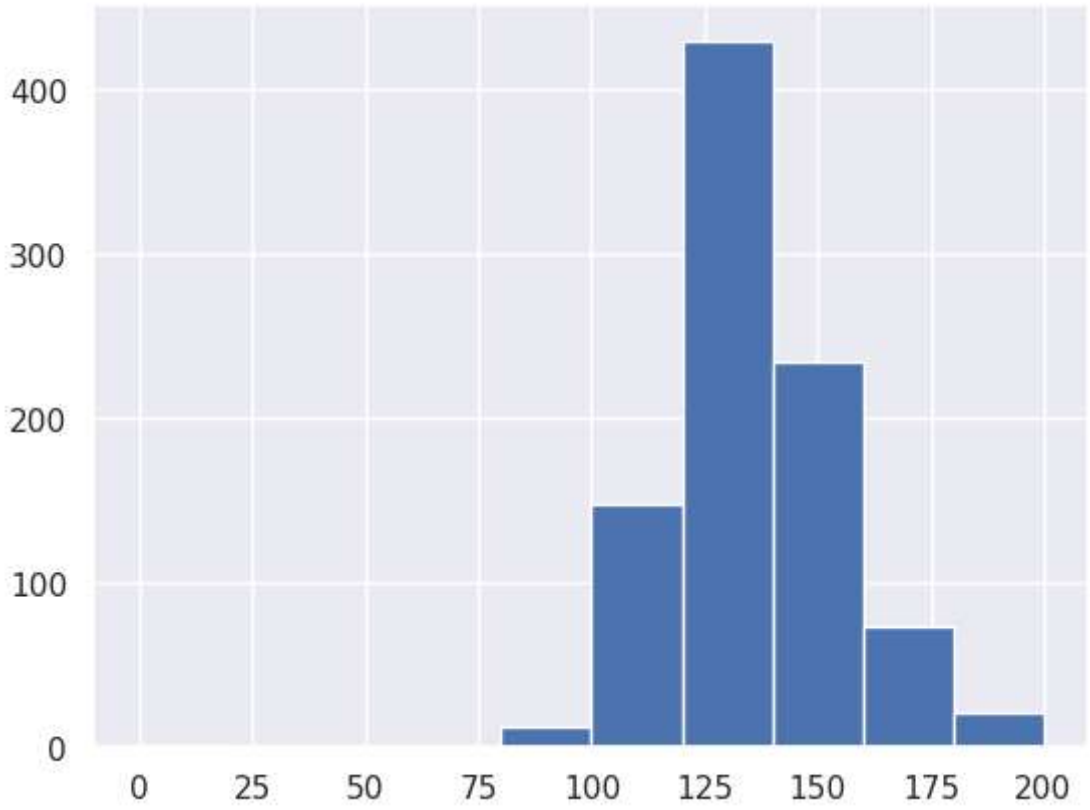
Out[206]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingB
S',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

Check RestingBP

In [207]:
```
df['RestingBP'].value_counts()
```

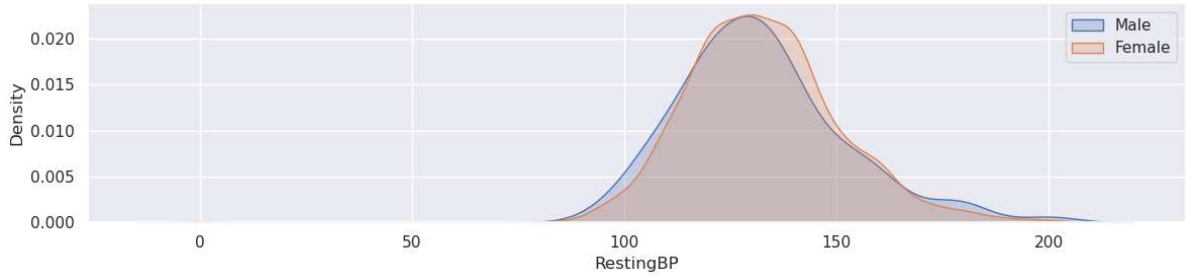
Out[207]:
```
120    132
130    118
140    107
110     58
150     55
      ...
185      1
98       1
92       1
113      1
164      1
Name: RestingBP, Length: 67, dtype: int64
```

In [208]:
```
df['RestingBP'].hist()
plt.show()
```



compair RestingBP and sex

In [209]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'RestingBP',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```



In [210]:
```
df.columns
```

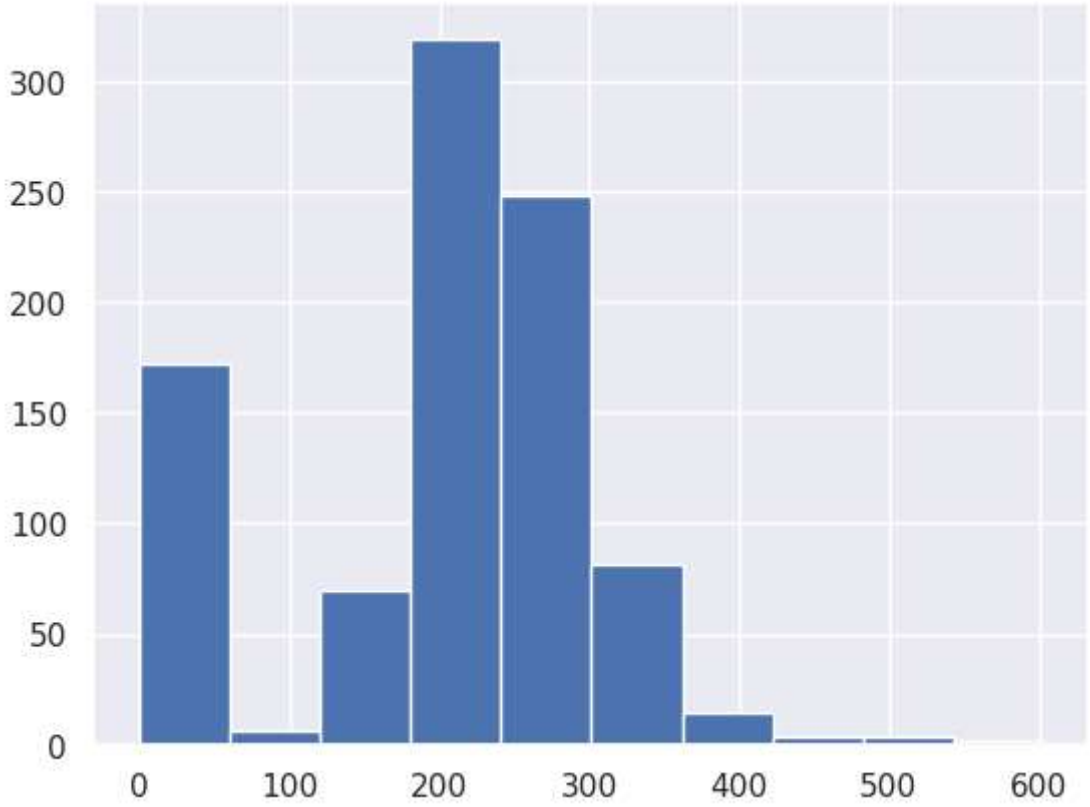
Out[210]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingB
S',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

Check Cholesterol

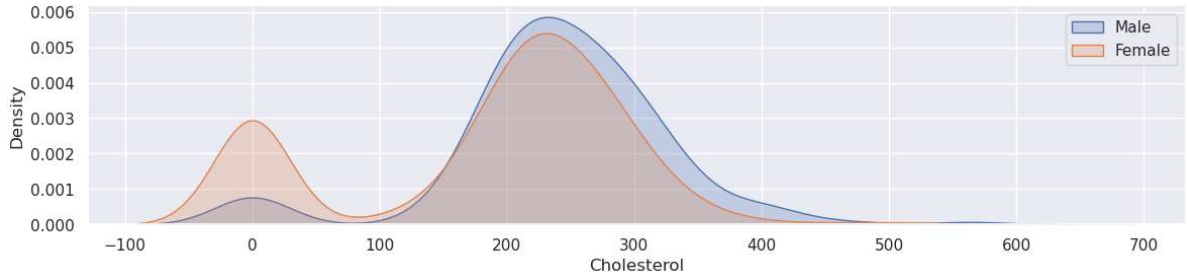
In [211]:
```
df['Cholesterol'].value_counts()
```

Out[211]:
```
0      172
254     11
223     10
220     10
230      9
       ...
392      1
316      1
153      1
466      1
131      1
Name: Cholesterol, Length: 222, dtype: int64
```

In [212]:
```
df['Cholesterol'].hist()
plt.show()
```



In [213]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'Cholesterol',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

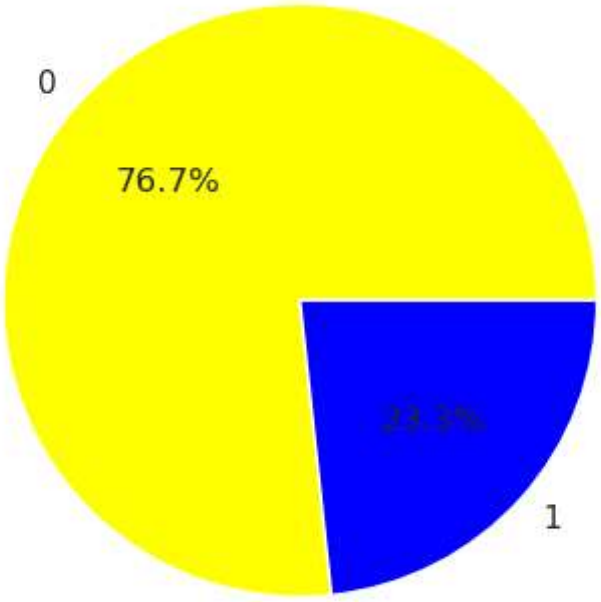


Check FastingBS

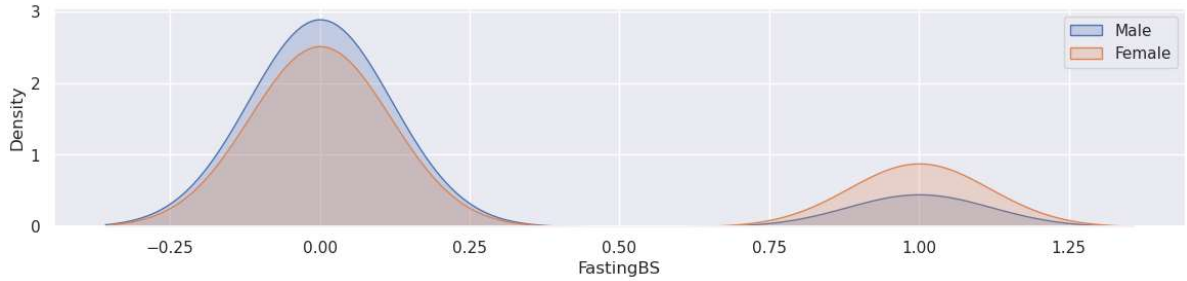
In [214]:
```
df['FastingBS'].value_counts()
```

Out[214]:
```
0    704
1    214
Name: FastingBS, dtype: int64
```

In [215]:
```
FastingBS = [0,1]
quantity = [704,214]
plt.pie(quantity,labels=FastingBS,autopct='%0.1f%%',colors=['yellow','blue'])
plt.show()
```



In [216]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'FastingBS',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

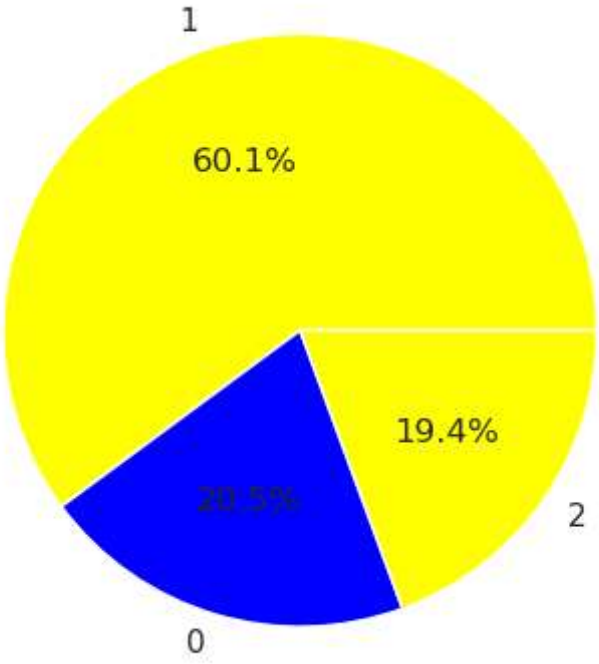


check RestingECG

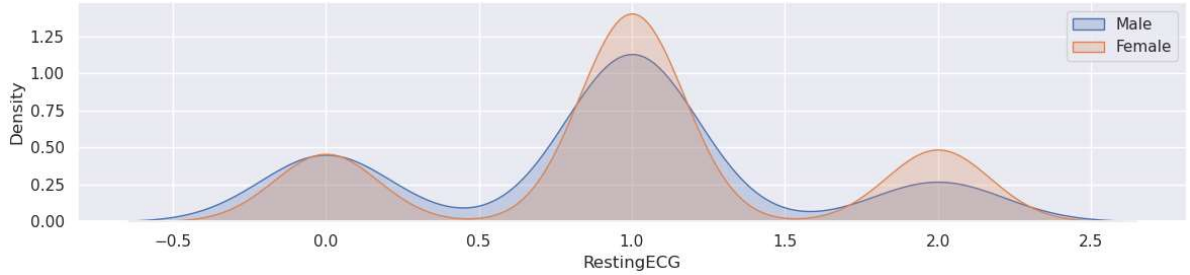
In [217]:
```
df['RestingECG'].value_counts()
```

Out[217]:
```
1    552
0    188
2    178
Name: RestingECG, dtype: int64
```

In [218]:
```
RestingECG = [1,0,2]
quantity = [552,188,178]
plt.pie(quantity,labels=RestingECG,autopct='%0.1f%%',colors=['yellow','blue'])
plt.show()
```



In [219]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'RestingECG',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

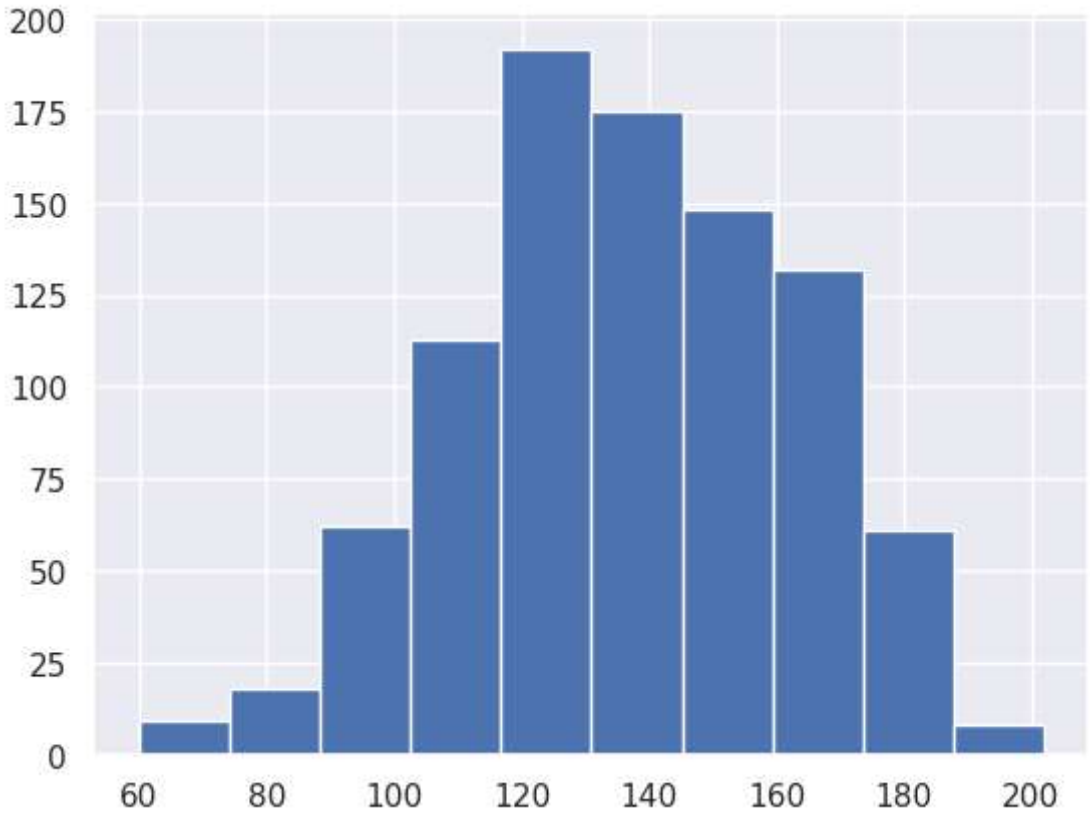


check MaxHR

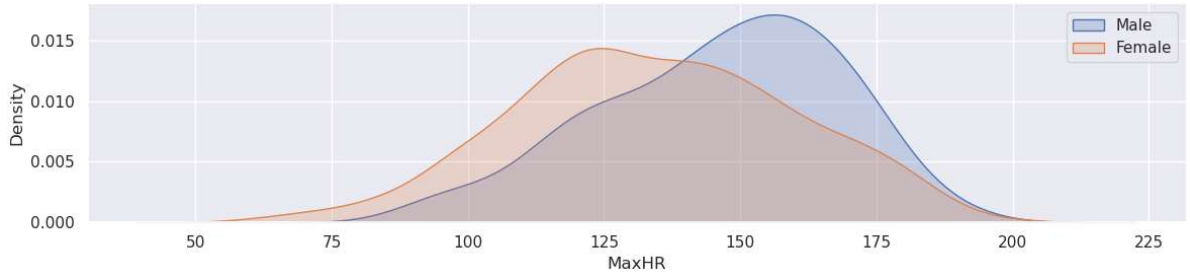
In [220]:
```
df['MaxHR'].value_counts()
```

Out[220]:
```
150    43
140    41
120    36
130    33
160    25
       ..
63      1
83      1
60      1
78      1
202     1
Name: MaxHR, Length: 119, dtype: int64
```

In [221]:
```
df['MaxHR'].hist()
plt.show()
```



In [222]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'MaxHR',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

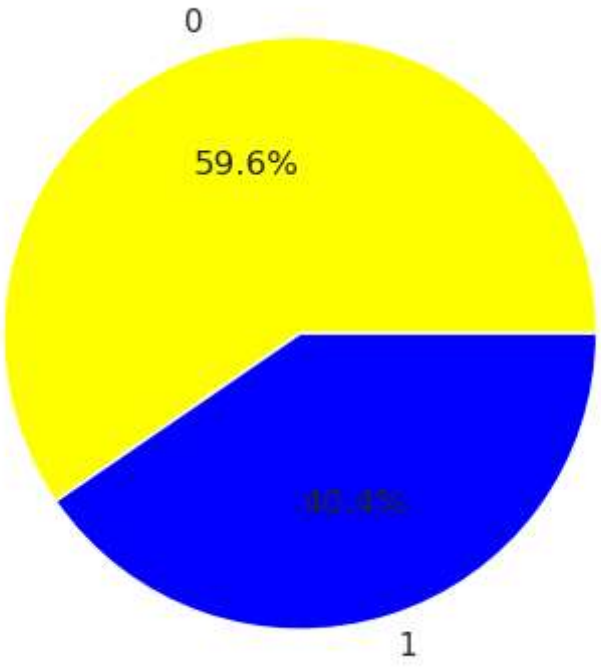


check ExerciseAngina**

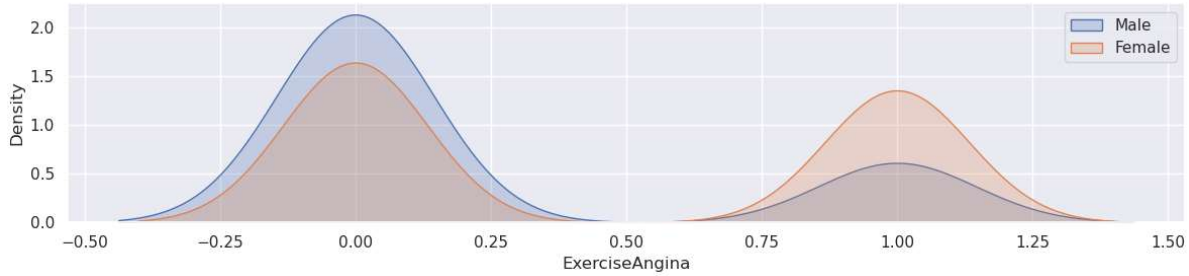
In [223]:
```
df['ExerciseAngina'].value_counts()
```

Out[223]:
```
0    547
1    371
Name: ExerciseAngina, dtype: int64
```

In [224]:
```
ExerciseAngina = [0,1]
quantity = [547,371]
plt.pie(quantity,labels=ExerciseAngina,autopct='%0.1f%%',colors=['yellow','blu
plt.show()
```



In [225]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'ExerciseAngina',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

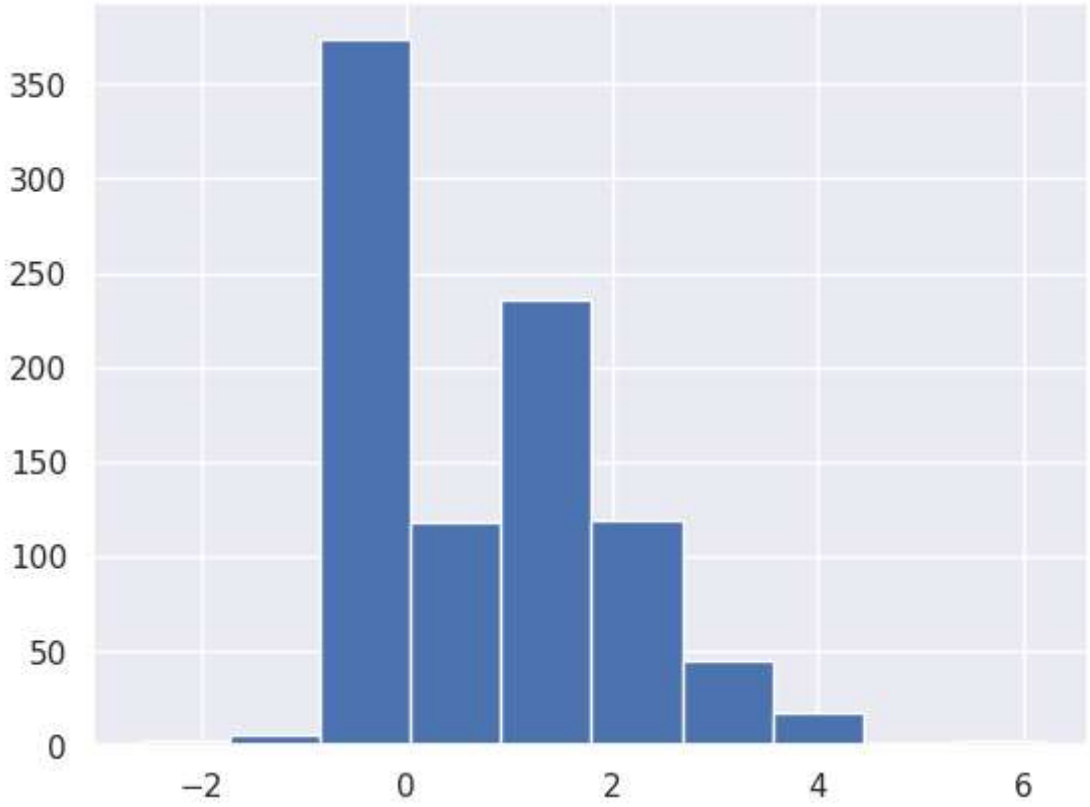


Check Oldpeak

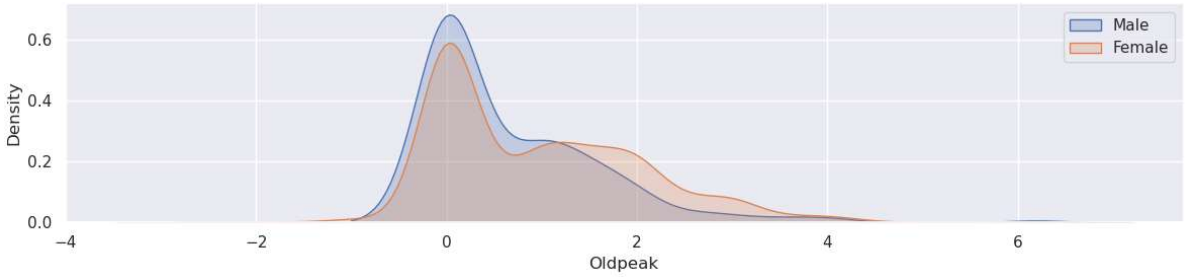
In [226]:
```
df['Oldpeak'].value_counts()
```

Out[226]:
```
 0.0    368
 1.0     86
 2.0     76
 1.5     53
 3.0     28
 1.2     26
 0.2     22
 0.5     19
 1.4     18
 1.8     17
 2.5     16
 0.8     16
 1.6     16
 0.1     14
 0.6     14
 0.4     11
 0.3     11
 4.0      8
 0.7      7
 2.8      7
 1.9      7
 1.3      7
 2.6      7
 1.1      7
 1.7      6
 2.2      5
 0.9      4
 2.4      4
 3.6      4
 3.4      3
 4.2      2
 3.5      2
-0.5      2
 2.3      2
 3.2      2
 2.1      2
-1.0      2
-0.1      2
 5.6      1
 2.9      1
 6.2      1
 3.8      1
-1.5      1
 3.1      1
-2.0      1
 3.7      1
-0.8      1
-0.7      1
-1.1      1
-2.6      1
-0.9      1
 5.0      1
 4.4      1
Name: Oldpeak, dtype: int64
```

In [227]:
```
df['Oldpeak'].hist()
plt.show()
```



In [228]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'Oldpeak',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```

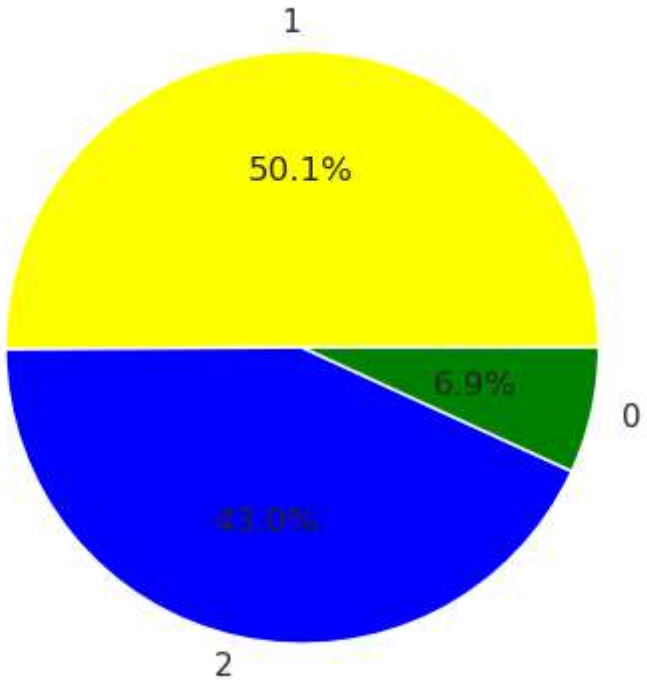


check ST_Slope

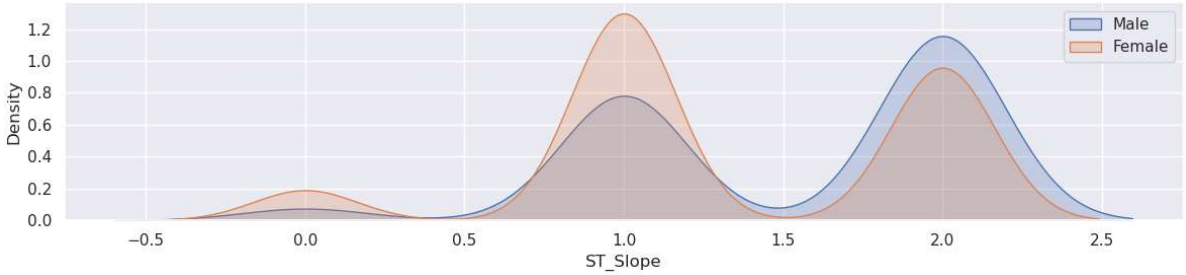
In [229]:
```
df['ST_Slope'].value_counts()
```

Out[229]:
```
1    460
2    395
0     63
Name: ST_Slope, dtype: int64
```

In [230]:
```
ST_Slope= [1,2,0]
quantity = [460,395,63]
plt.pie(quantity,labels=ST_Slope,autopct='%0.1f%%',colors=['yellow','blue','gr
plt.show()
```



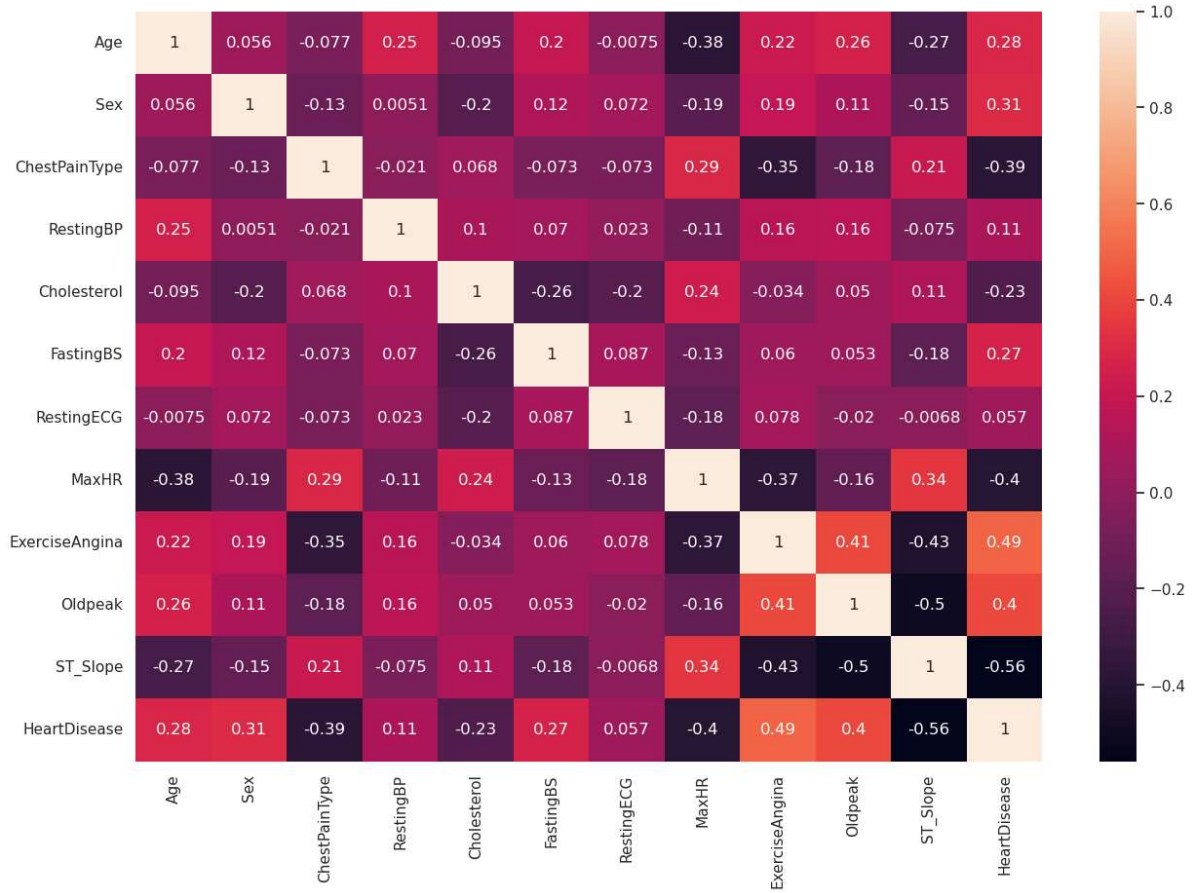
In [231]:
```
g = sns.FacetGrid(df,hue="Sex",aspect=4)
g.map(sns.kdeplot,'ST_Slope',shade=True)
plt.legend(labels=['Male','Female'])
plt.show()
```



Correlation Matrix

In [232]:
```
plt.figure(figsize= (15,10))
sns.heatmap(df.corr(),annot=True)
```

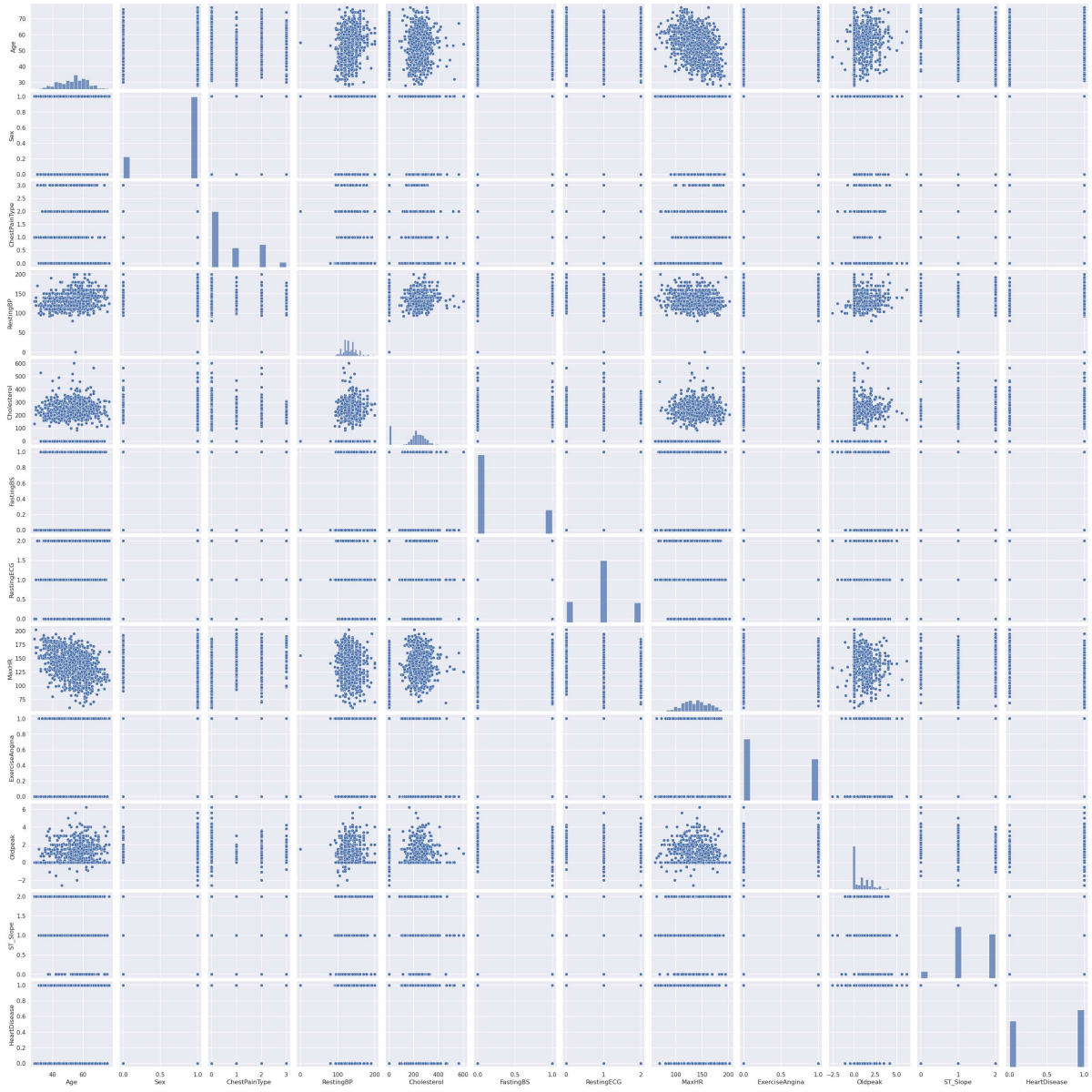
Out[232]: <Axes: >



Pairplot

In [233]:
```
sns.pairplot(df)
```

Out[233]:
```
<seaborn.axisgrid.PairGrid at 0x7e615b343430>
```



Feature Scaling

In [323]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_percentage_error, mean_squa
from sklearn import metrics
```

split the data dependent and independent variable**

In [236]:
```
x=df.drop(['HeartDisease'],axis=1)
y=df['HeartDisease']
```

In [237]:
```
x.head()
```

Out[237]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseA
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	

In [238]:
```
y.head()
```

Out[238]:
```
0    0
1    1
2    0
3    1
4    0
Name: HeartDisease, dtype: int64
```

In [239]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[239]:

	0	1	2	3	4	5	6	7	
0	-1.433140	0.515952	0.229032	0.410909	0.825070	-0.551341	0.017255	1.382928	-0.823
1	-0.478484	-1.938163	1.275059	1.491752	-0.171961	-0.551341	0.017255	0.754157	-0.823
2	-1.751359	0.515952	0.229032	-0.129513	0.770188	-0.551341	1.601219	-1.525138	-0.823
3	-0.584556	-1.938163	-0.816995	0.302825	0.139040	-0.551341	0.017255	-1.132156	1.214
4	0.051881	0.515952	1.275059	0.951331	-0.034755	-0.551341	0.017255	-0.581981	-0.823
...	...	...	...	...	...	...	...	...	
913	-0.902775	0.515952	2.321086	-1.210356	0.596393	-0.551341	0.017255	-0.188999	-0.823
914	1.536902	0.515952	-0.816995	0.627078	-0.053049	1.813758	0.017255	0.164684	-0.823
915	0.370100	0.515952	-0.816995	-0.129513	-0.620168	-0.551341	0.017255	-0.857069	1.214
916	0.370100	-1.938163	0.229032	-0.129513	0.340275	-0.551341	-1.566710	1.461525	-0.823
917	-1.645286	0.515952	1.275059	0.302825	-0.217696	-0.551341	0.017255	1.422226	-0.823

918 rows × 11 columns

VIF Variance Inflation Factor

In [240]:
```
variable = sc_x
variable.shape
```

Out[240]: (918, 11)

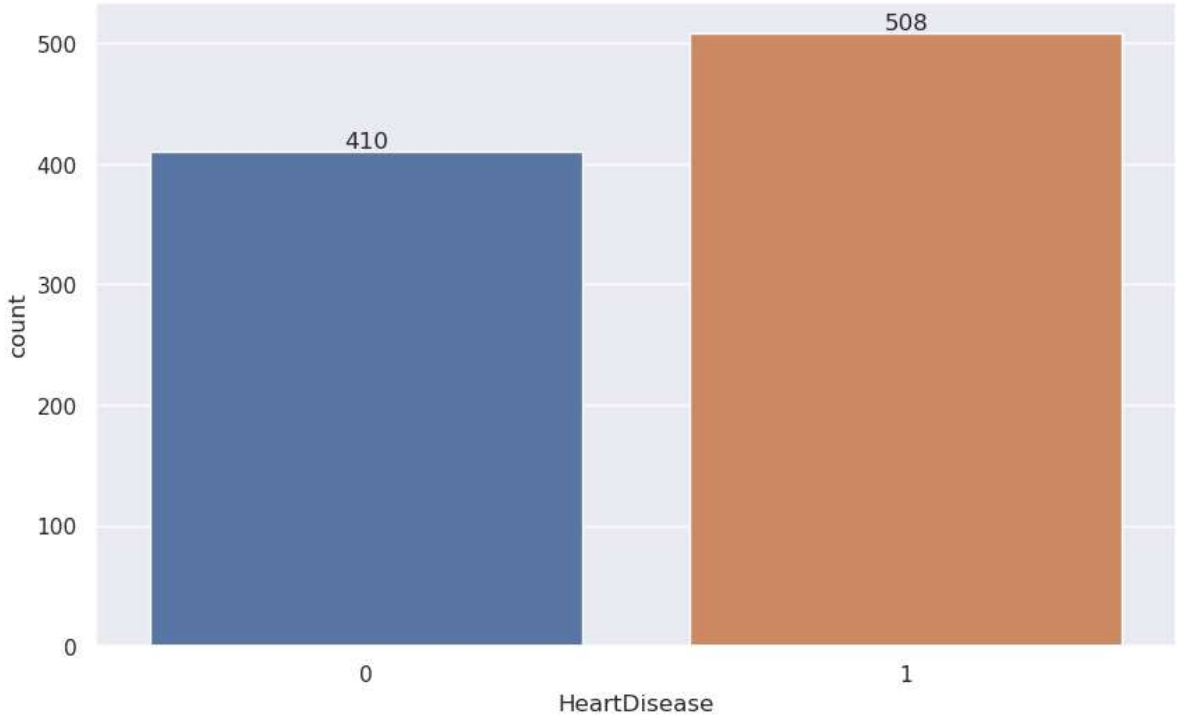
In [241]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable = sc_x

vif = pd.DataFrame()

vif['Variance Inflation Factor'] = [variance_inflation_factor(variable, i ) fo

vif['Features'] = x.columns
```

In [242]:
```
vif
```

Out[242]:

	Variance Inflation Factor	Features
0	1.336288	Age
1	1.098135	Sex
2	1.197203	ChestPainType
3	1.115990	RestingBP
4	1.216232	Cholesterol
5	1.138110	FastingBS
6	1.079094	RestingECG
7	1.510853	MaxHR
8	1.531765	ExerciseAngina
9	1.505506	Oldpeak
10	1.594313	ST_Slope

In [243]:
```
plt.figure(figsize=(10,6),dpi=100)
ax=sns.countplot(x='HeartDisease',data=df)

for i in ax.containers:
    ax.bar_label(i)
```



Split the data into training and test for building the model and for prediction

In [244]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rando
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(642, 11) (276, 11) (642,) (276,)
```

Logistic Regression Model

In [245]:
```
from sklearn.linear_model import LogisticRegression
```

In [246]:
```
logit = LogisticRegression()
logit.fit(x_train, y_train)
```

Out[246]:
```
LogisticRegression
LogisticRegression()
```

Predict the data

In [247]:
```
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
```

Evaluate the model

In [248]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_
```

In [249]:
```
print("Trainging Accuracy Score :", accuracy_score(y_train, y_pred_train))
print("****************"*10)
print("Test Accuracy Score :", accuracy_score(y_test, y_pred_test))
```

```
Trainging Accuracy Score : 0.8629283489096573
**********************************************************************
**********************************************************************
******
Test Accuracy Score : 0.8297101449275363
```

In [250]:
```
print( classification_report(y_train, y_pred_train))
print("****************"*10)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84       284
           1       0.88      0.88      0.88       358

    accuracy                           0.86       642
   macro avg       0.86      0.86      0.86       642
weighted avg       0.86      0.86      0.86       642

**********************************************************************
**********************************************************************
******
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       126
           1       0.84      0.85      0.84       150

    accuracy                           0.83       276
   macro avg       0.83      0.83      0.83       276
weighted avg       0.83      0.83      0.83       276
```

In [251]:
```
print( confusion_matrix(y_train, y_pred_train))
print("****************"*10)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[239  45]
 [ 43 315]]
**********************************************************************
**********************************************************************
******
[[101  25]
 [ 22 128]]
```

In [252]:
```
y_pred_test
```

Out[252]:
```
array([0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1])
```

In [253]:
```
y_test
```

Out[253]:
```
417    0
325    1
267    0
241    1
367    1
      ..
196    0
83     0
258    0
290    0
407    1
Name: HeartDisease, Length: 276, dtype: int64
```

DecisionTree

In [254]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
from sklearn.tree import export_text
```

In [255]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
dt_clf = DecisionTreeClassifier()
dt_clf.fit(x_train, y_train)
```

Out[255]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [256]:
```
# predict
y_pred_train = dt_clf.predict(x_train)
y_pred_test = dt_clf.predict(x_test)
```

In [257]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [258]:
```
confusion_matrix(y_test, y_pred_test)
```

Out[258]:
```
array([[104,  22],
       [ 35, 115]])
```

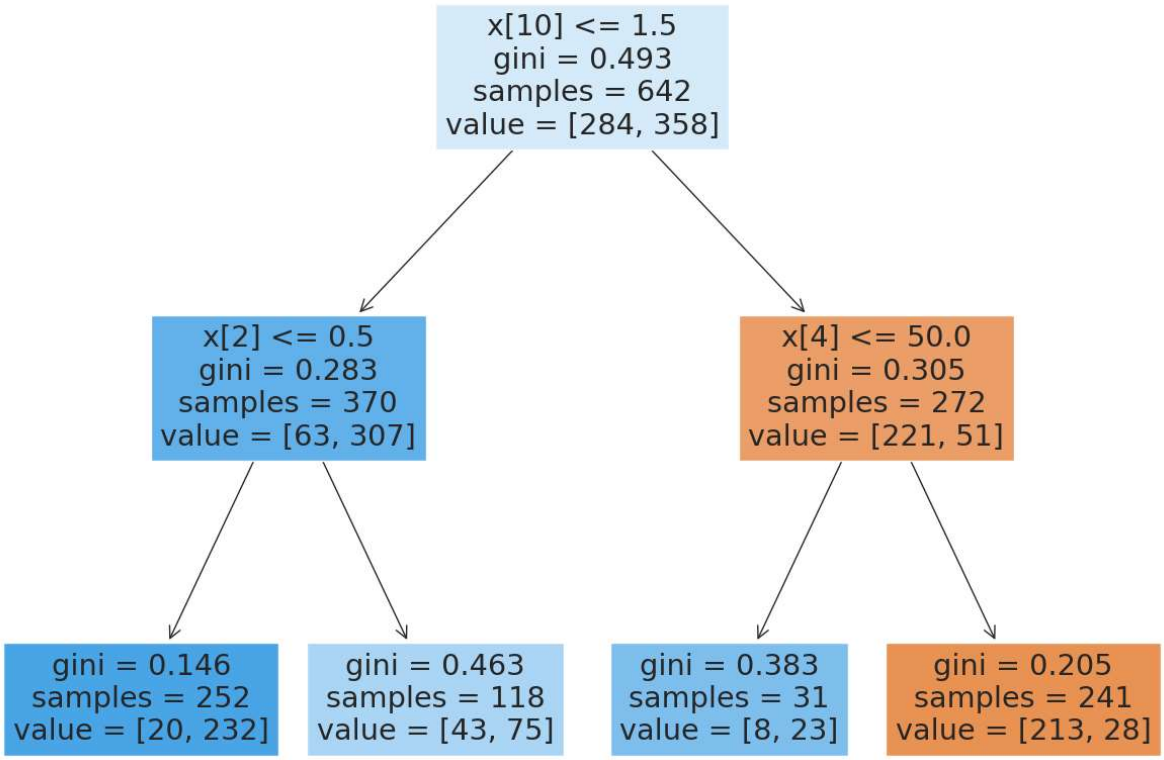
In [259]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.83      0.78       126
           1       0.84      0.77      0.80       150

    accuracy                           0.79       276
   macro avg       0.79      0.80      0.79       276
weighted avg       0.80      0.79      0.79       276
```

In [260]:
```
print(accuracy_score(y_train, y_pred_train))
print()
print(accuracy_score(y_test, y_pred_test))
```

```
1.0

0.7934782608695652
```

In [261]:
```
# plot the decisionTree Classifier
dt_clf = DecisionTreeClassifier(max_depth=2)
dt_clf.fit(x_train, y_train)

y_pred_train = dt_clf.predict(x_train)
y_pred_test = dt_clf.predict(x_test)


from sklearn.tree import plot_tree
from sklearn.tree import export_text
plt.figure(figsize=(15,12))
plot_tree(dt_clf, filled=True, feature_names=None,
          class_names=None)
plt.show()
```



In [262]:
```
print(accuracy_score(y_train, y_pred_train))
print()
print(accuracy_score(y_test, y_pred_test))
```

```
0.8457943925233645

0.8115942028985508
```

Building Bagging Algorithm

In [270]:
```
# split the data into training and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, rand
```

In [271]:
```
from sklearn.ensemble import BaggingClassifier
bagging = BaggingClassifier()
bagging.fit(x_train, y_train)
```

Out[271]:
```
BaggingClassifier

BaggingClassifier()
```

In [272]:
```
# predict
y_pred_train_bgg = bagging.predict(x_train)
y_pred_test_bgg = bagging.predict(x_test)
```

In [274]:
```
# Evaluate
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [275]:
```
print(confusion_matrix(y_train, y_pred_train_bgg))
print()
print(confusion_matrix(y_test, y_pred_test_bgg))
```

```
[[305   4]
 [  3 376]]

[[ 88  13]
 [ 23 106]]
```

In [276]:
```
print(classification_report(y_train, y_pred_train_bgg))
print()
print(classification_report(y_test, y_pred_test_bgg))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       309
           1       0.99      0.99      0.99       379

    accuracy                           0.99       688
   macro avg       0.99      0.99      0.99       688
weighted avg       0.99      0.99      0.99       688


              precision    recall  f1-score   support

           0       0.79      0.87      0.83       101
           1       0.89      0.82      0.85       129

    accuracy                           0.84       230
   macro avg       0.84      0.85      0.84       230
weighted avg       0.85      0.84      0.84       230
```

In [277]:
```
print(accuracy_score(y_train, y_pred_train_bgg))
print()
print(accuracy_score(y_test, y_pred_test_bgg))
```

```
0.9898255813953488

0.8434782608695652
```

RandomForest Classifier Model

In [278]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200, criterion='entropy',
                            bootstrap=True, oob_score=False)
rf.fit(x_train, y_train)
```

Out[278]:
```
RandomForestClassifier

RandomForestClassifier(criterion='entropy', n_estimators=200)
```

In [279]:
```
# predict
y_pred_train_rf = rf.predict(x_train)
y_pred_test_rf = rf.predict(x_test)
```

In [280]:
```
print(confusion_matrix(y_train, y_pred_train_rf))
print()
print(confusion_matrix(y_test, y_pred_test_rf))
```

```
[[309   0]
 [  0 379]]

[[ 85  16]
 [ 18 111]]
```

In [281]:
```
print(classification_report(y_train, y_pred_train_rf))
print()
print(classification_report(y_test, y_pred_test_rf))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       309
           1       1.00      1.00      1.00       379

    accuracy                           1.00       688
   macro avg       1.00      1.00      1.00       688
weighted avg       1.00      1.00      1.00       688


              precision    recall  f1-score   support

           0       0.83      0.84      0.83       101
           1       0.87      0.86      0.87       129

    accuracy                           0.85       230
   macro avg       0.85      0.85      0.85       230
weighted avg       0.85      0.85      0.85       230
```

In [282]:
```
print(accuracy_score(y_train, y_pred_train_rf))
print()
print(accuracy_score(y_test, y_pred_test_rf))
```

```
1.0

0.8521739130434782
```

In [283]:
```
# applying cross validation method in RandomForest
from sklearn.model_selection import cross_val_score
accuracy = cross_val_score(rf, x_train, y_train, cv=10)
accuracy.mean()
```

Out[283]:
```
0.8793904518329072
```

In [284]:
```
print("Trainging Accuracy : ", accuracy.mean())
print()
print("Test Accuracy: ", accuracy_score(y_test, y_pred_test_rf))
```

```
Trainging Accuracy :  0.8793904518329072

Test Accuracy:  0.8521739130434782
```

SVM

In [285]:
```
from sklearn.svm import SVC

#kernel - linear
svm_linear = SVC(kernel='linear')
svm_linear.fit(x_train, y_train)
y_pred_train_linear = svm_linear.predict(x_train)
y_pred_test_linear = svm_linear.predict(x_test)

#kernel - sigmoid
svm_sigmoid = SVC(kernel='sigmoid')
svm_sigmoid.fit(x_train, y_train)
y_pred_train_sigmoid = svm_sigmoid.predict(x_train)
y_pred_test_sigmoid = svm_sigmoid.predict(x_test)

#kernel - poly
svm_poly = SVC(kernel='poly')
svm_poly.fit(x_train, y_train)
y_pred_train_poly = svm_poly.predict(x_train)
y_pred_test_poly = svm_poly.predict(x_test)

#kernel - rbf
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
y_pred_train_rbf = svm_rbf.predict(x_train)
y_pred_test_rbf = svm_rbf.predict(x_test)
```

In [286]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [287]:
```
print("Training Accuracy - Linear :", accuracy_score(y_train, y_pred_train_lin
print("**********"*5)
print("Test Accuracy - Linear :", accuracy_score(y_test, y_pred_test_linear))
print("**********"*5)
print("Training Accuracy - sigmoid :", accuracy_score(y_train, y_pred_train_si
print("**********"*5)
print("Test Accuracy - sigmoid :", accuracy_score(y_test, y_pred_test_sigmoid)
print("**********"*5)
print("Training Accuracy - poly :", accuracy_score(y_train, y_pred_train_poly)
print("**********"*5)
print("Test Accuracy - poly :", accuracy_score(y_test, y_pred_test_poly))
print("**********"*5)
print("Training Accuracy - rbf :", accuracy_score(y_train, y_pred_train_rbf))
print("**********"*5)
print("Test Accuracy - rbf :", accuracy_score(y_test, y_pred_test_rbf))
```

```
Training Accuracy - Linear : 0.8677325581395349
**************************************************
Test Accuracy - Linear : 0.8260869565217391
**************************************************
Training Accuracy - sigmoid : 0.5261627906976745
**************************************************
Test Accuracy - sigmoid : 0.4826086956521739
**************************************************
Training Accuracy - poly : 0.7369186046511628
**************************************************
Test Accuracy - poly : 0.7086956521739131
**************************************************
Training Accuracy - rbf : 0.7354651162790697
**************************************************
Test Accuracy - rbf : 0.6826086956521739
```

In [288]:
```
print("Training Accuracy - Linear :", classification_report(y_train, y_pred_tr
print("**********"*5)
print("Test Accuracy - Linear :", classification_report(y_test, y_pred_test_li
```

```
Training Accuracy - Linear :               precision    recall  f1-score   su
pport

           0       0.86      0.84      0.85       309
           1       0.87      0.89      0.88       379

    accuracy                           0.87       688
   macro avg       0.87      0.87      0.87       688
weighted avg       0.87      0.87      0.87       688

**************************************************************
Test Accuracy - Linear :              precision    recall  f1-score   suppor
t

           0       0.80      0.80      0.80       101
           1       0.84      0.84      0.84       129

    accuracy                           0.83       230
   macro avg       0.82      0.82      0.82       230
weighted avg       0.83      0.83      0.83       230
```

In [289]:
```
# cross validation method
from sklearn.model_selection import cross_val_score
train_accuracy = cross_val_score(svm_linear, x_train, y_train, cv=10)
test_accuracy = cross_val_score(svm_linear, x_test, y_test, cv=10)
print("Training accuracy :", train_accuracy)
print("**********"*5)
print("Training Mean Accuracy :", train_accuracy.mean())
print("**********"*5)
print("Training Max Accuracy :", train_accuracy.max())


print("Test accuracy :", test_accuracy)
print("**********"*5)
print("Test Mean Accuracy :", test_accuracy.mean())
print("**********"*5)
print("Test Max Accuracy :", test_accuracy.max())
```

```
Training accuracy : [0.82608696 0.88405797 0.84057971 0.85507246 0.92753623
0.85507246
 0.89855072 0.84057971 0.86764706 0.83823529]
**************************************************
Training Mean Accuracy : 0.8633418584825234
**************************************************
Training Max Accuracy : 0.927536231884058
Test accuracy : [0.86956522 0.91304348 0.7826087  0.91304348 0.7826087  0.826
08696
 0.7826087  0.69565217 0.82608696 0.82608696]
**************************************************
Test Mean Accuracy : 0.8217391304347826
**************************************************
Test Max Accuracy : 0.9130434782608695
```

In [290]:
```
# Grid Search CV
from sklearn.model_selection import GridSearchCV
```

In [291]:
```
SVC()
```

Out[291]:
```
SVC()
```

In [292]:
```
param_grid = {'C':[0,1,2,10,100], 'gamma':[1,0.1,0.001,0.01]}
grid = GridSearchCV(SVC(), param_grid, refit=True)
grid.fit(x_train, y_train)
grid_predict = grid.predict(x_test)
print(accuracy_score(y_test, grid_predict))
```

```
0.7043478260869566
```

KNN

In [293]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

Out[293]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

In [294]:
```
y_pred_knn_train = knn.predict(x_train)
y_pred_knn_test = knn.predict(x_test)
```

In [295]:
```
print("Training Accuracy - knn :", accuracy_score(y_train, y_pred_knn_train))
print("**********"*5)
print("Test Accuracy - knn :", accuracy_score(y_test, y_pred_knn_test))
```

```
Training Accuracy - knn : 0.7863372093023255
**************************************************
Test Accuracy - knn : 0.691304347826087
```

Using Voting Method

In [300]:
```
estimators = [('SVM_Linear',svm_linear),('SVM_Sigmoid',svm_sigmoid),('SVM_Poly
              ("SVM_RBF" ,svm_rbf),('Logistic',logit),('KNN',knn)]
```

In [301]:
```
estimators1 = [('SVM_Linear',svm_linear),('Logistic',logit),('KNN',knn)]
```

In [302]:
```
from sklearn.ensemble import VotingClassifier
```

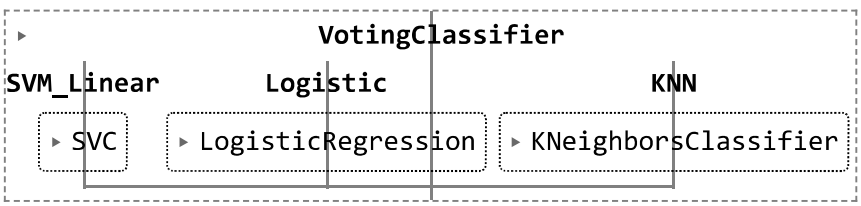
Hard Voting

In [303]:
```
voting_hard = VotingClassifier(estimators = estimators, voting='hard')
v_train_accuracy = cross_val_score(voting_hard, x_train, y_train, cv=10, scori
v_test_accuracy = cross_val_score(voting_hard, x_test, y_test, cv=10, scoring=
print(np.round(np.mean(v_train_accuracy),2))
print()
print(np.round(np.mean(v_test_accuracy),2))
```

```
0.78

0.76
```

In [304]:
```
voting_hard1 = VotingClassifier(estimators = estimators1, voting='hard')
voting_hard1.fit(x_train, y_train)
```

Out[304]:



In [305]:
```
y_pred_train_voting = voting_hard1.predict(x_train)
y_pred_test_voting = voting_hard1.predict(x_test)
```

In [306]:
```
print("Training Accuracy - voting_hard :", accuracy_score(y_train, y_pred_trai
print("**********"*5)
print("Test Accuracy - voting_hard :", accuracy_score(y_test, y_pred_test_voti
```

```
Training Accuracy - voting_hard : 0.8648255813953488
**************************************************
Test Accuracy - voting_hard : 0.8304347826086956
```

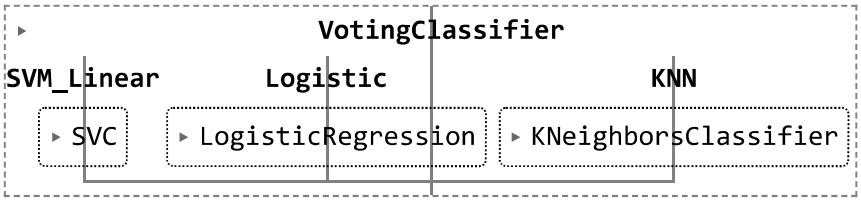
Soft Voting

In [307]:
```
voting_soft = VotingClassifier(estimators = estimators, voting='soft')
v_train_accuracy1 = cross_val_score(voting_soft, x_train, y_train, cv=10, scor
v_test_accuracy1 = cross_val_score(voting_soft, x_test, y_test, cv=10, scoring
print(np.round(np.mean(v_train_accuracy1),2))
print()
print(np.round(np.mean(v_test_accuracy1),2))
```

```
nan

nan
```

In [308]:
```
voting_soft1 = VotingClassifier(estimators = estimators1, voting='soft')
voting_soft1.fit(x_train, y_train)
```

Out[308]:



In [ ]:
```
y_pred_train_soft = voting_soft1.predict(x_train)
y_pred_test_soft = voting_soft1.predict(x_test)
```

In [315]:
```
print("Training Accuracy - voting_soft :", accuracy_score(y_train, y_pred_trai
print("**********"*5)
print("Test Accuracy - voting_soft :", accuracy_score(y_test, y_pred_test_voti
```

```
Training Accuracy - voting_soft : 0.8648255813953488
**************************************************
Test Accuracy - voting_soft : 0.8304347826086956
```

In [ ]:
```

```