Nassim Rafiefard

Code:27

Project: heartdata

EDA

Import all libraries

In [122…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats#
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix, classifi
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold

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

Import HeartData

In [123…
```
heartdata=pd.read_csv('/Users/nasimrafie/Documents/data science/Tehran Data/Python/1/Section 1/Datasets/Heart da
```

In [123…
```
heartdata.head()
```

Out[123…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	63	1	1	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	37	1	3	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
2	41	0	2	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
3	56	1	2	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0
4	57	0	4	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0

In [123…
```
heartdata.tail()
```

Out[123…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
592	52	1	4	140.0	266.0	0.0	0.0	134.0	1.0	2.0	2.0	NaN	NaN	1
593	43	1	4	140.0	288.0	0.0	0.0	135.0	1.0	2.0	2.0	NaN	NaN	1
594	41	1	4	120.0	336.0	0.0	0.0	118.0	1.0	3.0	2.0	NaN	NaN	1
595	44	1	4	135.0	491.0	0.0	0.0	135.0	0.0	0.0	NaN	NaN	NaN	1
596	49	1	4	150.0	222.0	0.0	0.0	122.0	0.0	2.0	2.0	NaN	NaN	1

Data has 587 rows and 14 columns

In [123…
```
heartdata.shape
```

Out[123…
```
(597, 14)
```

In [124…
```
print('There are', heartdata.shape[0], 'rows and', heartdata.shape[1], 'columns in heartdata.')
```

```
There are 597 rows and 14 columns in heartdata.
```

In [124…
```
heartdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 597 entries, 0 to 596
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Age (age in year)     597 non-null    int64  
 1   sex                   597 non-null    int64  
 2   chest pain            597 non-null    int64  
 3   blood pressure        596 non-null    float64
 4   cholestoral           574 non-null    float64
 5   blood sugar           589 non-null    float64
 6   electrocardiographic  596 non-null    float64
 7   heart rate            596 non-null    float64
 8   exercise induced      596 non-null    float64
 9   depression            597 non-null    float64
 10  slope                 407 non-null    float64
 11  ca                    303 non-null    float64
 12  thal                  329 non-null    float64
 13  c                     597 non-null    int64  
dtypes: float64(10), int64(4)
memory usage: 65.4 KB
```

In [124…
```
heartdata.describe().T
```

Out[124…

	count	mean	std	min	25%	50%	75%	max
Age (age in year)	597.0	51.182580	9.074366	28.0	44.0	52.0	58.00	77.0
sex	597.0	0.701843	0.457833	0.0	0.0	1.0	1.00	1.0
chest pain	597.0	3.072027	0.965776	1.0	2.0	3.0	4.00	4.0
blood pressure	596.0	132.129195	17.603812	92.0	120.0	130.0	140.00	200.0
cholestoral	574.0	248.655052	59.784805	85.0	211.0	242.5	278.75	603.0
blood sugar	589.0	0.110357	0.313600	0.0	0.0	0.0	0.00	1.0
electrocardiographic	596.0	0.610738	0.869358	0.0	0.0	0.0	2.00	2.0
heart rate	596.0	144.456376	23.794282	71.0	128.0	146.0	162.00	202.0
exercise induced	596.0	0.315436	0.465080	0.0	0.0	0.0	1.00	1.0
depression	597.0	0.816248	1.067938	0.0	0.0	0.2	1.50	6.2
slope	407.0	1.675676	0.572758	1.0	1.0	2.0	2.00	3.0
ca	303.0	0.693069	1.049212	0.0	0.0	0.0	1.00	9.0
thal	329.0	4.811550	1.928854	3.0	3.0	3.0	7.00	7.0
c	597.0	0.410385	0.492316	0.0	0.0	0.0	1.00	1.0

Dataset Attributes

Age : age of the patient [years]

Sex : sex of the patient [M: Male, F: Female]

ChestPainType : chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]

RestingBP : resting blood pressure [mm Hg]

Cholesterol : serum cholesterol [mm/dl]

FastingBS : fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]

RestingECG : resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]

MaxHR : maximum heart rate achieved [Numeric value between 60 and 202]

ExerciseAngina : exercise-induced angina [Y: Yes, N: No]

Oldpeak : oldpeak = ST [Numeric value measured in depression]

ST_Slope : the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]

HeartDisease : output class [1: heart disease, 0: Normal]

In [124…
```
heartdata.columns
```

Out[124…

```
Index(['Age (age in year)', 'sex', 'chest pain', 'blood pressure',
       'cholestoral ', 'blood sugar', 'electrocardiographic ', 'heart rate',
       'exercise induced', 'depression ', 'slope', 'ca', 'thal', 'c'],
      dtype='object')
```

Category Counts in Categorical Columns

In [125…

```python
cat_col=['sex','chest pain', 'electrocardiographic ','exercise induced' ,'slope','ca','thal','blood sugar']
for column in cat_col:
    print(heartdata[column].value_counts( dropna=False))
    print('-'*50)
```

```
sex
1    419
0    178
Name: count, dtype: int64
--------------------------------------------------
chest pain
4    267
2    156
3    140
1     34
Name: count, dtype: int64
--------------------------------------------------
electrocardiographic 
0.0    386
2.0    154
1.0     56
NaN      1
Name: count, dtype: int64
--------------------------------------------------
exercise induced
0.0    408
1.0    188
NaN      1
Name: count, dtype: int64
--------------------------------------------------
slope
2.0    231
NaN    190
1.0    154
3.0     22
Name: count, dtype: int64
--------------------------------------------------
ca
NaN    294
0.0    179
1.0     65
2.0     38
3.0     20
9.0      1
Name: count, dtype: int64
--------------------------------------------------
thal
NaN    268
3.0    173
7.0    128
6.0     28
Name: count, dtype: int64
--------------------------------------------------
blood sugar
0.0    524
1.0     65
NaN      8
Name: count, dtype: int64
--------------------------------------------------
```

In [125…

```python
for column in cat_col:
    print(heartdata[column].value_counts(normalize=True, dropna=False))
    print('-'*50)
```

```
sex
1    0.701843
0    0.298157
Name: proportion, dtype: float64
--------------------------------------------------
chest pain
4    0.447236
2    0.261307
3    0.234506
1    0.056951
Name: proportion, dtype: float64
--------------------------------------------------
electrocardiographic
0.0    0.646566
2.0    0.257956
1.0    0.093802
NaN    0.001675
Name: proportion, dtype: float64
--------------------------------------------------
exercise induced
0.0    0.683417
1.0    0.314908
NaN    0.001675
Name: proportion, dtype: float64
--------------------------------------------------
slope
2.0    0.386935
NaN    0.318258
1.0    0.257956
3.0    0.036851
Name: proportion, dtype: float64
--------------------------------------------------
ca
NaN    0.492462
0.0    0.299832
1.0    0.108878
2.0    0.063652
3.0    0.033501
9.0    0.001675
Name: proportion, dtype: float64
--------------------------------------------------
thal
NaN    0.448911
3.0    0.289782
7.0    0.214405
6.0    0.046901
Name: proportion, dtype: float64
--------------------------------------------------
blood sugar
0.0    0.877722
1.0    0.108878
NaN    0.013400
Name: proportion, dtype: float64
--------------------------------------------------
```

Observation In 'ca' column value 9.0 is observed which should be 0.0 and has been typed 9.0 by mistake.

In [125…
```
heartdata['ca'][heartdata['ca']==9]=0
```

In [125…
```
heartdata['ca'].value_counts(normalize=True, dropna=False)
```

Out[125…
```
ca
NaN    0.492462
0.0    0.301508
1.0    0.108878
2.0    0.063652
3.0    0.033501
Name: proportion, dtype: float64
```

Missing Values

In [126…
```
heartdata.isnull().sum()
```

```
Out[126… Age (age in year)          0
         sex                        0
         chest pain                 0
         blood pressure             1
         cholestoral               23
         blood sugar                8
         electrocardiographic       1
         heart rate                 1
         exercise induced           1
         depression                 0
         slope                    190
         ca                       294
         thal                     268
         c                          0
         dtype: int64
```

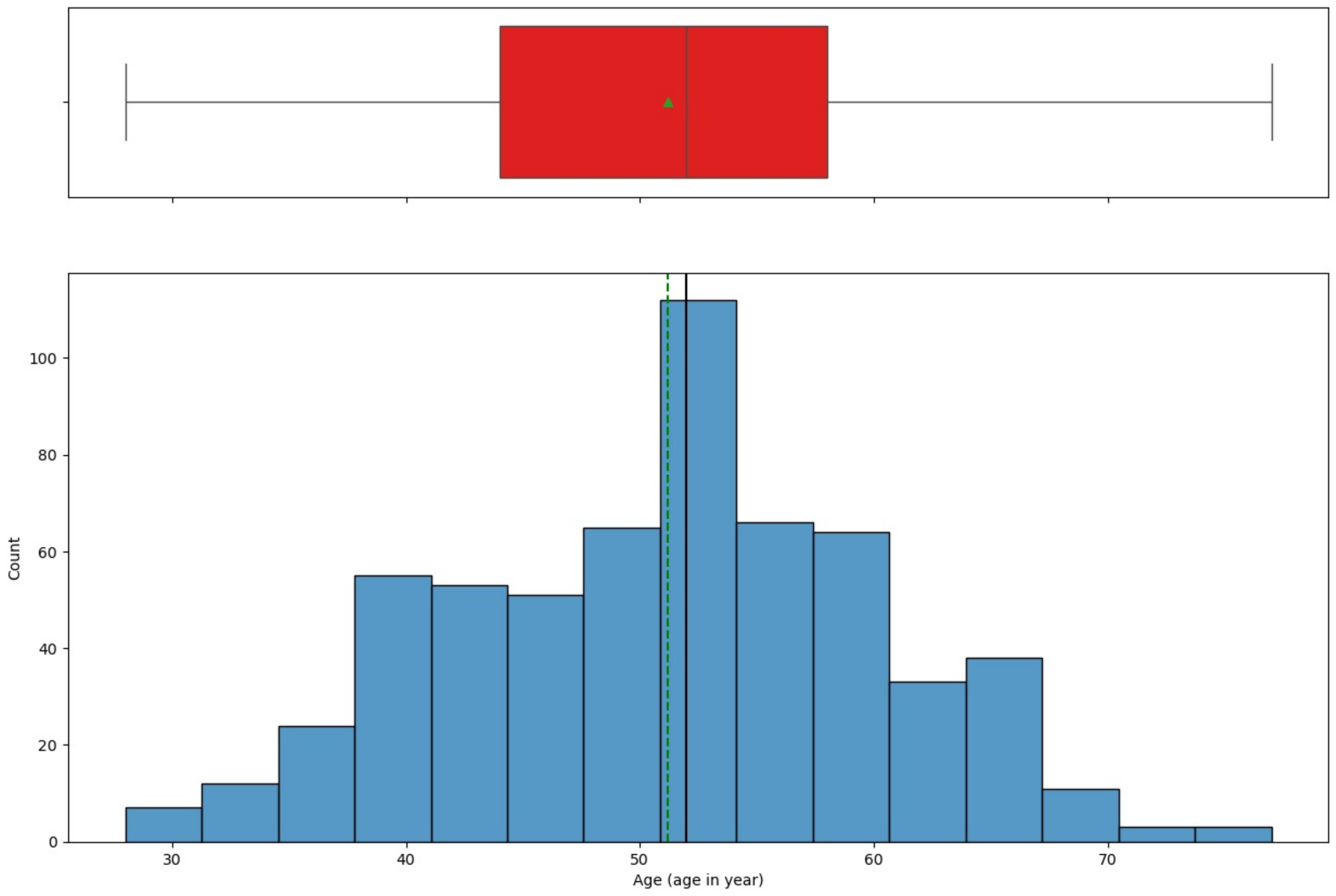
Univariate Observation

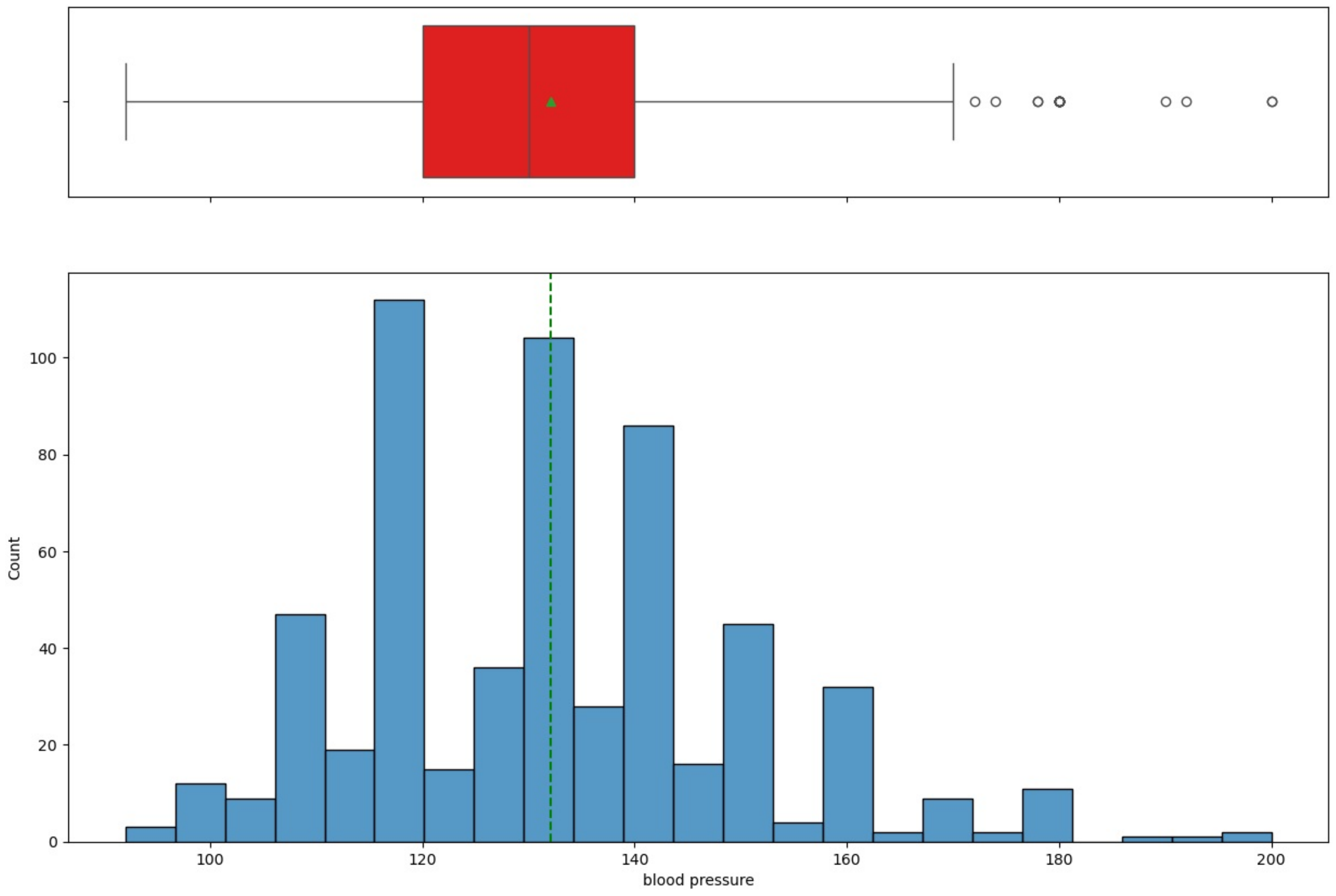
```
In [126… def histogram_boxplot(feature, figsize=(15, 10), bins="auto"):
             """ Boxplot and histogram combined
             feature: 1-d feature array
             figsize: size of fig (default (15, 10))
             bins: number of bins (default "auto")
             """
             f, (ax_box, ax_hist) = plt.subplots(
                 nrows=2,  # Number of rows of the subplot grid
                 sharex=True,  # The X-axis will be shared among all the subplots
                 gridspec_kw={"height_ratios": (.25, .75)},
                 figsize=figsize
             )

             # Creating the subplots
             # Boxplot will be created and the mean value of the column will be indicated using some symbol
             sns.boxplot(x=feature, ax=ax_box, showmeans=True, color='red')

             # For histogram
             sns.histplot(x=feature, kde=False, ax=ax_hist, bins=bins)
             ax_hist.axvline(np.mean(feature), color='g', linestyle='--')      # Add mean to the histogram
             ax_hist.axvline(np.median(feature), color='black', linestyle='-') # Add median to the histogram

             plt.show()
```

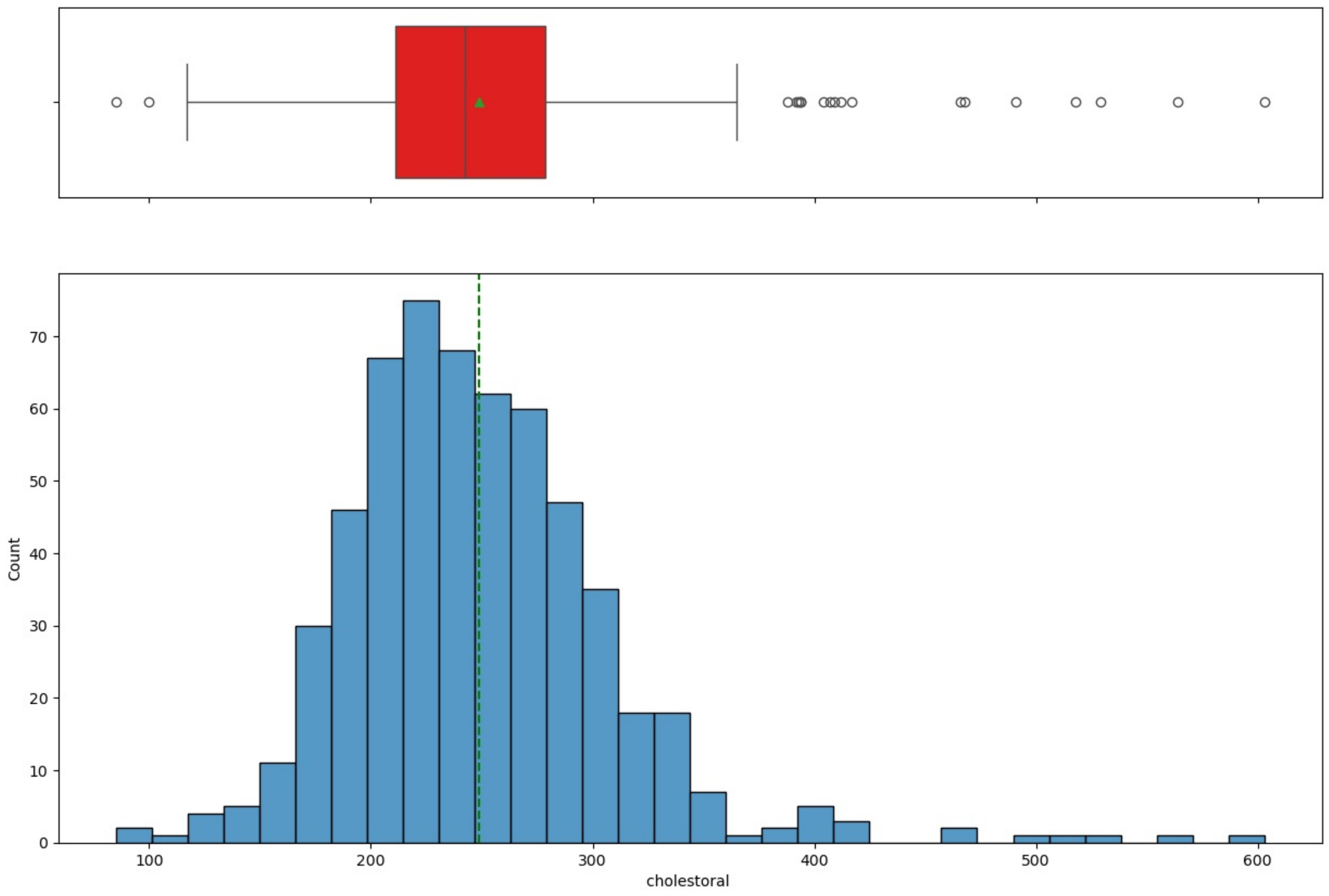
```
In [126… num_col1 =heartdata.drop(columns=cat_col)
         num_col=num_col1.drop('c',axis=1)
         num_col.columns
```

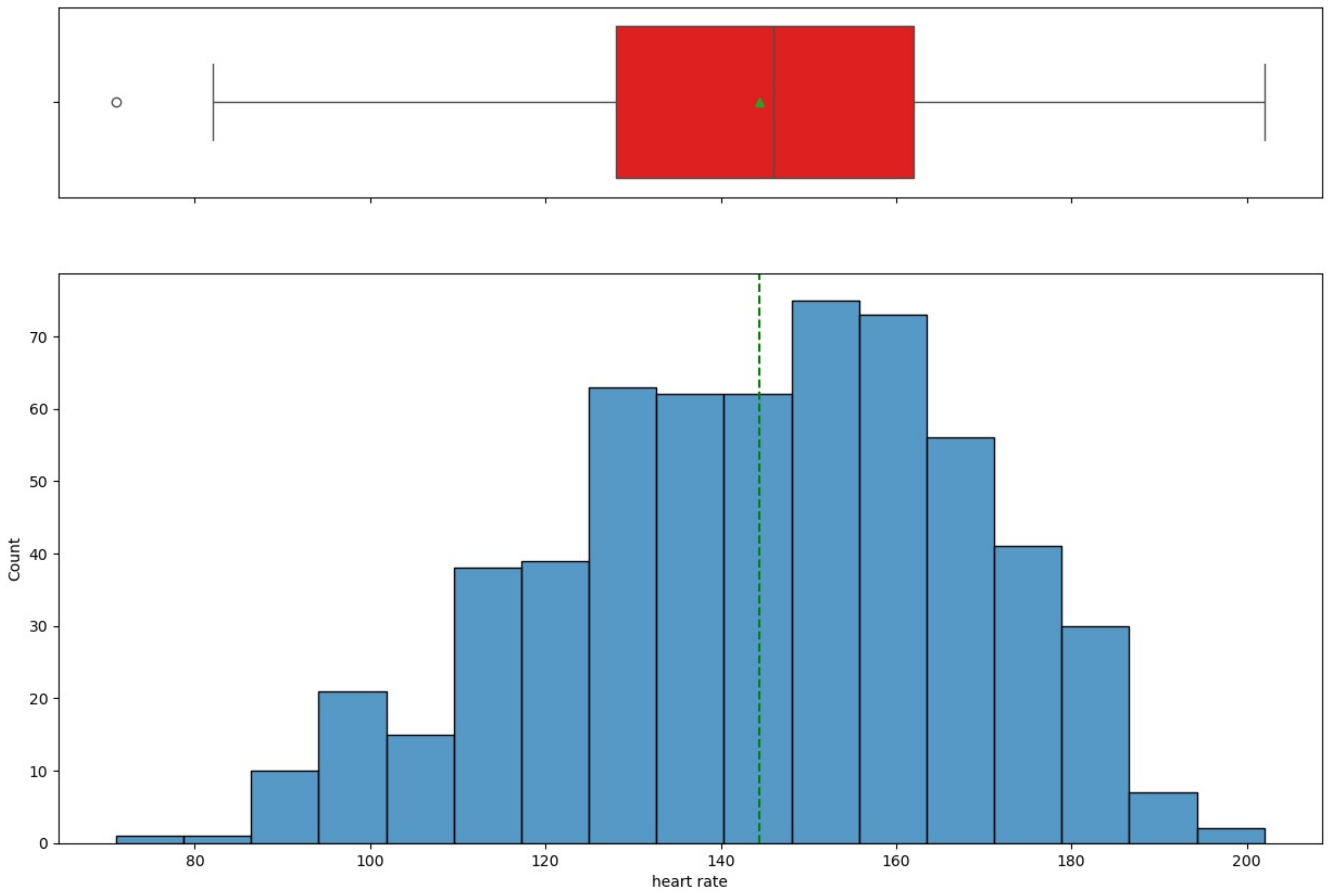
```
Out[126… Index(['Age (age in year)', 'blood pressure', 'cholestoral ', 'heart rate',
                'depression '],
               dtype='object')
```

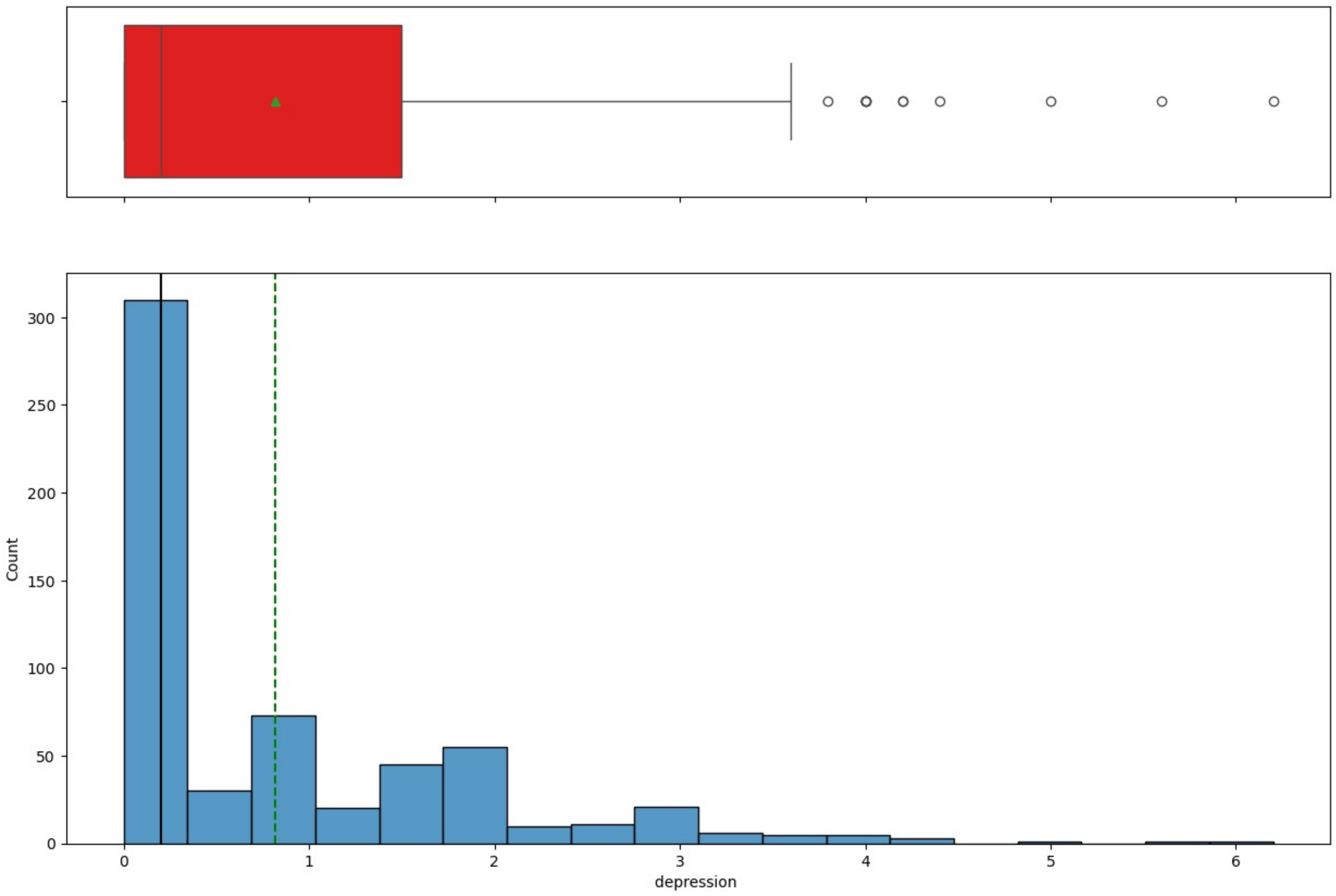
```
In [126… for i in num_col:
             p = histogram_boxplot(heartdata[i])
             plt.show()
```


Count
0
20
40
60
80
100
30
40
50
60
70
Age (age in year)


Count
0
20
40
60
80
100
100
120
140
160
180
200
blood pressure


Count
0
10
20
30
40
50
60
70
100
200
300
400
500
600
cholestoral


Count
0
10
20
30
40
50
60
70
80
100
120
140
160
180
200
heart rate

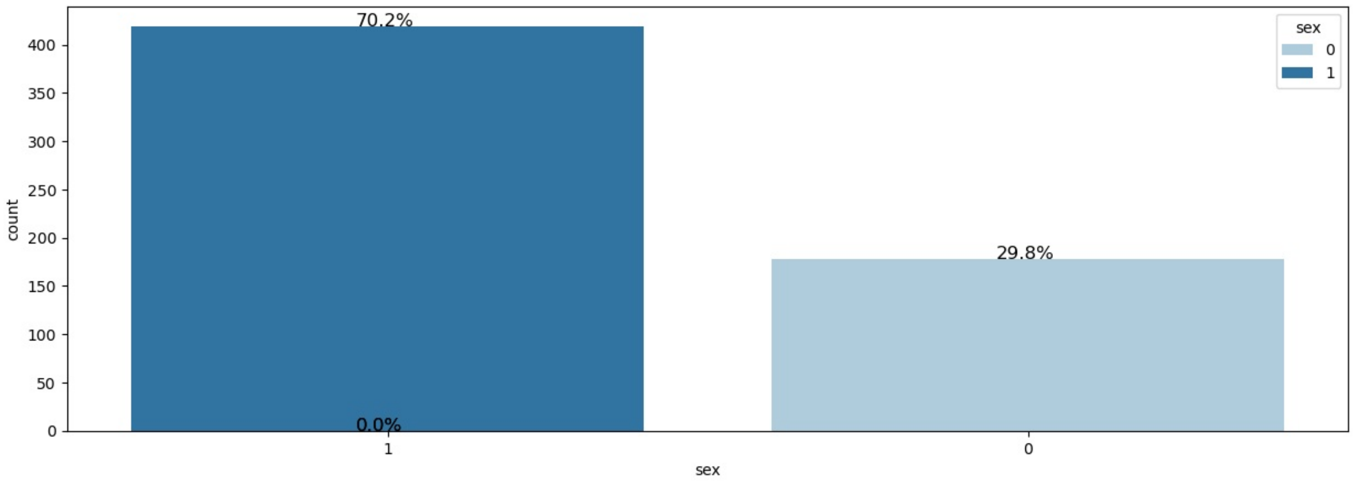


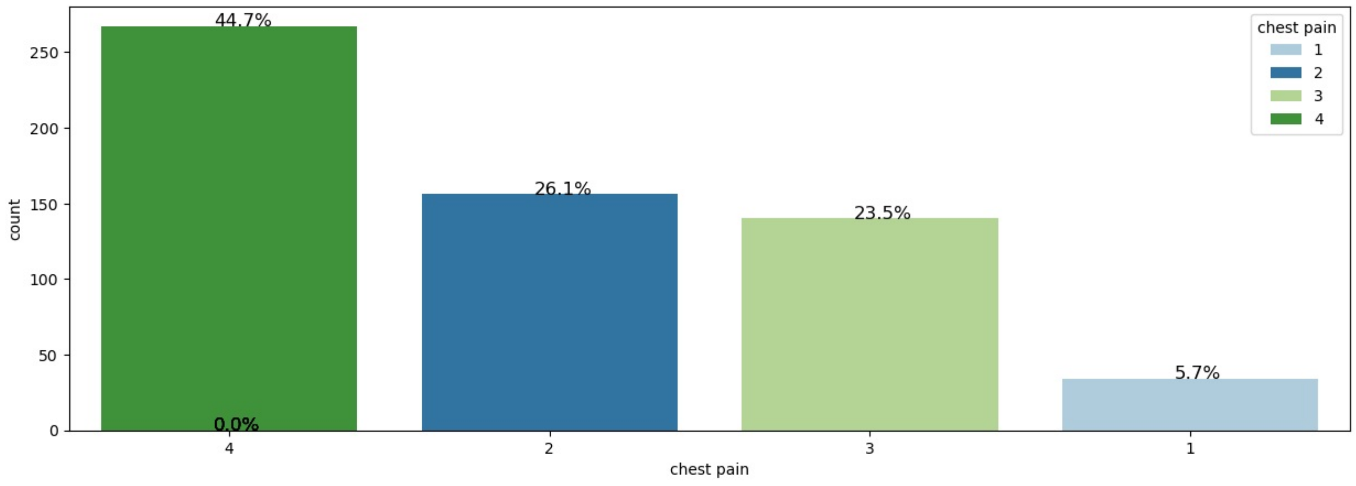
Observation

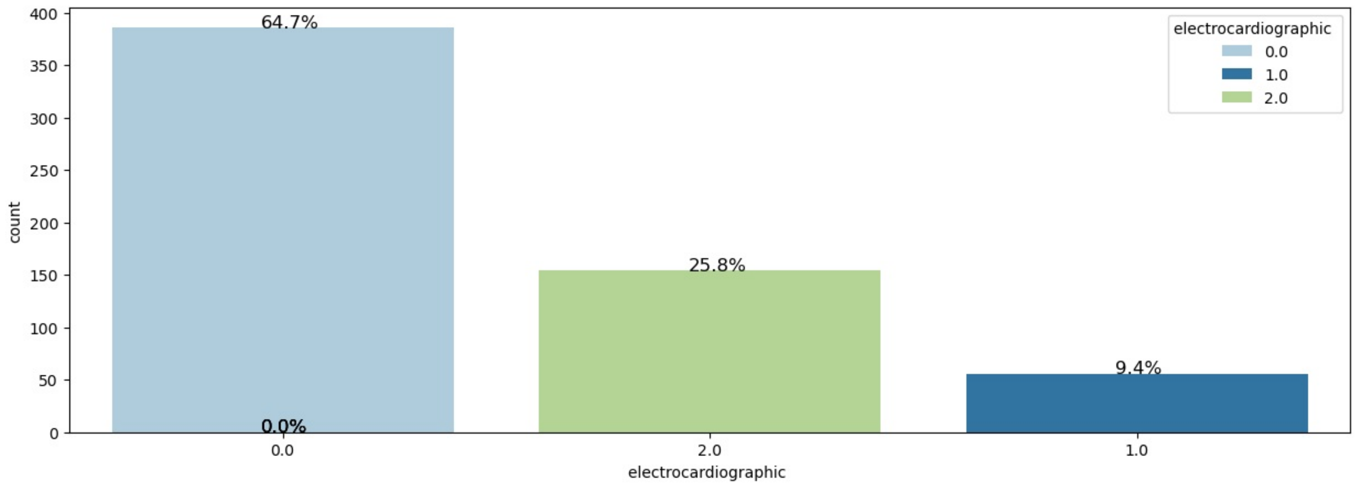
```
In [ ]:
```

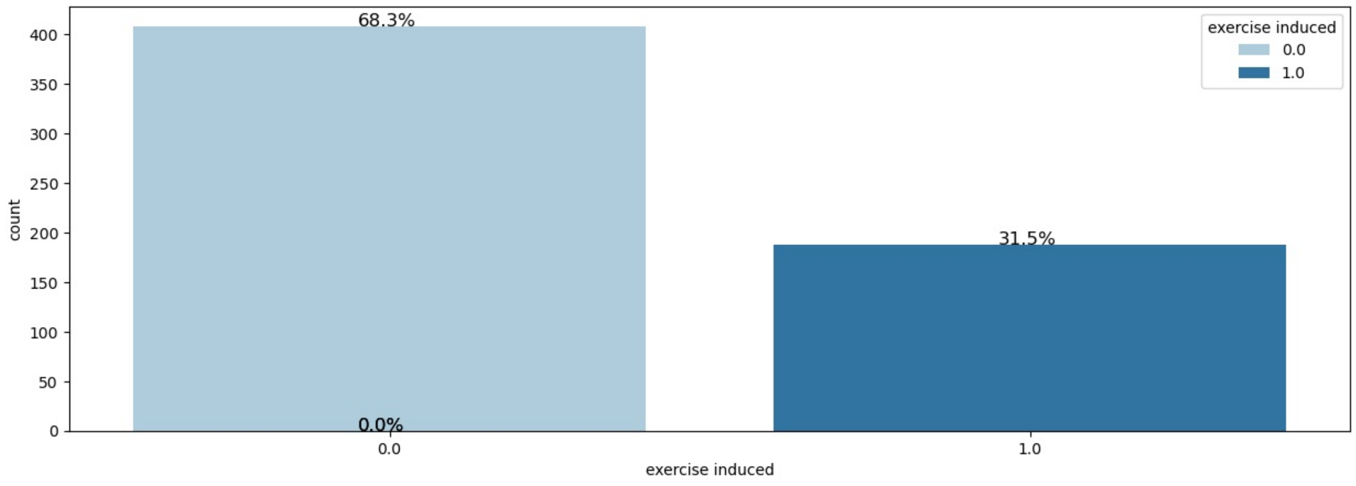
```
In [127…
def bar_perc(data, z):
    total = len(data[z]) # Length of the column
    plt.figure(figsize = (15, 5))

    # Convert the column to a categorical data type
    data[z] = data[z].astype('category')

    ax = sns.countplot(x=z, data=data, hue=z, palette='Paired', order=data[z].value_counts().index)

    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height() / total) # Percentage of each class
        x = p.get_x() + p.get_width() / 2 - 0.05                   # Width of the plot
        y = p.get_y() + p.get_height()                              # Height of the plot
        ax.annotate(percentage, (x, y), size = 12)                  # Annotate the percentage

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

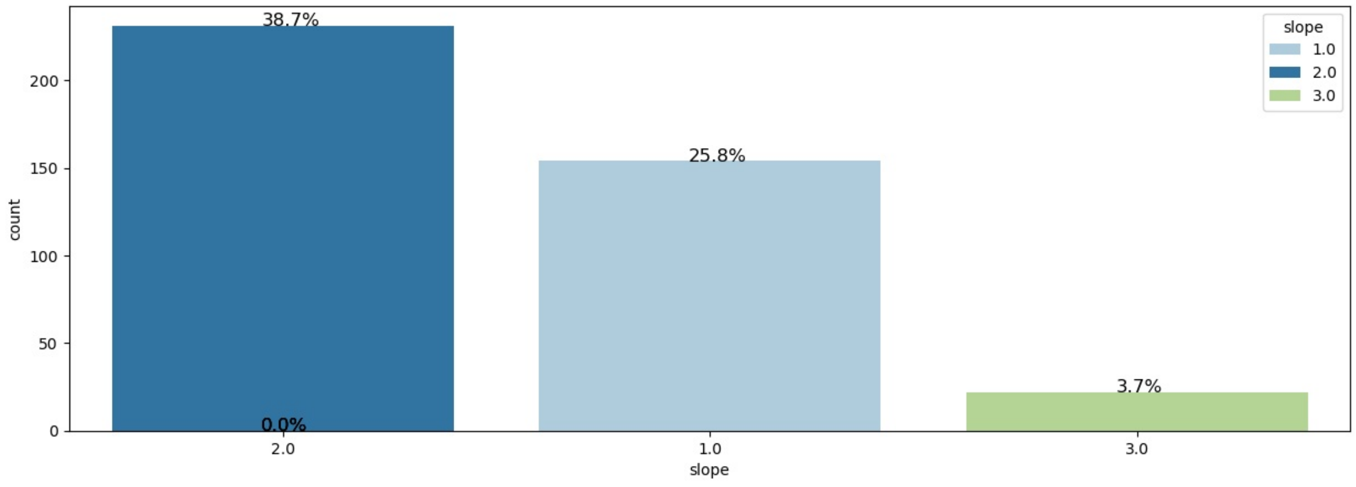
```
In [127…
for i in cat_col:
    p = bar_perc(heartdata,i)
    plt.show()
```

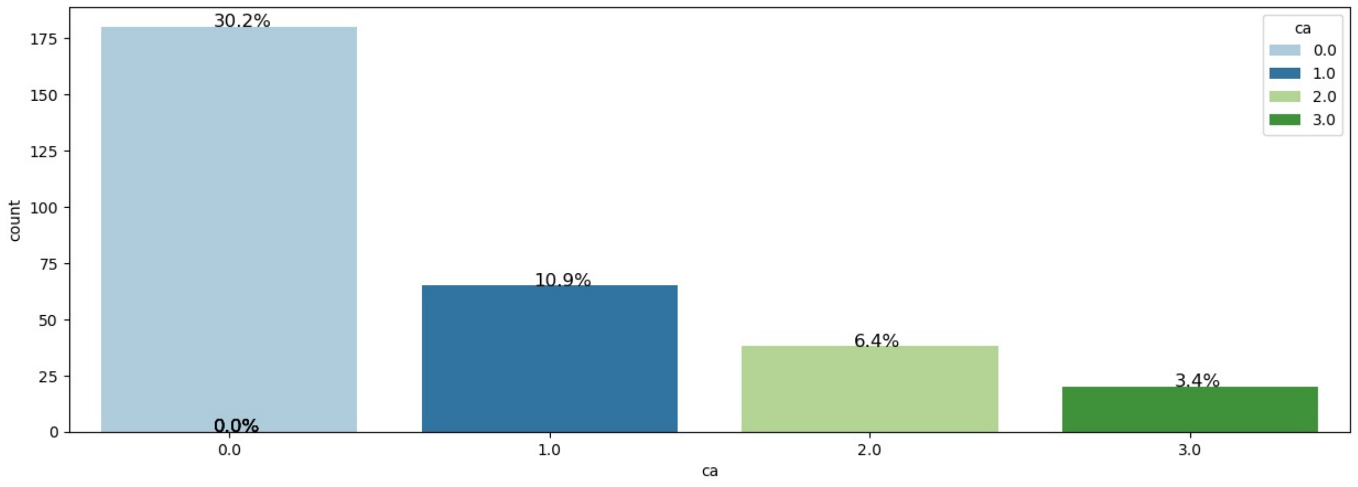


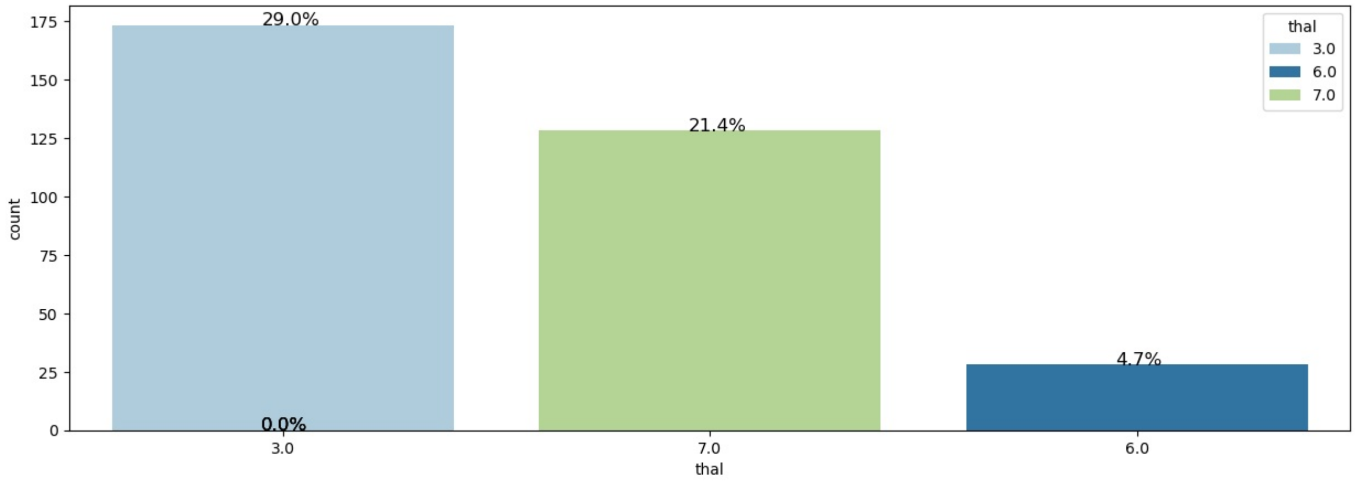

44.7%
26.1%
23.5%
5.7%
0.0%
chest pain
1
2
3
4
count
chest pain

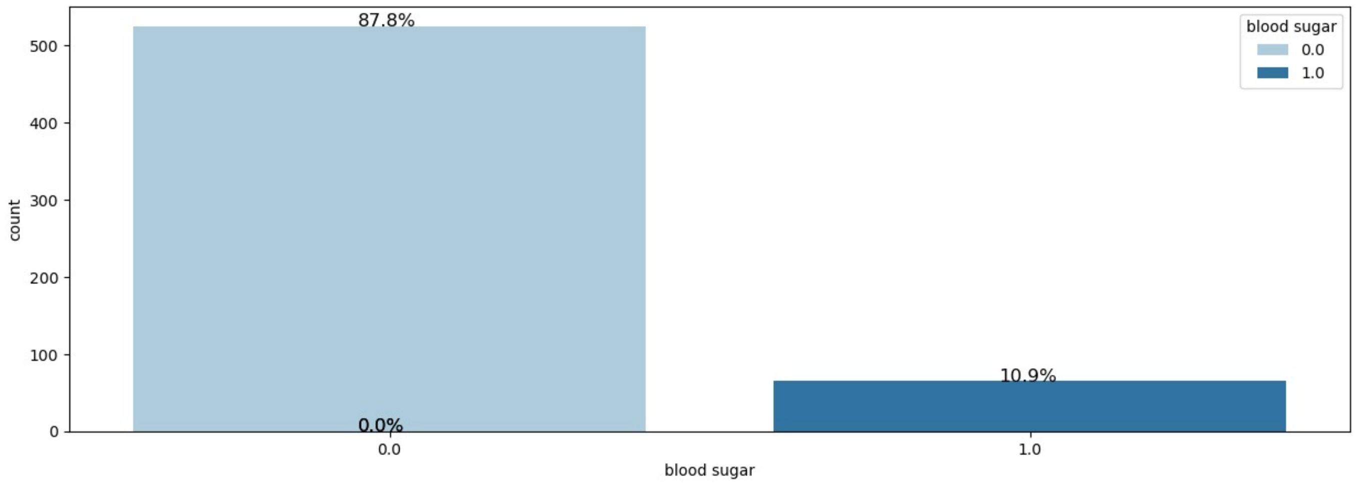

64.7%
25.8%
9.4%
0.0%
electrocardiographic
0.0
1.0
2.0
count
electrocardiographic


68.3%
31.5%
0.0%
exercise induced
0.0
1.0
count
exercise induced


38.7%
25.8%
3.7%
0.0%
slope
1.0
2.0
3.0
count
slope







Multiivariate Observation

In [127…
```
heartdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 597 entries, 0 to 596
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype   
---  ------               --------------  -----   
 0   Age (age in year)    597 non-null    int64   
 1   sex                  597 non-null    category
 2   chest pain           597 non-null    category
 3   blood pressure       596 non-null    float64 
 4   cholestoral          574 non-null    float64 
 5   blood sugar          589 non-null    category
 6   electrocardiographic 596 non-null    category
 7   heart rate           596 non-null    float64 
 8   exercise induced     596 non-null    category
 9   depression           597 non-null    float64 
 10  slope                407 non-null    category
 11  ca                   303 non-null    category
 12  thal                 329 non-null    category
 13  c                    597 non-null    int64   
dtypes: category(8), float64(4), int64(2)
memory usage: 33.9 KB
```

In [127…

```
corr=heartdata.corr()
corr
```

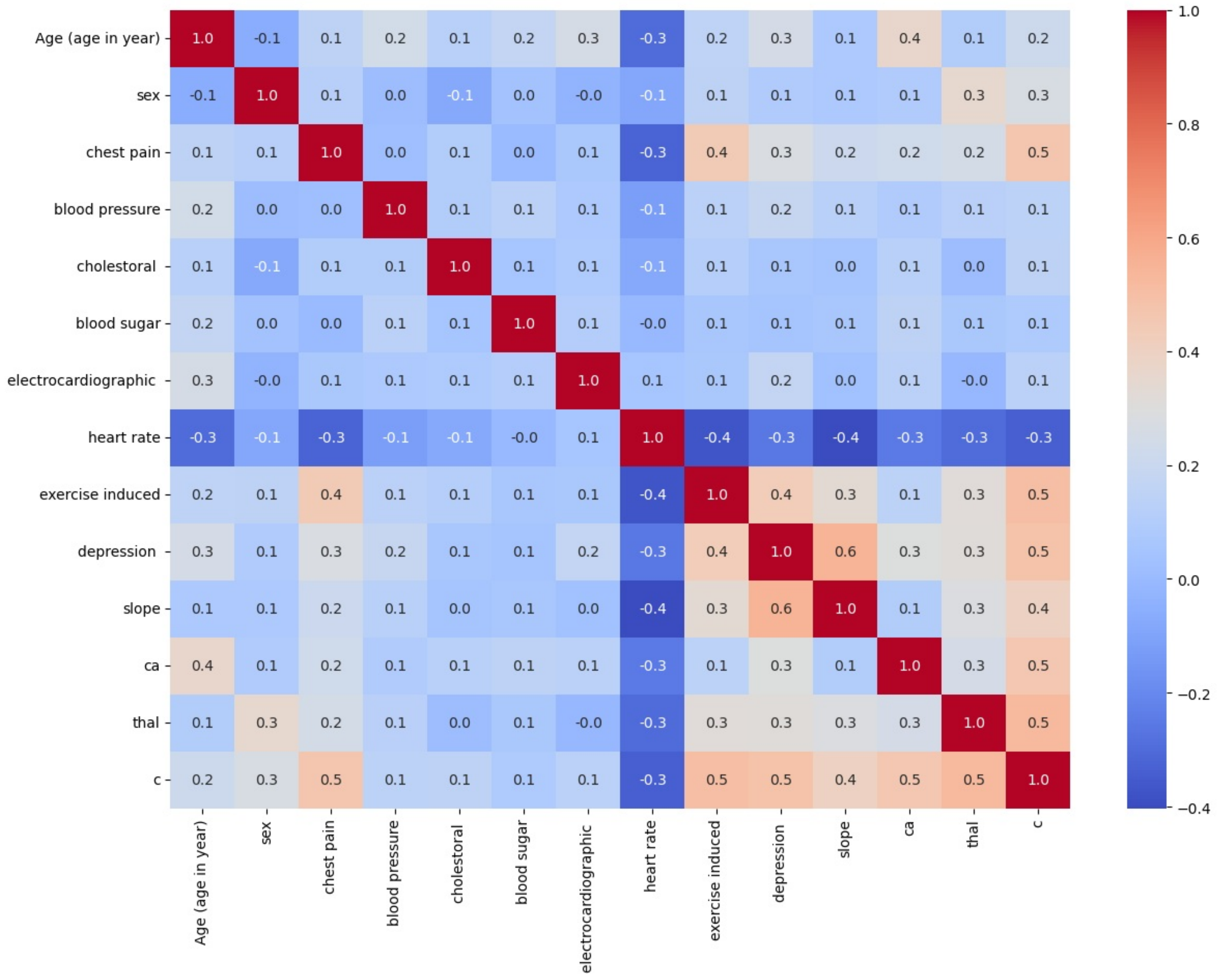
Out[127…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced
Age (age in year)	1.000000	-0.062397	0.147064	0.238490	0.123624	0.176286	0.260132	-0.303596	0.155862
sex	-0.062397	1.000000	0.120748	0.010620	-0.076399	0.038030	-0.034982	-0.088691	0.148814
chest pain	0.147064	0.120748	1.000000	0.021586	0.104111	0.001428	0.073320	-0.322748	0.438328
blood pressure	0.238490	0.010620	0.021586	1.000000	0.105189	0.136097	0.077768	-0.117829	0.136658
cholestoral	0.123624	-0.076399	0.104111	0.105189	1.000000	0.054867	0.088498	-0.076064	0.117111
blood sugar	0.176286	0.038030	0.001428	0.136097	0.054867	1.000000	0.111847	-0.005236	0.063431
electrocardiographic	0.260132	-0.034982	0.073320	0.077768	0.088498	0.111847	1.000000	0.052515	0.071970
heart rate	-0.303596	-0.088691	-0.322748	-0.117829	-0.076064	-0.005236	0.052515	1.000000	-0.374642
exercise induced	0.155862	0.148814	0.438328	0.136658	0.117111	0.063431	0.071970	-0.374642	1.000000
depression	0.253305	0.095716	0.277695	0.185216	0.065998	0.050842	0.175329	-0.259880	0.426849
slope	0.078979	0.075835	0.209141	0.126015	0.047846	0.058897	0.032245	-0.402652	0.332025
ca	0.364036	0.090833	0.227668	0.093548	0.123661	0.148741	0.136486	-0.253548	0.140423
thal	0.105296	0.349134	0.245214	0.133696	0.011964	0.069128	-0.012682	-0.302562	0.320352
c	0.216430	0.268343	0.463527	0.142178	0.145802	0.090071	0.137410	-0.342209	0.504280

In [128…

```
## Check for correlation among numerical variables
#corr = heartdata.corr()

# Plot the mapp
plt.figure(figsize = (14, 10))
sns.heatmap(corr, annot = True, cmap = 'coolwarm', fmt = ".1f")
```

Out[128… <Axes: >



Observation

Chest pain, exercise-induced angina, depression, calcium levels (ca), and thalassemia (thal) all appear to have a correlation with c, with values around 0.5.

Missing value

In [128…
```
heartdata.isnull().sum()
```

Out[128…
```
Age (age in year)          0
sex                        0
chest pain                 0
blood pressure             1
cholestoral               23
blood sugar                8
electrocardiographic       1
heart rate                 1
exercise induced           1
depression                 0
slope                    190
ca                       294
thal                     268
c                          0
dtype: int64
```

Outlier Values

Number of outliers based on IQR and Z-score

In [129…
```
for variable in num_col:
    print(variable)
    Q1=heartdata[variable].quantile(0.25)
    Q3=heartdata[variable].quantile(0.75)
    IQR=Q3-Q1
    zscore=np.abs(stats.zscore(heartdata[variable]))

   # print ('min=',heartdata[variable].min(),'max=',heartdata[variable].max(),'median=',heartdata[variable].med
    print('number of outliers with IQR=',heartdata[(heartdata[variable]<Q1-1.5*IQR)|(heartdata[variable]>Q3+1.5
    print('number of outliers with zscore=',heartdata[zscore>3].shape[0])
    print( zscore[zscore>3])
```

```
Age (age in year)
number of outliers with IQR= 0
number of outliers with zscore= 0
Series([], Name: Age (age in year), dtype: float64)
blood pressure
number of outliers with IQR= 17
number of outliers with zscore= 0
Series([], Name: blood pressure, dtype: float64)
cholestoral
number of outliers with IQR= 19
number of outliers with zscore= 0
Series([], Name: cholestoral , dtype: float64)
heart rate
number of outliers with IQR= 1
number of outliers with zscore= 0
Series([], Name: heart rate, dtype: float64)
depression
number of outliers with IQR= 11
number of outliers with zscore= 6
101    3.171150
514    5.045489
520    4.483187
527    3.171150
581    3.358583
591    3.920885
Name: depression , dtype: float64
```

Fill outliers which are found by Z-score with null values and the rest with upper and lower amount of data.

In [129…
```
def treat_outliers(df, col):
    # Calculate IQR and Z-score
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1 - 1.5 * IQR
    upper_whisker = Q3 + 1.5 * IQR
    zscore = np.abs(stats.zscore(df[col]))

    # Loop through each value and apply conditions
    for idx, value in enumerate(df[col]):
        if (value < lower_whisker or value > upper_whisker) and zscore[idx] > 3:
            # Set to NaN if outlier in both IQR and Z-score
            df.at[idx, col] = np.nan
        elif value < lower_whisker:
            # Clip to lower limit if only an IQR outlier
            df.at[idx, col] = lower_whisker
        elif value > upper_whisker:
            # Clip to upper limit if only an IQR outlier
            df.at[idx, col] = upper_whisker

    return df

# Apply the function to each relevant column
num_cols = ['blood pressure', 'cholestoral ', 'heart rate', 'depression ']
for col in num_cols:
    heartdata = treat_outliers(heartdata, col)
```

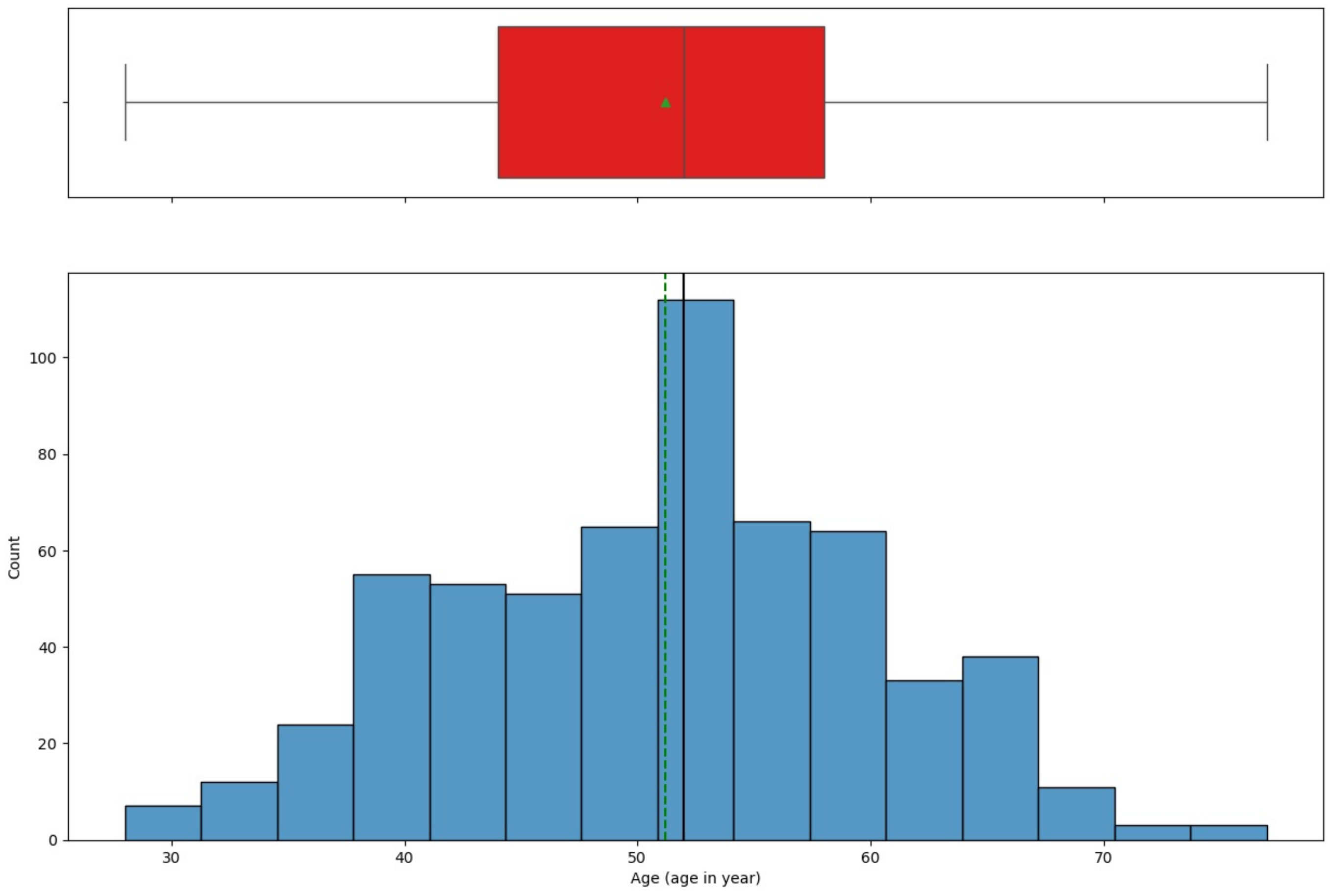
Outlier found based on Z-score are added to null values.

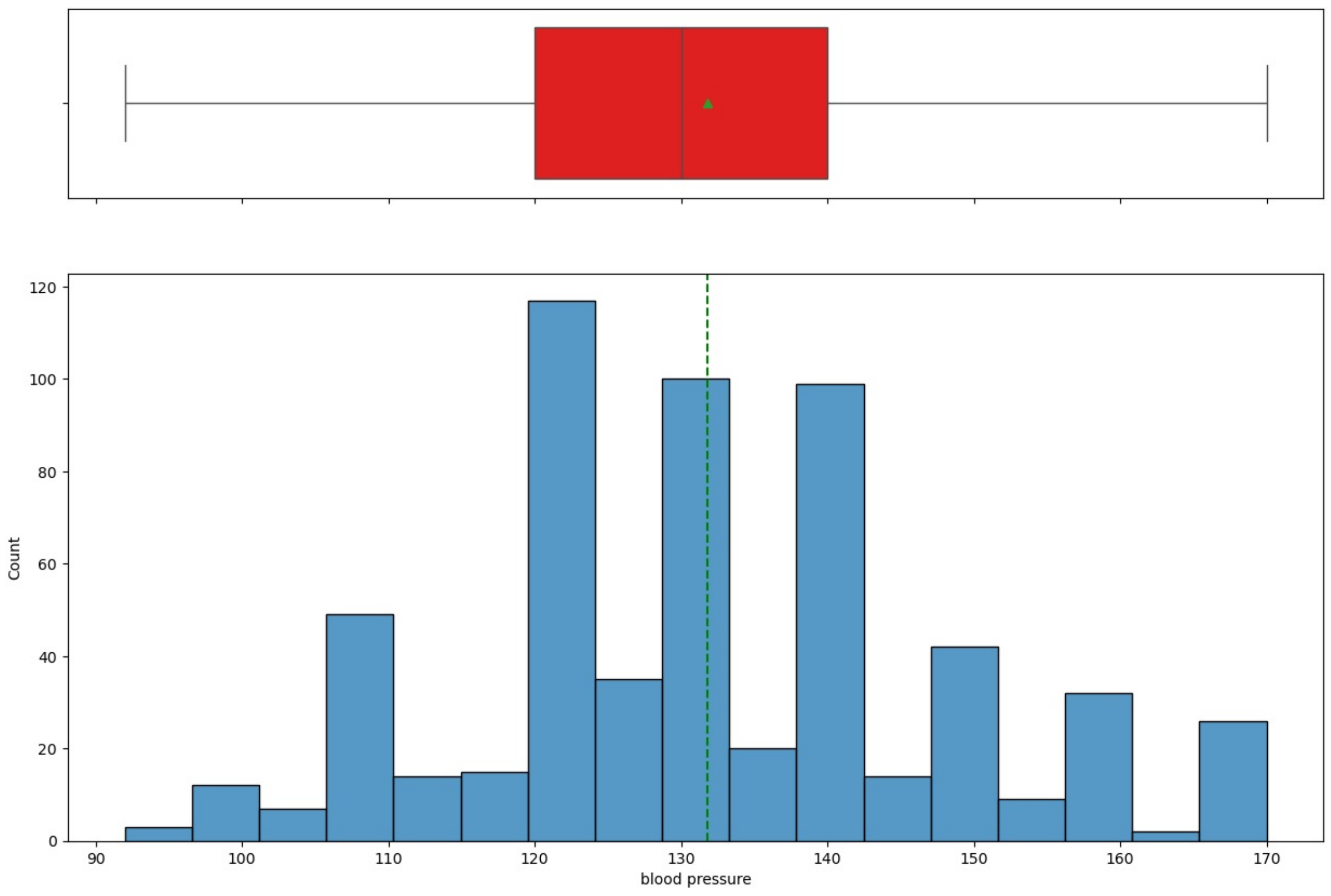
In [129…
```
heartdata.isnull().sum()
```

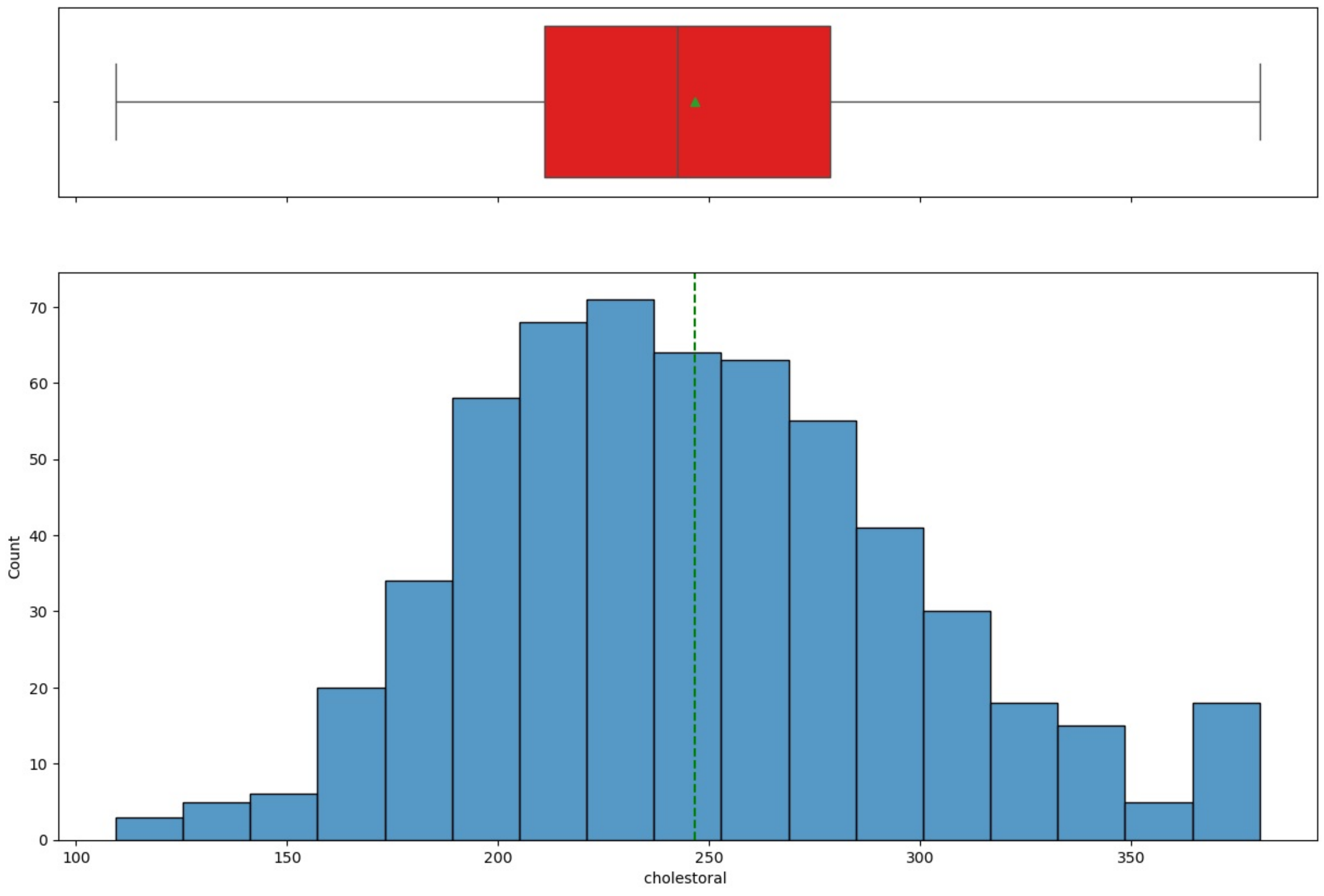
Out[129…
```
Age (age in year)          0
sex                        0
chest pain                 0
blood pressure             1
cholestoral               23
blood sugar                8
electrocardiographic       1
heart rate                 1
exercise induced           1
depression                 6
slope                    190
ca                       294
thal                     268
c                          0
dtype: int64
```

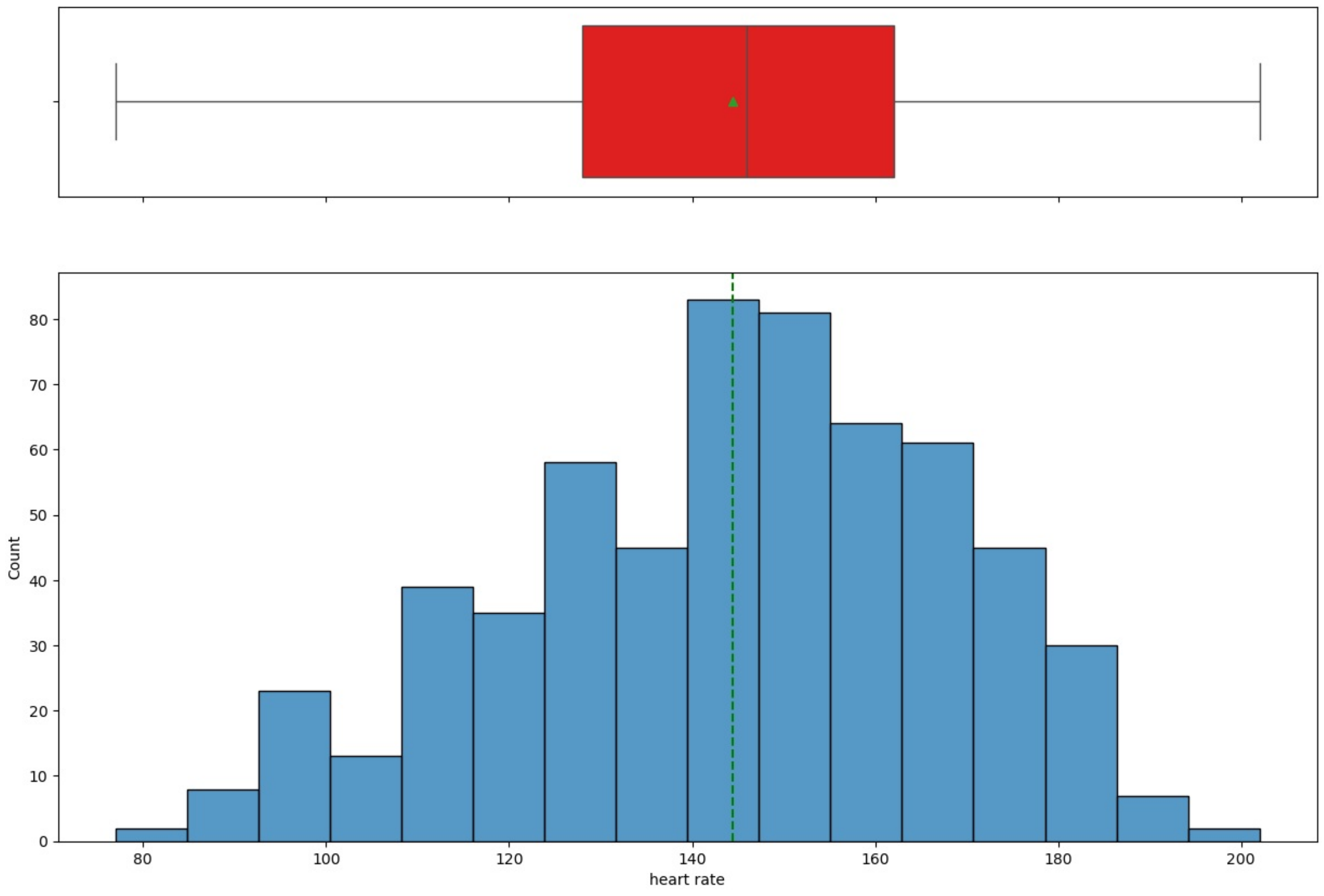
No more outliers are detected in box-plot

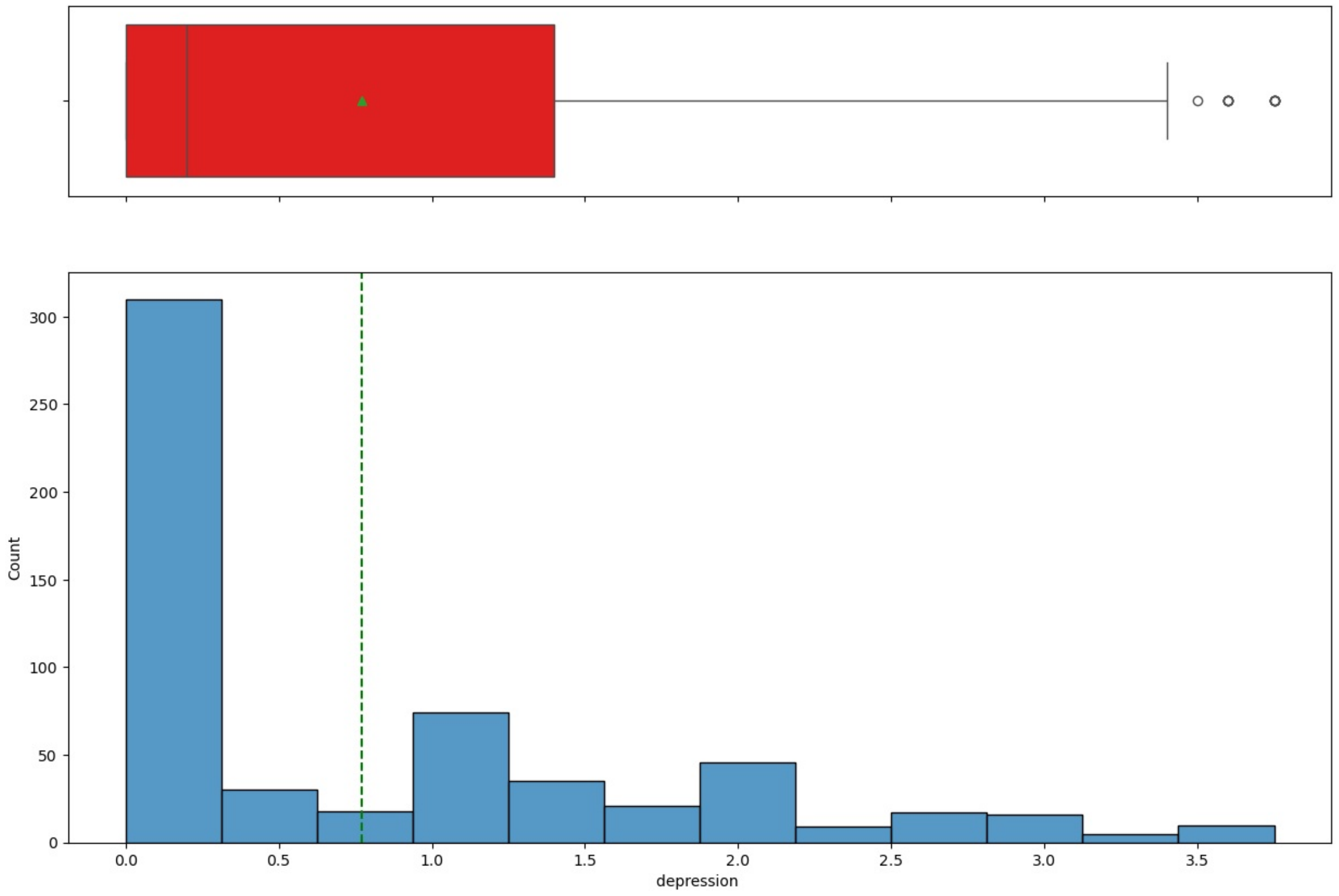
In [130…
```
for i in num_col:
    p = histogram_boxplot(heartdata[i])
    plt.show()
```




Count
120
100
80
60
40
20
0
90
100
110
120
130
140
150
160
170
blood pressure


Count
70
60
50
40
30
20
10
0
100
150
200
250
300
350
cholestoral


Count
0
10
20
30
40
50
60
70
80
80
100
120
140
160
180
200
heart rate


Count
0
50
100
150
200
250
300
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
depression

Missing

In [130…
```
heartdata.isnull().sum()
```

Out[130…
```
Age (age in year)          0
sex                        0
chest pain                 0
blood pressure             1
cholestoral               23
blood sugar                8
electrocardiographic       1
heart rate                 1
exercise induced           1
depression                 6
slope                    190
ca                       294
thal                     268
c                          0
dtype: int64
```

In [130…
```
pd.DataFrame({'Count':heartdata.isnull().sum()[heartdata.isnull().sum()>0], 'Percentage':heartdata.isnull().sum
```

Out[130…

	Count	Percentage
blood pressure	1	0.167504
cholestoral	23	3.852596
blood sugar	8	1.340034
electrocardiographic	1	0.167504
heart rate	1	0.167504
exercise induced	1	0.167504
depression	6	1.005025
slope	190	31.825796
ca	294	49.246231
thal	268	44.891122

Observation

ca and thal columns have the highest percentage (almost 50%) of missing values.

blood pressure, electrocardiographic, heart rate and exercise induced have the lowest percentage of missing values.

There is no column with more than 50% of missing values.

Finding rows with more than 4 null features to drop

In [131…
```
heartdata.columns
```

Out[131…
```
Index(['Age (age in year)', 'sex', 'chest pain', 'blood pressure',
       'cholestoral ', 'blood sugar', 'electrocardiographic ', 'heart rate',
       'exercise induced', 'depression ', 'slope', 'ca', 'thal', 'c'],
      dtype='object')
```

In [131…
```
heartdata.loc[heartdata['blood pressure'].isnull()==True]
```

Out[131…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
347	48	0	2	NaN	308.0	0.0	1.0	NaN	NaN	2.0	1.0	NaN	NaN	0

In [131…
```
heartdata.loc[heartdata['electrocardiographic '].isnull()==True]
```

Out[131…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
562	55	1	1	140.0	295.0	0.0	NaN	136.0	0.0	0.0	NaN	NaN	NaN	1

In [131…
```
heartdata.loc[heartdata['heart rate'].isnull()==True]
```

Out[131…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
347	48	0	2	NaN	308.0	0.0	1.0	NaN	NaN	2.0	1.0	NaN	NaN	0

In [131…

```
heartdata.loc[heartdata['exercise induced'].isnull()==True]
```

Out[131…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
347	48	0	2	NaN	308.0	0.0	1.0	NaN	NaN	2.0	1.0	NaN	NaN	0

In [132…

```
heartdata.loc[heartdata['blood sugar'].isnull()==True]
```

Out[132…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
183	53	0	2	113.0	380.375	NaN	0.0	127.0	0.0	0.0	NaN	NaN	NaN	0
213	46	1	3	150.0	163.000	NaN	0.0	116.0	0.0	0.0	NaN	NaN	NaN	0
289	49	1	4	120.0	297.000	NaN	0.0	132.0	0.0	1.0	2.0	NaN	NaN	0
301	56	0	3	130.0	219.000	NaN	1.0	164.0	0.0	0.0	NaN	NaN	7.0	0
314	38	0	2	120.0	275.000	NaN	0.0	129.0	0.0	0.0	NaN	NaN	NaN	0
316	54	0	2	140.0	309.000	NaN	1.0	140.0	0.0	0.0	NaN	NaN	NaN	0
421	40	1	4	120.0	380.375	NaN	0.0	152.0	1.0	1.0	2.0	NaN	6.0	1
431	41	1	4	120.0	237.000	NaN	0.0	138.0	1.0	1.0	2.0	NaN	NaN	1

In [132…

```
heartdata.loc[heartdata['cholestoral '].isnull()==True].head()
```

Out[132…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
166	54	1	3	150.0	NaN	0.0	0.0	122.0	0.0	0.0	NaN	NaN	NaN	0
185	56	1	3	130.0	NaN	0.0	0.0	114.0	0.0	0.0	NaN	NaN	NaN	0
189	48	0	2	120.0	NaN	1.0	1.0	148.0	0.0	0.0	NaN	NaN	NaN	0
204	43	0	3	150.0	NaN	0.0	0.0	175.0	0.0	0.0	NaN	NaN	3.0	0
221	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0

Blood pressure, heart rate and exercise induced's only missing value are from the same index No. 347, where Thal and Ca is also null. That is why it makes sense to delete the data. There is one line with 5 missing values.

In [132…

```
heartdata = heartdata.drop(347).reset_index(drop=True)
```

line 347 removed

In [132…

```
pd.DataFrame({'Count':heartdata.isnull().sum()[heartdata.isnull().sum()>0], 'Percentage':heartdata.isnull().sum
```

Out[132…

	Count	Percentage
cholestoral	23	3.859060
blood sugar	8	1.342282
electrocardiographic	1	0.167785
depression	6	1.006711
slope	190	31.879195
ca	293	49.161074
thal	267	44.798658

Exercise doesnt have null value any more.

Check if null values from thal,ca and slope are from same index:

In [133…
```
heartdata.loc[heartdata['thal'].isnull()==True,'ca'].value_counts(normalize=True,dropna=False)
```

Out[133…
```
ca
NaN     0.981273
 0.0    0.018727
 1.0    0.000000
 2.0    0.000000
 3.0    0.000000
Name: proportion, dtype: float64
```

In [133…
```
heartdata.loc[heartdata['slope'].isnull()==True,'ca'].value_counts(normalize=True, dropna=False)
```

Out[133…
```
ca
NaN     0.989474
 0.0    0.010526
 1.0    0.000000
 2.0    0.000000
 3.0    0.000000
Name: proportion, dtype: float64
```

In [133…
```
heartdata.loc[heartdata['slope'].isnull()==True,'thal'].value_counts(normalize=True,dropna=False)
```

Out[133…
```
thal
NaN     0.910526
 7.0    0.036842
 6.0    0.031579
 3.0    0.021053
Name: proportion, dtype: float64
```

In [133…
```
heartdata.loc[heartdata['cholestoral '].isnull()==True,'ca'].value_counts(normalize=True, dropna=False)
```

Out[133…
```
ca
NaN     0.956522
 0.0    0.043478
 1.0    0.000000
 2.0    0.000000
 3.0    0.000000
Name: proportion, dtype: float64
```

There seems to be a strong pattern in missing values, as wherever the ca column has missing data the thal, cholestral and slope columns also have missing values.

In [134…
```
nullcholestral=heartdata.loc[heartdata['cholestoral '].isnull()==True]
nullcholestral
```

Out[134…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
166	54	1	3	150.0	NaN	0.0	0.0	122.0	0.0	0.0	NaN	NaN	NaN	0
185	56	1	3	130.0	NaN	0.0	0.0	114.0	0.0	0.0	NaN	NaN	NaN	0
189	48	0	2	120.0	NaN	1.0	1.0	148.0	0.0	0.0	NaN	NaN	NaN	0
204	43	0	3	150.0	NaN	0.0	0.0	175.0	0.0	0.0	NaN	NaN	3.0	0
221	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
251	39	1	2	120.0	NaN	0.0	1.0	146.0	0.0	2.0	1.0	NaN	NaN	0
257	39	1	2	130.0	NaN	0.0	0.0	120.0	0.0	0.0	NaN	NaN	NaN	0
258	48	1	2	100.0	NaN	0.0	0.0	100.0	0.0	0.0	NaN	NaN	NaN	0
262	49	1	4	140.0	NaN	0.0	0.0	130.0	0.0	0.0	NaN	NaN	NaN	0
265	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
267	52	0	2	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	NaN	NaN	0
272	40	1	3	140.0	NaN	0.0	0.0	188.0	0.0	0.0	NaN	NaN	NaN	0
286	45	1	3	135.0	NaN	0.0	0.0	110.0	0.0	0.0	NaN	NaN	NaN	0
305	29	1	2	140.0	NaN	0.0	0.0	170.0	0.0	0.0	NaN	NaN	NaN	0
332	47	0	3	130.0	NaN	0.0	0.0	145.0	0.0	2.0	2.0	NaN	NaN	0
336	45	0	2	170.0	NaN	0.0	0.0	180.0	0.0	0.0	NaN	NaN	NaN	0
345	59	1	4	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	0.0	NaN	0
346	53	1	2	120.0	NaN	0.0	0.0	132.0	0.0	0.0	NaN	NaN	NaN	0
489	54	1	4	140.0	NaN	0.0	0.0	118.0	1.0	0.0	NaN	NaN	NaN	1
491	38	1	4	110.0	NaN	0.0	0.0	150.0	1.0	1.0	2.0	NaN	NaN	1
504	52	1	4	170.0	NaN	0.0	0.0	126.0	1.0	1.5	2.0	NaN	NaN	1
549	66	1	4	140.0	NaN	0.0	0.0	94.0	1.0	1.0	2.0	NaN	NaN	1
565	59	1	4	130.0	NaN	0.0	0.0	125.0	0.0	0.0	NaN	NaN	NaN	1

In [134…

```
nullcholestralslope=nullcholestral.loc[nullcholestral['slope'].isnull()==True]
nullcholestralslope
```

Out[134…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
166	54	1	3	150.0	NaN	0.0	0.0	122.0	0.0	0.0	NaN	NaN	NaN	0
185	56	1	3	130.0	NaN	0.0	0.0	114.0	0.0	0.0	NaN	NaN	NaN	0
189	48	0	2	120.0	NaN	1.0	1.0	148.0	0.0	0.0	NaN	NaN	NaN	0
204	43	0	3	150.0	NaN	0.0	0.0	175.0	0.0	0.0	NaN	NaN	3.0	0
221	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
257	39	1	2	130.0	NaN	0.0	0.0	120.0	0.0	0.0	NaN	NaN	NaN	0
258	48	1	2	100.0	NaN	0.0	0.0	100.0	0.0	0.0	NaN	NaN	NaN	0
262	49	1	4	140.0	NaN	0.0	0.0	130.0	0.0	0.0	NaN	NaN	NaN	0
265	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
267	52	0	2	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	NaN	NaN	0
272	40	1	3	140.0	NaN	0.0	0.0	188.0	0.0	0.0	NaN	NaN	NaN	0
286	45	1	3	135.0	NaN	0.0	0.0	110.0	0.0	0.0	NaN	NaN	NaN	0
305	29	1	2	140.0	NaN	0.0	0.0	170.0	0.0	0.0	NaN	NaN	NaN	0
336	45	0	2	170.0	NaN	0.0	0.0	180.0	0.0	0.0	NaN	NaN	NaN	0
345	59	1	4	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	0.0	NaN	0
346	53	1	2	120.0	NaN	0.0	0.0	132.0	0.0	0.0	NaN	NaN	NaN	0
489	54	1	4	140.0	NaN	0.0	0.0	118.0	1.0	0.0	NaN	NaN	NaN	1
565	59	1	4	130.0	NaN	0.0	0.0	125.0	0.0	0.0	NaN	NaN	NaN	1

In [134…

```
nullcholestralslopeca=nullcholestralslope.loc[nullcholestralslope['ca'].isnull()==True]
```

```
nullcholestralslopeca
```

Out[134…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
166	54	1	3	150.0	NaN	0.0	0.0	122.0	0.0	0.0	NaN	NaN	NaN	0
185	56	1	3	130.0	NaN	0.0	0.0	114.0	0.0	0.0	NaN	NaN	NaN	0
189	48	0	2	120.0	NaN	1.0	1.0	148.0	0.0	0.0	NaN	NaN	NaN	0
204	43	0	3	150.0	NaN	0.0	0.0	175.0	0.0	0.0	NaN	NaN	3.0	0
221	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
257	39	1	2	130.0	NaN	0.0	0.0	120.0	0.0	0.0	NaN	NaN	NaN	0
258	48	1	2	100.0	NaN	0.0	0.0	100.0	0.0	0.0	NaN	NaN	NaN	0
262	49	1	4	140.0	NaN	0.0	0.0	130.0	0.0	0.0	NaN	NaN	NaN	0
265	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
267	52	0	2	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	NaN	NaN	0
272	40	1	3	140.0	NaN	0.0	0.0	188.0	0.0	0.0	NaN	NaN	NaN	0
286	45	1	3	135.0	NaN	0.0	0.0	110.0	0.0	0.0	NaN	NaN	NaN	0
305	29	1	2	140.0	NaN	0.0	0.0	170.0	0.0	0.0	NaN	NaN	NaN	0
336	45	0	2	170.0	NaN	0.0	0.0	180.0	0.0	0.0	NaN	NaN	NaN	0
346	53	1	2	120.0	NaN	0.0	0.0	132.0	0.0	0.0	NaN	NaN	NaN	0
489	54	1	4	140.0	NaN	0.0	0.0	118.0	1.0	0.0	NaN	NaN	NaN	1
565	59	1	4	130.0	NaN	0.0	0.0	125.0	0.0	0.0	NaN	NaN	NaN	1

In [134…

```
nullcholestralslopecathal= nullcholestralslopeca.loc[nullcholestralslopeca['thal'].isnull()==True]
nullcholestralslopecathal
```

Out[134…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
166	54	1	3	150.0	NaN	0.0	0.0	122.0	0.0	0.0	NaN	NaN	NaN	0
185	56	1	3	130.0	NaN	0.0	0.0	114.0	0.0	0.0	NaN	NaN	NaN	0
189	48	0	2	120.0	NaN	1.0	1.0	148.0	0.0	0.0	NaN	NaN	NaN	0
221	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
257	39	1	2	130.0	NaN	0.0	0.0	120.0	0.0	0.0	NaN	NaN	NaN	0
258	48	1	2	100.0	NaN	0.0	0.0	100.0	0.0	0.0	NaN	NaN	NaN	0
262	49	1	4	140.0	NaN	0.0	0.0	130.0	0.0	0.0	NaN	NaN	NaN	0
265	49	0	2	110.0	NaN	0.0	0.0	160.0	0.0	0.0	NaN	NaN	NaN	0
267	52	0	2	140.0	NaN	0.0	0.0	140.0	0.0	0.0	NaN	NaN	NaN	0
272	40	1	3	140.0	NaN	0.0	0.0	188.0	0.0	0.0	NaN	NaN	NaN	0
286	45	1	3	135.0	NaN	0.0	0.0	110.0	0.0	0.0	NaN	NaN	NaN	0
305	29	1	2	140.0	NaN	0.0	0.0	170.0	0.0	0.0	NaN	NaN	NaN	0
336	45	0	2	170.0	NaN	0.0	0.0	180.0	0.0	0.0	NaN	NaN	NaN	0
346	53	1	2	120.0	NaN	0.0	0.0	132.0	0.0	0.0	NaN	NaN	NaN	0
489	54	1	4	140.0	NaN	0.0	0.0	118.0	1.0	0.0	NaN	NaN	NaN	1
565	59	1	4	130.0	NaN	0.0	0.0	125.0	0.0	0.0	NaN	NaN	NaN	1

In [134…

```
nullcholestralslopecathal.index
```

Out[134…

```
Index([166, 185, 189, 221, 257, 258, 262, 265, 267, 272, 286, 305, 336, 346,
       489, 565],
      dtype='int64')
```

In above indexes all four cholestoral, thal,ca and slope are null. These 16 indexes were chosen to be deleted from data

Totally 17 indexes are dropped.

In [135…

```
heartdata = heartdata.drop(nullcholestralslopecathal.index).reset_index(drop=True)
```

In [135…
```
pd.DataFrame({'Count':heartdata.isnull().sum()[heartdata.isnull().sum()>0], 'Percentage':heartdata.isnull().sum
```
Out[135…

	Count	Percentage
cholestoral	7	1.206897
blood sugar	8	1.379310
electrocardiographic	1	0.172414
depression	6	1.034483
slope	174	30.000000
ca	277	47.758621
thal	251	43.275862

In [135…
```
corr
```
Out[135…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced
Age (age in year)	1.000000	-0.062397	0.147064	0.238490	0.123624	0.176286	0.260132	-0.303596	0.155862
sex	-0.062397	1.000000	0.120748	0.010620	-0.076399	0.038030	-0.034982	-0.088691	0.148814
chest pain	0.147064	0.120748	1.000000	0.021586	0.104111	0.001428	0.073320	-0.322748	0.438328
blood pressure	0.238490	0.010620	0.021586	1.000000	0.105189	0.136097	0.077768	-0.117829	0.136658
cholestoral	0.123624	-0.076399	0.104111	0.105189	1.000000	0.054867	0.088498	-0.076064	0.117111
blood sugar	0.176286	0.038030	0.001428	0.136097	0.054867	1.000000	0.111847	-0.005236	0.063431
electrocardiographic	0.260132	-0.034982	0.073320	0.077768	0.088498	0.111847	1.000000	0.052515	0.071970
heart rate	-0.303596	-0.088691	-0.322748	-0.117829	-0.076064	-0.005236	0.052515	1.000000	-0.374642
exercise induced	0.155862	0.148814	0.438328	0.136658	0.117111	0.063431	0.071970	-0.374642	1.000000
depression	0.253305	0.095716	0.277695	0.185216	0.065998	0.050842	0.175329	-0.259880	0.426849
slope	0.078979	0.075835	0.209141	0.126015	0.047846	0.058897	0.032245	-0.402652	0.332025
ca	0.364036	0.090833	0.227668	0.093548	0.123661	0.148741	0.136486	-0.253548	0.140423
thal	0.105296	0.349134	0.245214	0.133696	0.011964	0.069128	-0.012682	-0.302562	0.320352
c	0.216430	0.268343	0.463527	0.142178	0.145802	0.090071	0.137410	-0.342209	0.504280

In [135…
```
heartdata.columns
```
Out[135…
```
Index(['Age (age in year)', 'sex', 'chest pain', 'blood pressure',
       'cholestoral ', 'blood sugar', 'electrocardiographic ', 'heart rate',
       'exercise induced', 'depression ', 'slope', 'ca', 'thal', 'c'],
      dtype='object')
```
In [136…
```
heartdata.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 580 entries, 0 to 579
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age (age in year)      580 non-null    int64
 1   sex                    580 non-null    category
 2   chest pain             580 non-null    category
 3   blood pressure         580 non-null    float64
 4   cholestoral            573 non-null    float64
 5   blood sugar            572 non-null    category
 6   electrocardiographic   579 non-null    category
 7   heart rate             580 non-null    float64
 8   exercise induced       580 non-null    category
 9   depression             574 non-null    float64
 10  slope                  406 non-null    category
 11  ca                     303 non-null    category
 12  thal                   329 non-null    category
 13  c                      580 non-null    int64
dtypes: category(8), float64(4), int64(2)
memory usage: 33.0 KB
```

Determine the mode for categorical data grouped by sex , so that any null values can be appropriately filled with the most frequent value.

Null values in the 'blood sugar' column are filled with the mode of Blood sugar, calculated within groups based on sex.

In [136…
```
heartdata.groupby(['sex'])['blood sugar'].agg(lambda x: x.mode().iloc[0])
```

Out[136…
```
sex
0    0.0
1    0.0
Name: blood sugar, dtype: category
Categories (2, float64): [0.0, 1.0]
```

In [136…
```
heartdata['blood sugar'] = heartdata['blood sugar'].fillna(
    heartdata.groupby(['sex'])['blood sugar'].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the 'slope' column are filled with the mode of slope, calculated within groups based on sex.

In [136…
```
heartdata.groupby(['sex'])['slope'].agg(lambda x: x.mode().iloc[0])
```

Out[136…
```
sex
0    2.0
1    2.0
Name: slope, dtype: category
Categories (3, float64): [1.0, 2.0, 3.0]
```

In [137…
```
heartdata['slope'] = heartdata['slope'].fillna(
    heartdata.groupby(['sex'])['slope'].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the 'ca' column are filled with the mode of Ca, calculated within groups based on sex.

In [137…
```
heartdata.groupby(['sex'])['ca'].agg(lambda x: x.mode().iloc[0])
```

Out[137…
```
sex
0    0.0
1    0.0
Name: ca, dtype: category
Categories (4, float64): [0.0, 1.0, 2.0, 3.0]
```

In [137…
```
heartdata['ca'] = heartdata['ca'].fillna(
    heartdata.groupby(['sex'])['ca'].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the 'thal' column are filled with the mode of thal, calculated within groups based on sex.

In [137…
```
heartdata.groupby(['sex'])['thal'].agg(lambda x: x.mode().iloc[0])
```

Out[137…
```
sex
0    3.0
1    7.0
Name: thal, dtype: category
Categories (3, float64): [3.0, 6.0, 7.0]
```

In [138…
```
heartdata['thal'] = heartdata['thal'].fillna(
    heartdata.groupby(['sex'])['thal'].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the electrocardiographic column are filled with the mode of electrocardiographic, calculated within groups based on sex.

In [138…
```
heartdata.groupby(['sex'])['electrocardiographic '].agg(lambda x: x.mode().iloc[0])
```

Out[138…
```
sex
0    0.0
1    0.0
Name: electrocardiographic , dtype: category
Categories (3, float64): [0.0, 1.0, 2.0]
```

In [138…
```
heartdata['electrocardiographic '] = heartdata['electrocardiographic '].fillna(
    heartdata.groupby(['sex'])['electrocardiographic '].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the Blood Pressure column are filled with the mode of Blood Pressure, calculated within groups based on sex.

In [138…
```
heartdata.groupby(['sex'])['blood pressure'].agg(lambda x: x.mode().iloc[0])
```

Out[138…
```
sex
0    130.0
1    120.0
Name: blood pressure, dtype: float64
```

In [139…
```
heartdata['blood pressure'] = heartdata['blood pressure'].fillna(
    heartdata.groupby(['sex'])['blood pressure'].transform(lambda x: x.mode().iloc[0] ))
```

Null values in the cholestoral column are filled with the mean of cholestral, calculated within groups based on sex.

In [139…
```
heartdata.groupby(['sex'])['cholestoral '].mean()
```

Out[139…
```
sex
0    252.913971
1    243.860422
Name: cholestoral , dtype: float64
```

In [139…
```
heartdata['cholestoral '] = heartdata['cholestoral '].fillna(value =  heartdata.groupby(['sex'])['cholestoral '
```

Null values in the heart rate column are filled with the mode of heart rate, calculated within groups based on sex and chest pain.

In [139…
```
heartdata.groupby(['sex'])['heart rate'].mean()
```

Out[139…
```
sex
0    147.412791
1    143.455882
Name: heart rate, dtype: float64
```

In [140…
```
heartdata['heart rate'] = heartdata['heart rate'].fillna(value =  heartdata.groupby(['sex'])['heart rate'].tran
```

Null values in the depression column are filled with the mode of depression, calculated within groups based on sex and chest pain.

In [140…
```
heartdata.groupby(['sex'])['depression '].mean()
```

Out[140…
```
sex
0    0.635673
1    0.858437
Name: depression , dtype: float64
```

In [140…
```
heartdata['depression '] = heartdata['depression '].fillna(value =  heartdata.groupby(['sex'])['depression '].t
```

In [140…
```
pd.DataFrame({'Count':heartdata.isnull().sum()[heartdata.isnull().sum()>0], 'Percentage':heartdata.isnull().sum
```

Out[140…

Count	Percentage

Scaling

In [141…
```
from sklearn.preprocessing import MinMaxScaler

sc=MinMaxScaler()

heartdata_scaled=pd.DataFrame(sc.fit_transform(heartdata),columns=heartdata.columns,index=heartdata.index)
heartdata_scaled
```

Out[141…

	Age (age in year)	sex	chest pain	blood pressure	cholestoral	blood sugar	electrocardiographic	heart rate	exercise induced	depression	slope	ca	thal	c
0	0.714286	1.0	0.000000	0.679487	0.456181	1.0	1.0	0.584	0.0	0.613333	1.0	0.0	0.75	0.C
1	0.183673	1.0	0.666667	0.487179	0.518911	0.0	0.0	0.880	0.0	0.933333	1.0	0.0	0.00	0.C
2	0.265306	0.0	0.333333	0.487179	0.349170	0.0	1.0	0.760	0.0	0.373333	0.0	0.0	0.00	0.C
3	0.571429	1.0	0.333333	0.358974	0.467251	0.0	0.0	0.808	0.0	0.213333	0.0	0.0	0.00	0.C
4	0.591837	0.0	1.000000	0.358974	0.902675	0.0	0.0	0.688	1.0	0.160000	0.0	0.0	0.00	0.C
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
575	0.489796	1.0	1.000000	0.615385	0.577952	0.0	0.0	0.456	1.0	0.533333	0.5	0.0	1.00	1.C
576	0.306122	1.0	1.000000	0.615385	0.659133	0.0	0.0	0.464	1.0	0.533333	0.5	0.0	1.00	1.C
577	0.265306	1.0	1.000000	0.358974	0.836255	0.0	0.0	0.328	1.0	0.800000	0.5	0.0	1.00	1.C
578	0.326531	1.0	1.000000	0.551282	1.000000	0.0	0.0	0.464	0.0	0.000000	0.5	0.0	1.00	1.C
579	0.428571	1.0	1.000000	0.743590	0.415590	0.0	0.0	0.360	0.0	0.533333	0.5	0.0	1.00	1.C

580 rows × 14 columns

Duplicate

In [141…
```
#□□□□□□□ □□□ □□□□□
df_dub = heartdata.drop_duplicates()

print(heartdata.shape, df_dub.shape)
#□□□□□□□ □□□□□□□ □□□ □□
```
```
(580, 14) (580, 14)
```

No duplicated value is detected.

Data Splitting

In [141…
```
x = heartdata_scaled.drop("c", axis=1)
y = heartdata_scaled.c  # df["c"]

# x is my ind features
# y is my target
```

In [142…
```
from sklearn.model_selection import train_test_split
# stratify = y
```

In [142…
```
# from sklearn.model_selection import train_test_split

Xtrain, Xtest, Ytrain, Ytest = train_test_split(x , y , test_size = 0.2, random_state= 42, stratify=y)

print(Xtrain.shape)
print(Xtest.shape)
```
```
(464, 13)
(116, 13)
```

In [142…
```
Ytest.value_counts()
```

Out[142…
```
c
0.0    67
1.0    49
Name: count, dtype: int64
```

Modeling

Decision Tree

In [142…
```
from sklearn.tree import DecisionTreeClassifier
```

Hyperparameter Analysis

In [143…
```
hp_max=7

result=[]

for i in range(1,hp_max):

    DT=DecisionTreeClassifier(max_depth=i)
    DT.fit(Xtrain,Ytrain)
    result.append([i, DT.fit(Xtrain,Ytrain).score(Xtrain,Ytrain),accuracy_score(Ytest,DT.predict(Xtest))])
result
```

Out[143…
```
[[1, 0.771551724137931, 0.75],
 [2, 0.7974137931034483, 0.8017241379310345],
 [3, 0.834051724137931, 0.7931034482758621],
 [4, 0.8556034482758621, 0.8189655172413793],
 [5, 0.8793103448275862, 0.8017241379310345],
 [6, 0.8987068965517241, 0.8017241379310345]]
```

Max depth= 4 was selected

In [143…
```
DT = DecisionTreeClassifier(random_state = 42,max_depth = 4,min_samples_leaf = 1)

DT.fit(Xtrain, Ytrain)

pred = DT.predict(Xtest)
#□□□□□ □□□ □□□ □□□□□□□ □□□ □□□□□ □□□□□□□□

pred
```

```
Out[143… array([0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0.,
                0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
                0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0.,
                1., 0., 0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1.,
                0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 0., 0., 1.,
                0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In [143… pred == Ytest
```

```
Out[143… 54      True
         166     True
         506     True
         106     True
         322     True
                ...
         273     True
         375    False
         105     True
         71      True
         58      True
         Name: c, Length: 116, dtype: bool
```

```
In [143… (pred == Ytest).mean()
         # accuracy
```

```
Out[143… 0.8189655172413793
```

```
In [144… #Metrics:
         acc = accuracy_score(Ytest, pred)
         #□□□□ □□□□□□ □□ □□□□ □□ □□□ □□□□ □□□□
         #□□□□ □□□□□□ □□□□ □□ □□ □□ □□□□□□ □□□

         rec = recall_score(Ytest, pred)

         pre = precision_score(Ytest, pred)

         fm = f1_score(Ytest, pred)

         conf = confusion_matrix(Ytest, pred)

         print(acc, rec, pre, fm)
         print(conf)
```

```
0.8189655172413793 0.6122448979591837 0.9375 0.7407407407407407
[[65  2]
 [19 30]]
```

```
In [144… print(classification_report(Ytest, pred))
```

```
              precision    recall  f1-score   support

         0.0       0.77      0.97      0.86        67
         1.0       0.94      0.61      0.74        49

    accuracy                           0.82       116
   macro avg       0.86      0.79      0.80       116
weighted avg       0.84      0.82      0.81       116
```

```
In [144… tf= pd.DataFrame(data=result,columns=['Depth','Train','Test'])
         tf
```

Out[144…

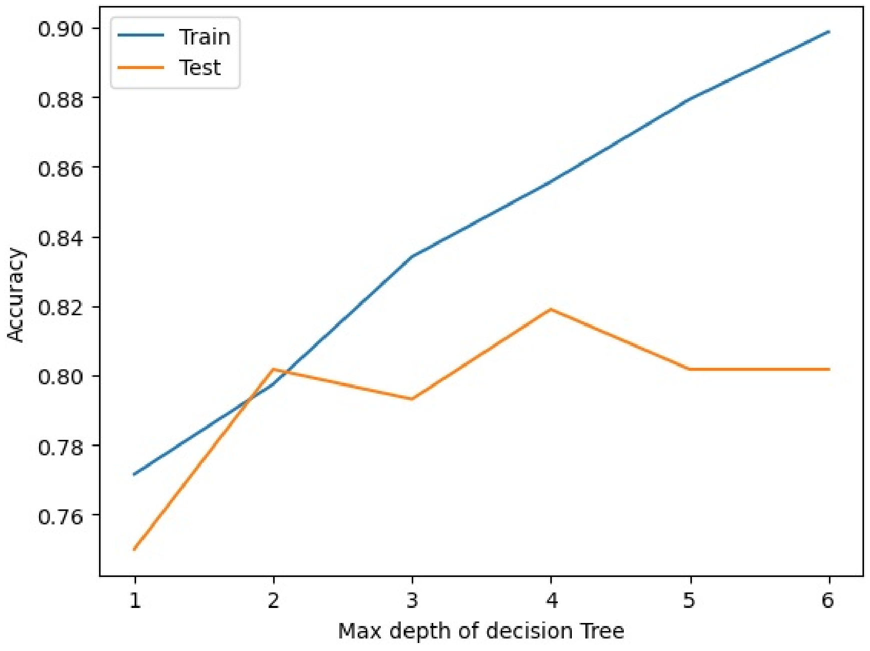
	Depth	Train	Test
0	1	0.771552	0.750000
1	2	0.797414	0.801724
2	3	0.834052	0.793103
3	4	0.855603	0.818966
4	5	0.879310	0.801724
5	6	0.898707	0.801724

```
In [144… tf.set_index('Depth',inplace=True)
         tf
```

Out[144…

Depth	Train	Test
1	0.771552	0.750000
2	0.797414	0.801724
3	0.834052	0.793103
4	0.855603	0.818966
5	0.879310	0.801724
6	0.898707	0.801724

In [144…
```
tf.plot(kind='line',xlabel='Max depth of decision Tree',ylabel='Accuracy');
```



In [145…
```
kf = KFold(10, shuffle = True, random_state = 42)
scorecross=cross_val_score(DT, x , y , cv = kf, scoring = 'accuracy' )
scorecross.mean()
```

Out[145…
```
0.7948275862068965
```

Concat Cross Val Score

In [145…
```
pred = cross_val_predict(DT, x, y, cv = 10)
confusion_matrix(y , pred)
```

Out[145…
```
array([[305,  32],
       [ 88, 155]])
```

In [145…
```
# def metric_name(x,y):
from sklearn.model_selection import GridSearchCV
DT = DecisionTreeClassifier( )

param = {'criterion':['gini', 'entropy'] ,
         'max_depth':[3,4,5,6,7] ,
         'min_samples_split':[3,4,5,6] ,
         'min_samples_leaf':[2,3,4]}

GS = GridSearchCV(DT, param ,cv = 10, scoring = "f1")
GS.fit(x, y)
#□□□□□□ □□□□□□ □□□ □□□□

# convex opt
# RandomSearch
# meta - Heurisitc
```

Out[145…
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [145…
```
GS = GridSearchCV(DT, param ,cv = 10, scoring = "f1")
%time GS.fit(x, y)
```

```
CPU times: user 2.6 s, sys: 4.59 ms, total: 2.6 s
Wall time: 2.62 s
```

Out[145…
```
▸           GridSearchCV
▸ estimator: DecisionTreeClassifier
     ▸ DecisionTreeClassifier
```

In [145…
```
GS= GridSearchCV(DT, param ,cv = 10, scoring = "f1",  n_jobs = -1)
%time GS.fit(x, y)
```

```
CPU times: user 278 ms, sys: 94.1 ms, total: 373 ms
Wall time: 2.27 s
```

Out[145…
```
▸           GridSearchCV
▸ estimator: DecisionTreeClassifier
     ▸ DecisionTreeClassifier
```

In [145…
```
GS.get_params()
```

Out[145…
```
{'cv': 10,
 'error_score': nan,
 'estimator__ccp_alpha': 0.0,
 'estimator__class_weight': None,
 'estimator__criterion': 'gini',
 'estimator__max_depth': None,
 'estimator__max_features': None,
 'estimator__max_leaf_nodes': None,
 'estimator__min_impurity_decrease': 0.0,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 2,
 'estimator__min_weight_fraction_leaf': 0.0,
 'estimator__monotonic_cst': None,
 'estimator__random_state': None,
 'estimator__splitter': 'best',
 'estimator': DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                        max_depth=None, max_features=None, max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        monotonic_cst=None, random_state=None, splitter='best'),
 'n_jobs': -1,
 'param_grid': {'criterion': ['gini', 'entropy'],
  'max_depth': [3, 4, 5, 6, 7],
  'min_samples_split': [3, 4, 5, 6],
  'min_samples_leaf': [2, 3, 4]},
 'pre_dispatch': '2*n_jobs',
 'refit': True,
 'return_train_score': False,
 'scoring': 'f1',
 'verbose': 0}
```

In [145…
```
from sklearn import set_config
set_config(print_changed_only= False)
DT
```

Out[145…
```
▾                    DecisionTreeClassifier
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       monotonic_cst=None, random_state=None, splitter='best')
```

In [146…
```
GS.best_params_
```

Out[146…
```
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_leaf': 4,
 'min_samples_split': 5}
```

In [146…
```
acc_DT=GS.best_score_
acc_DT
```

Out[146… 0.7274418820616491

KNN

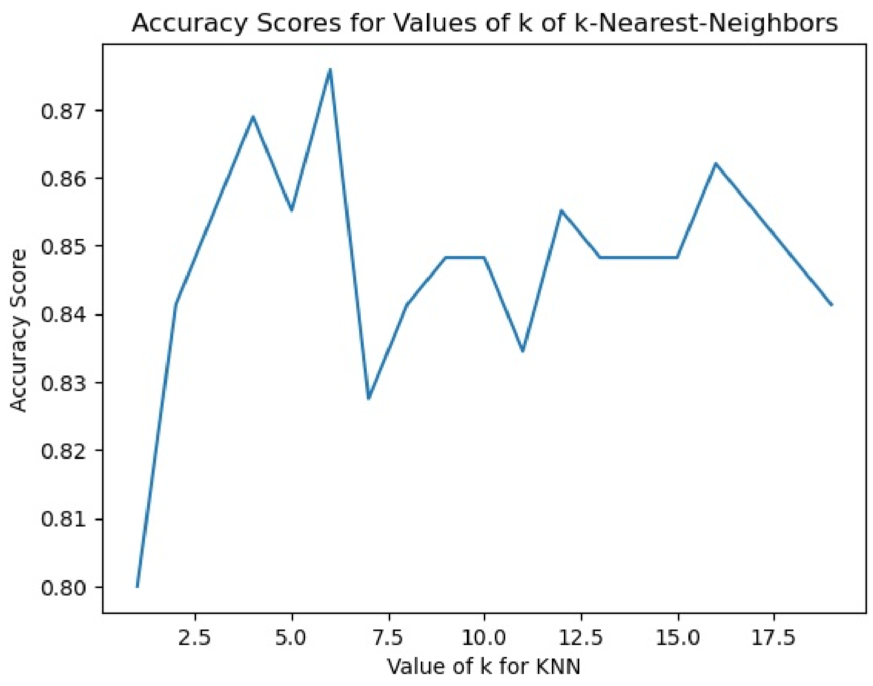
In [146…
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_predict , cross_val_score
```

In [146…
```
KNN = KNeighborsClassifier(n_neighbors = 7, weights = "distance")
scores = cross_val_score(KNN, x , y , cv =10 , scoring = 'accuracy').mean()
```

In [146…
```
import matplotlib.pyplot as plt
from sklearn import metrics

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,random_state=42)
```

In [146…
```
scores = []
k_range = list(range(1,20)) # >>[1,2,3,4,5, ... 25]
for i in k_range:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(metrics.accuracy_score(y_test, y_pred))

plt.plot(k_range, scores)

plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
```

Out[146…
```
Text(0.5, 1.0, 'Accuracy Scores for Values of k of k-Nearest-Neighbors')
```



In [146…
```
scores=np.array(scores)
scores
```

Out[146…
```
array([0.8       , 0.84137931, 0.85517241, 0.86896552, 0.85517241,
       0.87586207, 0.82758621, 0.84137931, 0.84827586, 0.84827586,
       0.83448276, 0.85517241, 0.84827586, 0.84827586, 0.84827586,
       0.86206897, 0.85517241, 0.84827586, 0.84137931])
```

In [146…
```
scores.mean()
```

Out[146…
```
0.8475499092558985
```

In [147…
```
k_range=range(1,15)
weights_custom=["uniform","distance"]

param_grid = dict(n_neighbors=k_range, weights=weights_custom)
KNNGreed = GridSearchCV(KNN, param_grid ,cv = 5, scoring = "accuracy")
KNNGreed.fit(x, y)
```

```
Out[147…  ▸          GridSearchCV
          ▸ estimator: KNeighborsClassifier
               ▸ KNeighborsClassifier
```

```
In [147… KNNGreed.cv_results_
```

```
Out[147… {'mean_fit_time': array([0.00130301, 0.00113258, 0.00125022, 0.00170555, 0.00062089,
                0.00063262, 0.00067306, 0.00062976, 0.00083013, 0.00057101,
                0.00074201, 0.00055408, 0.00058331, 0.00056872, 0.00055904,
                0.00054975, 0.00062146, 0.00060825, 0.00072684, 0.00059495,
                0.00058346, 0.00073786, 0.00055947, 0.00056047, 0.00054674,
                0.00054183, 0.00054507, 0.00053878]),
          'std_fit_time': array([7.51845864e-04, 6.69308696e-04, 8.31153435e-04, 1.40086430e-03,
                3.64720894e-05, 5.87757845e-05, 5.76714781e-05, 5.00289675e-05,
                2.57128680e-04, 1.78702638e-05, 1.09200644e-04, 1.47673319e-05,
                3.17655873e-05, 1.16455701e-05, 6.22706244e-06, 7.76878809e-06,
                8.91563846e-05, 2.10378695e-05, 8.35451633e-05, 4.00601678e-05,
                3.21769347e-05, 1.07782751e-04, 7.80994636e-06, 2.84470035e-05,
                5.56369052e-06, 6.92908546e-06, 7.45667209e-06, 6.67980614e-06]),
          'mean_score_time': array([0.00474958, 0.0018559 , 0.00424805, 0.00187235, 0.00324321,
                0.00106449, 0.00278244, 0.00109725, 0.00334525, 0.00105429,
                0.00288463, 0.0010746 , 0.00264063, 0.00108118, 0.00260096,
                0.00111246, 0.00277367, 0.00118384, 0.00303378, 0.00120139,
                0.00283775, 0.00146914, 0.00267591, 0.00120034, 0.00265465,
                0.0012044 , 0.00267015, 0.0012167 ]),
          'std_score_time': array([2.04382325e-03, 8.95732229e-04, 2.87450404e-03, 1.01815412e-03,
                1.38869617e-03, 5.47967437e-05, 1.29049227e-04, 4.30970583e-05,
                7.86860086e-04, 3.09485316e-05, 1.86478387e-04, 3.09868583e-05,
                3.66841211e-05, 2.12443664e-05, 1.70900726e-05, 3.05369435e-05,
                1.79726092e-04, 3.72142610e-05, 2.22723516e-04, 6.02539931e-05,
                2.01336245e-04, 2.12744146e-04, 6.02486344e-05, 4.00232581e-05,
                2.47241791e-05, 3.05795785e-05, 2.29579141e-05, 1.55139979e-05]),
          'param_n_neighbors': masked_array(data=[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9,
                            10, 10, 11, 11, 12, 12, 13, 13, 14, 14],
                      mask=[False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False,
                            False, False, False, False],
                fill_value='?',
                     dtype=object),
          'param_weights': masked_array(data=['uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance',
                            'uniform', 'distance', 'uniform', 'distance'],
                      mask=[False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False,
                            False, False, False, False, False, False, False, False,
                            False, False, False, False],
                fill_value='?',
                     dtype=object),
          'params': [{'n_neighbors': 1, 'weights': 'uniform'},
           {'n_neighbors': 1, 'weights': 'distance'},
           {'n_neighbors': 2, 'weights': 'uniform'},
           {'n_neighbors': 2, 'weights': 'distance'},
           {'n_neighbors': 3, 'weights': 'uniform'},
           {'n_neighbors': 3, 'weights': 'distance'},
           {'n_neighbors': 4, 'weights': 'uniform'},
           {'n_neighbors': 4, 'weights': 'distance'},
           {'n_neighbors': 5, 'weights': 'uniform'},
           {'n_neighbors': 5, 'weights': 'distance'},
           {'n_neighbors': 6, 'weights': 'uniform'},
           {'n_neighbors': 6, 'weights': 'distance'},
           {'n_neighbors': 7, 'weights': 'uniform'},
           {'n_neighbors': 7, 'weights': 'distance'},
           {'n_neighbors': 8, 'weights': 'uniform'},
           {'n_neighbors': 8, 'weights': 'distance'},
           {'n_neighbors': 9, 'weights': 'uniform'},
           {'n_neighbors': 9, 'weights': 'distance'},
           {'n_neighbors': 10, 'weights': 'uniform'},
           {'n_neighbors': 10, 'weights': 'distance'},
           {'n_neighbors': 11, 'weights': 'uniform'},
           {'n_neighbors': 11, 'weights': 'distance'},
           {'n_neighbors': 12, 'weights': 'uniform'},
           {'n_neighbors': 12, 'weights': 'distance'},
           {'n_neighbors': 13, 'weights': 'uniform'},
           {'n_neighbors': 13, 'weights': 'distance'},
  {'n_neighbors': 14, 'weights': 'uniform'},
  {'n_neighbors': 14, 'weights': 'distance'}],
 'split0_test_score': array([0.75      , 0.75      , 0.74137931, 0.75      , 0.72413793,
        0.72413793, 0.71551724, 0.73275862, 0.76724138, 0.76724138,
        0.74137931, 0.75      , 0.75      , 0.74137931, 0.75862069,
        0.75      , 0.75862069, 0.75862069, 0.75862069, 0.75      ,
        0.76724138, 0.76724138, 0.76724138, 0.75      , 0.75862069,
        0.76724138, 0.75862069, 0.75862069]),
 'split1_test_score': array([0.63793103, 0.63793103, 0.69827586, 0.63793103, 0.70689655,
        0.70689655, 0.73275862, 0.69827586, 0.73275862, 0.73275862,
        0.72413793, 0.71551724, 0.70689655, 0.70689655, 0.72413793,
        0.73275862, 0.72413793, 0.72413793, 0.74137931, 0.73275862,
        0.74137931, 0.74137931, 0.73275862, 0.73275862, 0.73275862,
        0.73275862, 0.71551724, 0.73275862]),
 'split2_test_score': array([0.79310345, 0.79310345, 0.77586207, 0.79310345, 0.79310345,
        0.78448276, 0.79310345, 0.81896552, 0.81034483, 0.79310345,
        0.79310345, 0.80172414, 0.80172414, 0.79310345, 0.80172414,
        0.81034483, 0.81896552, 0.81896552, 0.80172414, 0.81034483,
        0.81034483, 0.81034483, 0.81034483, 0.81034483, 0.81034483,
        0.81034483, 0.81896552, 0.81034483]),
 'split3_test_score': array([0.81896552, 0.81896552, 0.87068966, 0.81896552, 0.87068966,
        0.86206897, 0.87931034, 0.87068966, 0.87931034, 0.87068966,
        0.87931034, 0.86206897, 0.87068966, 0.86206897, 0.87931034,
        0.86206897, 0.88793103, 0.88793103, 0.87931034, 0.87931034,
        0.89655172, 0.89655172, 0.90517241, 0.87931034, 0.89655172,
        0.88793103, 0.9137931 , 0.89655172]),
 'split4_test_score': array([0.81896552, 0.81896552, 0.8362069 , 0.81896552, 0.82758621,
        0.85344828, 0.85344828, 0.8362069 , 0.84482759, 0.84482759,
        0.86206897, 0.85344828, 0.82758621, 0.8362069 , 0.8362069 ,
        0.82758621, 0.81896552, 0.81896552, 0.84482759, 0.82758621,
        0.84482759, 0.8362069 , 0.84482759, 0.82758621, 0.8362069 ,
        0.82758621, 0.84482759, 0.8362069 ]),
 'mean_test_score': array([0.7637931 , 0.7637931 , 0.78448276, 0.7637931 , 0.78448276,
        0.7862069 , 0.79482759, 0.79137931, 0.80689655, 0.80172414,
        0.8       , 0.79655172, 0.79137931, 0.78793103, 0.8       ,
        0.79655172, 0.80172414, 0.80172414, 0.80517241, 0.8       ,
        0.81206897, 0.81034483, 0.81206897, 0.8       , 0.80689655,
        0.80517241, 0.81034483, 0.80689655]),
 'std_test_score': array([0.06779174, 0.06779174, 0.06240331, 0.06779174, 0.06168463,
        0.06390949, 0.06437295, 0.06506194, 0.05246595, 0.05026683,
        0.06226024, 0.05707912, 0.05754593, 0.05759756, 0.05490236,
        0.0483374 , 0.05639802, 0.05639802, 0.05160907, 0.05325326,
        0.05517241, 0.05424873, 0.06007328, 0.05325326, 0.05785504,
        0.05302951, 0.06874966, 0.05785504]),
 'rank_test_score': array([26, 26, 24, 26, 25, 23, 19, 20,  5, 10, 13, 17, 20, 22, 13, 18, 10,
        10,  8, 13,  1,  3,  1, 13,  5,  9,  3,  5], dtype=int32)}
```

In [147…
```
KNNGreed.best_params_
```

Out[147…
```
{'n_neighbors': 11, 'weights': 'uniform'}
```

In [147…
```
acc_knn=KNNGreed.best_score_
acc_knn
```

Out[147…
```
0.8120689655172415
```

Neural Network

In [147…
```
from sklearn.neural_network import MLPClassifier
```

In [147…
```
MLP = MLPClassifier()

scores = cross_val_score(MLP, x , y , cv =10 , scoring = 'accuracy')
scores.mean()
```

Out[147…
```
0.8344827586206897
```

In [147…
```
from sklearn.model_selection import GridSearchCV

MLP = MLPClassifier(random_state = 42)

param = {"activation" : ["relu" , "logistic" , "tanh"],
         "hidden_layer_sizes":[(10), (20), (20,30)],
         "max_iter" : [10, 50, 100, 200],
         # "solver": ["sgd", "adam"],
         "learning_rate_init": [0.01, 0.001, 0.025]}

GS = GridSearchCV(MLP, param, cv = 10)
GS.fit(x , y)
```

Out[147…

```
▸ GridSearchCV
▸ estimator: MLPClassifier
  ▸ MLPClassifier
```

In [147…

```
GS.best_params_
```

Out[147…

```
{'activation': 'relu',
 'hidden_layer_sizes': (20, 30),
 'learning_rate_init': 0.025,
 'max_iter': 10}
```

In [147…

```
acc_nn=GS.best_score_
acc_nn
```

Out[147…

```
0.85
```

Logistic Regression

In [148…

```
from sklearn.linear_model import LogisticRegression
```

In [148…

```
import statsmodels.api as sm
```

In [148…

```
logreg = LogisticRegression( random_state = 42).fit(X_train,y_train)
logreg

print("Training set score: {:.3f}".format(logreg.score(X_train,y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test,y_test)))



import statsmodels.api as sm
# x = sm.add_constant(x)

logit_model=sm.Logit(y,x)
result=logit_model.fit()
print(result.summary())
```

```
Training set score: 0.832
Test set score: 0.862
Optimization terminated successfully.
         Current function value: 0.392219
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      c   No. Observations:                  580
Model:                          Logit   Df Residuals:                      567
Method:                           MLE   Df Model:                           12
Date:                Wed, 16 Oct 2024   Pseudo R-squ.:                  0.4232
Time:                        15:06:41   Log-Likelihood:                -227.49
converged:                       True   LL-Null:                       -394.37
Covariance Type:            nonrobust   LLR p-value:                 3.696e-64
=========================================================================================
                            coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Age (age in year)        -2.3886      0.667     -3.580      0.000      -3.696      -1.081
sex                       0.0608      0.356      0.171      0.864      -0.636       0.758
chest pain                1.1362      0.370      3.069      0.002       0.411       1.862
blood pressure           -0.2886      0.569     -0.507      0.612      -1.404       0.826
cholestoral               0.0875      0.605      0.145      0.885      -1.098       1.273
blood sugar               0.2550      0.389      0.656      0.512      -0.507       1.017
electrocardiographic      0.5280      0.303      1.745      0.081      -0.065       1.121
heart rate               -4.0220      0.566     -7.111      0.000      -5.131      -2.913
exercise induced          1.0964      0.284      3.860      0.000       0.540       1.653
depression                2.9783      0.567      5.252      0.000       1.867       4.090
slope                    -0.3913      0.509     -0.769      0.442      -1.389       0.606
ca                        3.6927      0.695      5.310      0.000       2.330       5.056
thal                      1.4656      0.355      4.132      0.000       0.770       2.161
=========================================================================================
```

In [148…

```
acc_lr=0.82
```

Support Vector Machines

In [148…

```
from sklearn.svm import SVC
```

In [149…
```
SVM = SVC()
score = cross_val_score(SVM , x, y, cv = 10 )
score.mean()
```

Out[149…
```
0.8396551724137931
```

In [150…
```
SVM = SVC(random_state = 42 , class_weight = {0:0.4 , 1:0.6})

param = [{"kernel" : ["linear"] ,"C" : [0.01 , 0.1, 1, 10, 100]},
         {"kernel" : ["rbf"], "gamma" : [0.01, 0.1, 0.2, 0.3], "C":  [0.01 , 0.1, 1, 10, 100]},
         {"kernel" : ["poly"], "degree": [2], "C": [0.01, 0.1, 1, 10, 100]}]

############ HINT
#□□□ □□□ □□ □□□□□□□ □□ □□□□□ □□ □□□□□□ □□□□□ □□
#□□□ □□ □□□□□□ □□ □□□ □□□□ □□□ □□□□□□□

GS = GridSearchCV(SVM, param, cv = 5, scoring = "accuracy" )

GS.fit(x , y)
```

Out[150…
```
▸  GridSearchCV
  ▸ estimator: SVC
       ▸ SVC
```

In [150…
```
GS.best_params_
```

Out[150…
```
{'C': 1, 'kernel': 'linear'}
```

In [150…
```
acc_svm=GS.best_score_
acc_svm
```

Out[150…
```
0.8172413793103448
```

Naive Bayes

In [150…
```
from sklearn.naive_bayes import GaussianNB , MultinomialNB
```

In [151…
```
GNB = GaussianNB()
scores = cross_val_score(GNB, x , y , cv =10 , scoring = 'accuracy')
acc_nbg=scores.mean()
acc_nbg
```

Out[151…
```
0.8206896551724139
```

In [151…
```
#□□ □□□□ □□□ □□□□□ □□□ □□□□
MNB = MultinomialNB()
scores = cross_val_score(MNB, x , y , cv =10 , scoring = 'accuracy')
acc_nbm=scores.mean()
acc_nbm
```

Out[151…
```
0.789655172413793
```

In [151…
```
pd.DataFrame(
    [acc_DT, acc_nn, acc_knn, acc_lr, acc_nbg, acc_nbm, acc_svm],
    index=['Decision Tree', 'Neural Network', 'K-Nearest Neighbor', 'Logistic Regression', 'Naive Bayes Gaussia
    columns=['Cross Val. Accuracy'])
```

Out[151…

	Cross Val. Accuracy
Decision Tree	0.727442
Neural Network	0.850000
K-Nearest Neighbor	0.812069
Logistic Regression	0.820000
Naive Bayes Gaussian	0.820690
Naive Bayes Multi	0.789655
Support Vector Machine	0.817241