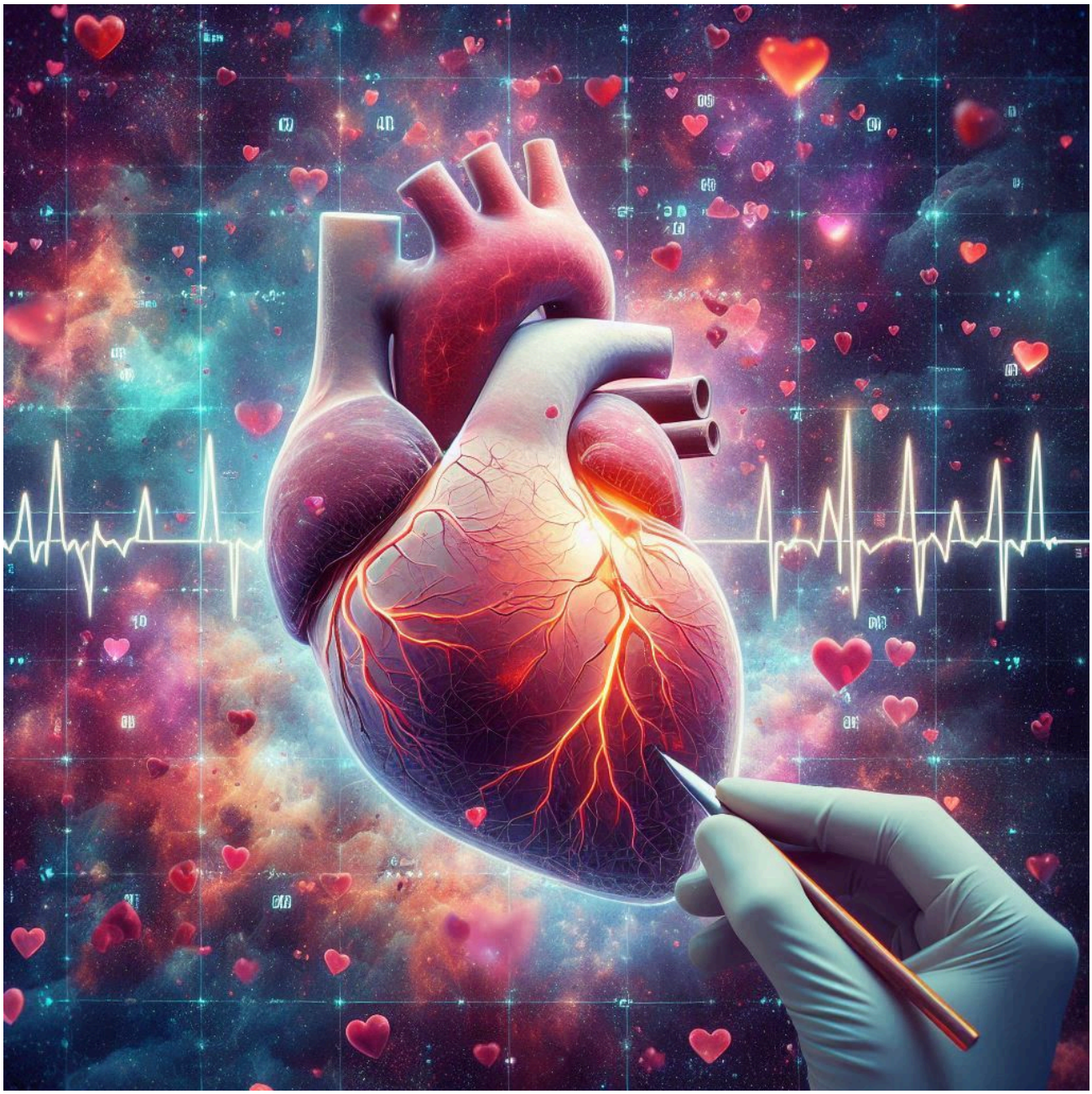
Heart Disease Prediction Model Development



About Author

- Project: Heart Disease Prediction
- Author: Muhammad Waqas
- Author's Contact Info:
  - **Email:** waqasliaqat630@gmailcom
  - Linkedin
  - Github

- kaggle

Meta-Data of Dataset

`Date:` 09-07-2024
`Dataset:` Heart Disease UCI

Context

This is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, oldpeak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more.

Column Descriptions:

- `id` (Unique id for each patient)
- `age` (Age of the patient in years)
- `origin` (place of study)
- `sex` (Male/Female)
- `cp` chest pain type ([typical angina, atypical angina, non-anginal, asymptomatic])
- `trestbps` resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))
- `chol` (serum cholesterol in mg/dl)
- `fbs` (if fasting blood sugar > 120 mg/dl)
- `restecg` (resting electrocardiographic results)
- `-- Values` : [normal, stt abnormality, lv hypertrophy]
- `thalach` : maximum heart rate achieved
- `exang` : exercise-induced angina (True/ False)
- `oldpeak` : ST depression induced by exercise relative to rest
- `slope` : the slope of the peak exercise ST segment
- `ca` : number of major vessels (0-3) colored by fluoroscopy
- `thal` : [normal; fixed defect; reversible defect]
- `num` : the predicted attribute

Acknowledgements

Creators:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Relevant Papers:

- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64,304--310.
- David W. Aha & Dennis Kibler. "Instance-based prediction of heart-disease presence with the Cleveland database."
- Gennari, J.H., Langley, P, & Fisher, D. (1989). Models of incremental concept formation. Artificial Intelligence, 40, 11--61.

Citation Request: The authors of the databases have requested that any publications resulting from the use of the data include the names of the principal investigator responsible for the data collection at each institution. They would be:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation:Robert Detrano, M.D., Ph.D.

Objective

```
To find best model with best parameters with more accuracy.
```

- Let's start with chapter 01.

Chapter 1: Import Libraries

In [1]:
```
# To handle data
import pandas as pd
import numpy as np
# to visualize data
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
%matplotlib inline
# removing warnings
import warnings
warnings.filterwarnings('ignore')
# pickle
import pickle
# sklearn imputers
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import SimpleImputer
# for normality tests
from scipy.stats import shapiro,chi2_contingency
# preprocessing
from sklearn.preprocessing import MinMaxScaler # min max scaler is for normalizing data and m
from sklearn.preprocessing import LabelEncoder
# for train test split
```

```
from sklearn.model_selection import train_test_split
# Loading Matrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
# parameter tuning
from sklearn.model_selection import GridSearchCV
```

Chapter 2: Load Dataset

In [2]:
```
df=pd.read_csv('heart_disease_uci.csv')
# show first 5 rows
df.head()
```

Out[2]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpea
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1

Chapter 3: Feeling the dataset

3.1. shape of dataset

In [3]:
```
df.shape
```

Out[3]: (920, 16)

3.2. sneak peak of data

In [4]:
```
df.head()
```

Out[4]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpea
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1

3.3. information about dataset

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object
 13  ca        309 non-null    float64
 14  thal      434 non-null    object
 15  num       920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

Observations:

1. num and ca looks categorical but they are in integers.
2. As id is unique and have no significance for prediction, we can remove it from dataset.

3.4. check for missing values

In [6]:
```
# number of missing values in each column in descending order
df.isnull().sum().sort_values(ascending=False)
```

Out[6]:
```
ca          611
thal        486
slope       309
fbs          90
oldpeak      62
trestbps     59
thalch       55
exang        55
chol         30
restecg       2
id            0
age           0
sex           0
dataset       0
cp            0
num           0
dtype: int64
```

In [7]:
```
# checking percentage of null
df.isnull().sum().sort_values(ascending=False)/len(df)*100
```

Out[7]:
```
ca          66.413043
thal        52.826087
slope       33.586957
fbs          9.782609
oldpeak      6.739130
trestbps     6.413043
thalch       5.978261
exang        5.978261
chol         3.260870
restecg      0.217391
id           0.000000
age          0.000000
sex          0.000000
dataset      0.000000
cp           0.000000
num          0.000000
dtype: float64
```

3.5. Statistical Description

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

Out[8]:

	id	age	trestbps	chol	thalch	oldpeak	ca	num
count	920.000000	920.000000	861.000000	890.000000	865.000000	858.000000	309.000000	920.000000
mean	460.500000	53.510870	132.132404	199.130337	137.545665	0.878788	0.676375	0.995652
std	265.725422	9.424685	19.066070	110.780810	25.926276	1.091226	0.935653	1.142693
min	1.000000	28.000000	0.000000	0.000000	60.000000	-2.600000	0.000000	0.000000
25%	230.750000	47.000000	120.000000	175.000000	120.000000	0.000000	0.000000	0.000000
50%	460.500000	54.000000	130.000000	223.000000	140.000000	0.500000	0.000000	1.000000
75%	690.250000	60.000000	140.000000	268.000000	157.000000	1.500000	1.000000	2.000000
max	920.000000	77.000000	200.000000	603.000000	202.000000	6.200000	3.000000	4.000000

Observations:

According to this dataset:

1. minimum age for heart disease is 28.
2. Average age of heart disease patients is `53.5`
3. 75 percent of heart patients have age greater than 47.
4. minimum trestbps is spotted 0, which is expeced to be as outlier.
5. Average trestbps is 132.
6. Maximum heart rate achieved(thalch) start from 60 and end at 202.
7. Between 25-50 percent of people, most of the people did not had any heart disease.

3.6. Checking for duplicates

In [9]:
```
# Checking for duplicates
df.duplicated().sum()
```

Out[9]:
```
0
```

- no duplicate record found.

3.7. Checking Correlation

In [10]:
```
# separating numerical and categorical features
num_features = []
categorical_features = []
for i in df.columns:
    if df[i].dtypes == 'object':
        categorical_features.append(i)
    elif df[i].dtype == 'int64' or df[i].dtype == 'float':
        num_features.append(i)
    else:
        print("no")
```

In [11]:
```
# Checking nm and ca
print(df['num'].unique())
print(df['ca'].unique())
```

```
[0 2 1 3 4]
[ 0.  3.  2.  1. nan]
```

In [12]:
```
# moving num and ca to categorical features
categorical_features.append('num')
categorical_features.append('ca')
# dropping num and ca from num_features
num_features.remove('num')
num_features.remove('ca')
```

In [13]:
```
df[num_features].corr()
```

Out[13]:

	id	age	trestbps	chol	thalch	oldpeak
id	1.000000	0.239301	0.052924	-0.376936	-0.466427	0.049930
age	0.239301	1.000000	0.244253	-0.086234	-0.365778	0.258243
trestbps	0.052924	0.244253	1.000000	0.092853	-0.104899	0.161908
chol	-0.376936	-0.086234	0.092853	1.000000	0.236121	0.047734
thalch	-0.466427	-0.365778	-0.104899	0.236121	1.000000	-0.151174
oldpeak	0.049930	0.258243	0.161908	0.047734	-0.151174	1.000000

Chapter 4: Univariate Analysis

4.1. age

In [14]:
```
# checking daatype of feature
df['age'].dtype
```

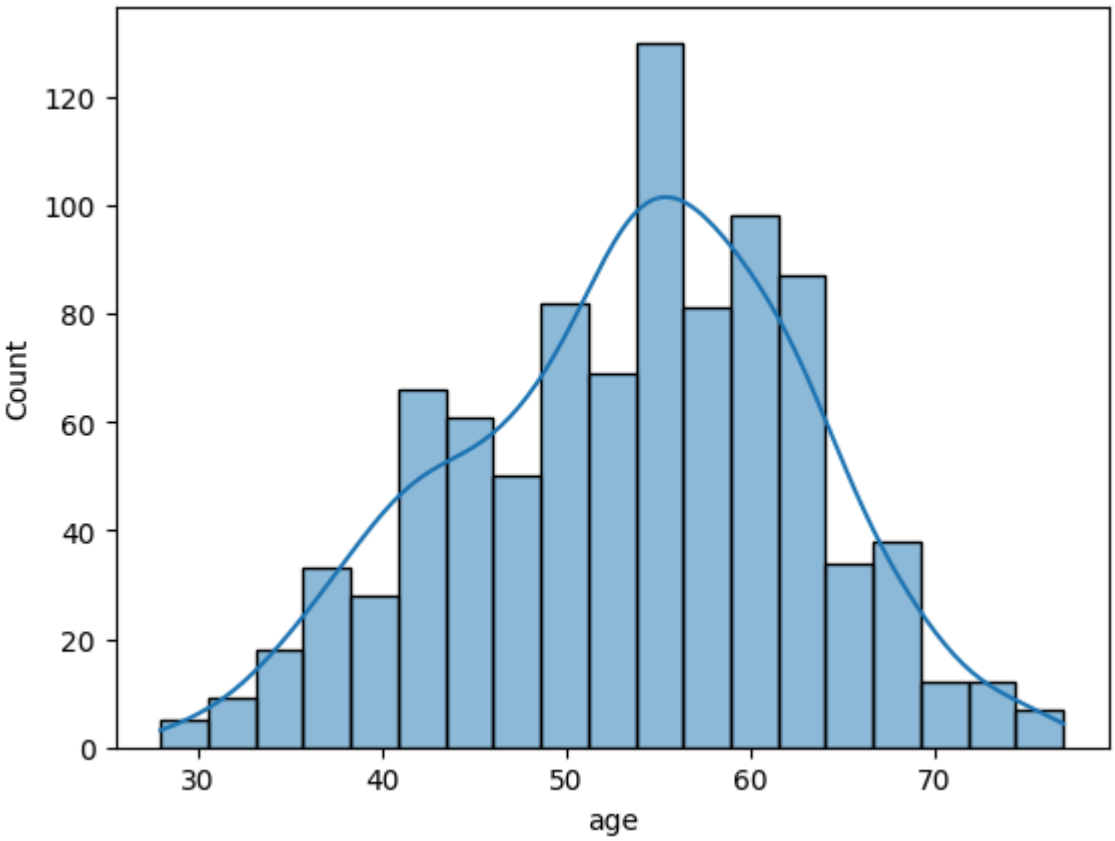
Out[14]:
```
dtype('int64')
```

In [15]:
```
# Checking null values
df['age'].isnull().sum()
```

Out[15]:
```
0
```

In [16]:
```
# for normality check
sns.histplot(data=df['age'],kde=True)
```

Out[16]:
```
<Axes: xlabel='age', ylabel='Count'>
```

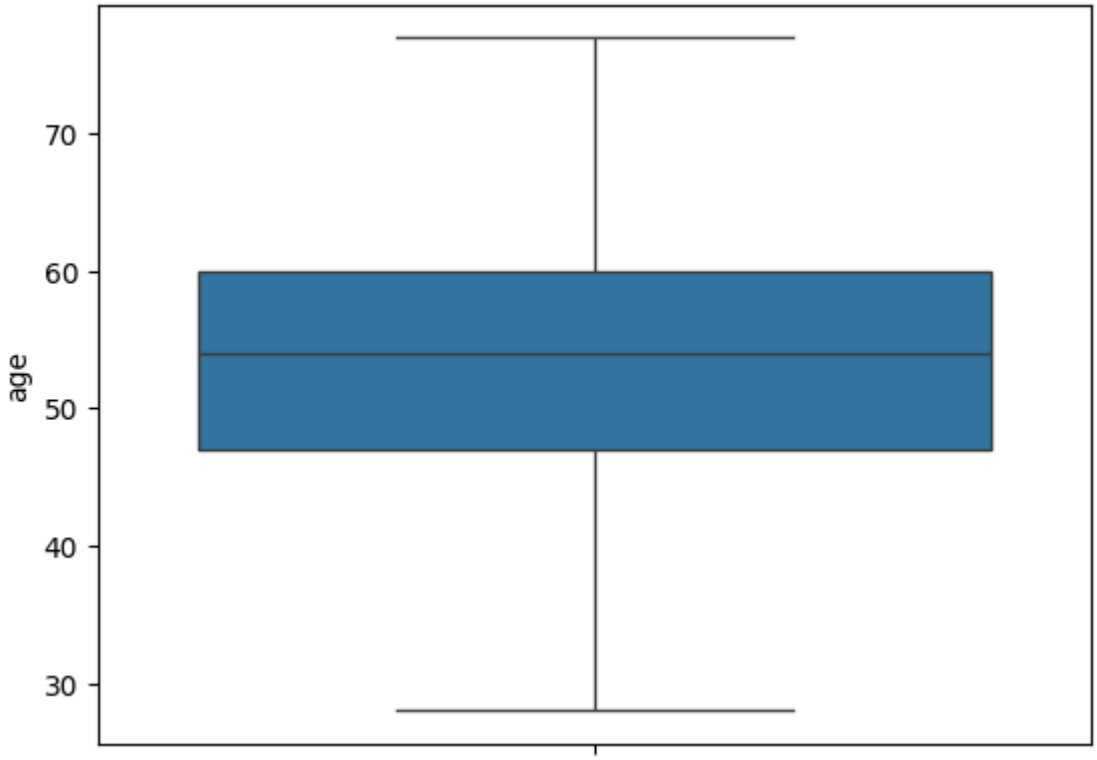


In [17]:
```
# statistical test for normality check
# H0: data is normal
# H1: data is not normal
stat,p=shapiro(df['age'])
if p>0.05:
    print('data is normal')
else:
    print('data is not normal')
```

```
data is not normal
```

In [18]:
```
# box plot for outliers
sns.boxplot(data=df['age'])
```

Out[18]:
```
<Axes: ylabel='age'>
```



4.2. sex

In [19]:
```
df['sex'].dtype
```

Out[19]:
```
dtype('O')
```

In [20]:
```
# Checking null values
df['sex'].isnull().sum()
```

Out[20]:
```
0
```

In [21]:
```
df['sex'].unique()
```

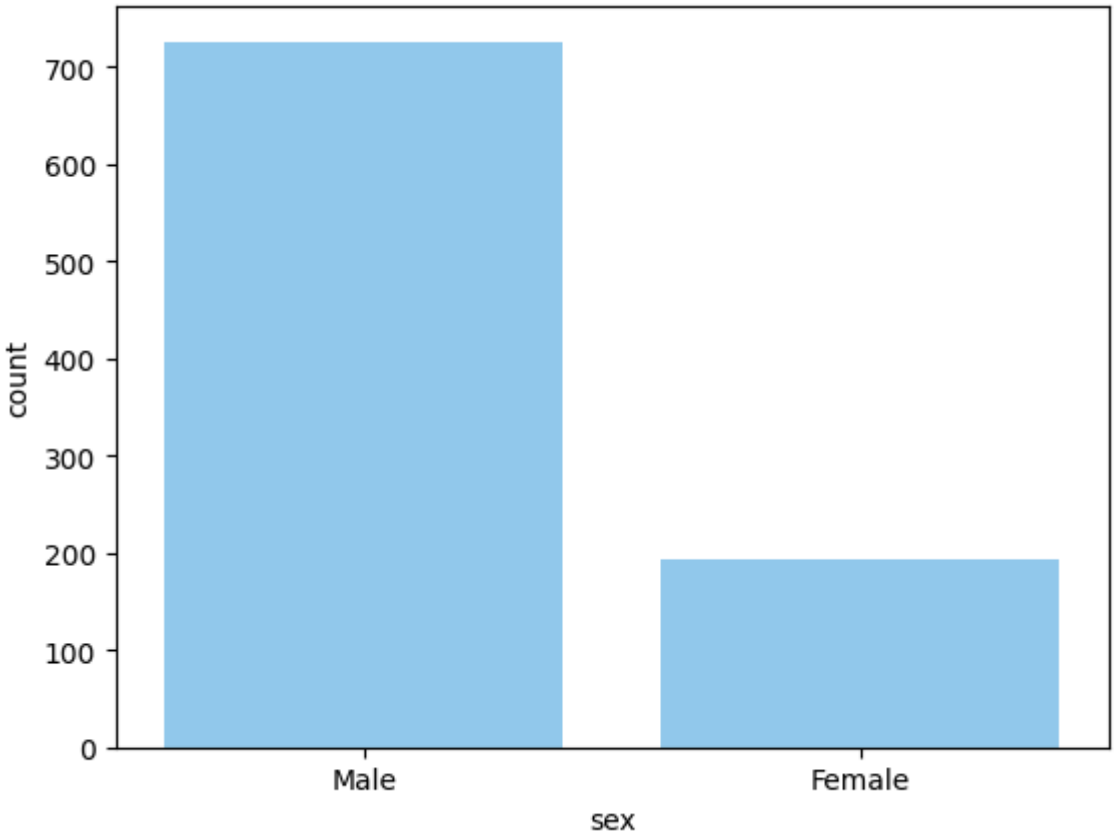
Out[21]:
```
array(['Male', 'Female'], dtype=object)
```

In [22]:
```
df[df['sex']=='Male']['sex'].count()/df[df['sex']=='Female']['sex'].count()
```

Out[22]:
```
3.7422680412371134
```

In [23]:
```
# countplot for sex
sns.countplot(data=df,x='sex',color='#87CEFA')
```

Out[23]:
```
<Axes: xlabel='sex', ylabel='count'>
```



4.3. dataset

In [24]:
```
df['dataset'].dtype
```

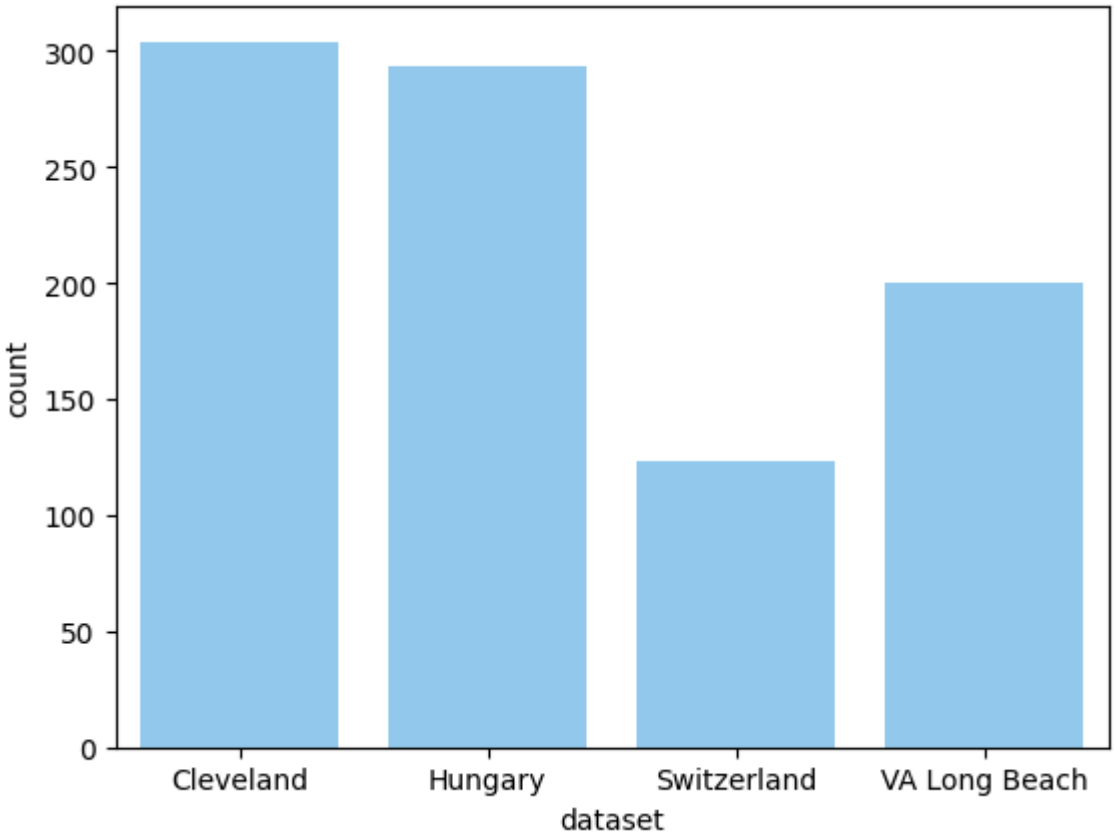
Out[24]:
```
dtype('O')
```

In [25]:
```
df['dataset'].unique()
```

Out[25]:
```
array(['Cleveland', 'Hungary', 'Switzerland', 'VA Long Beach'],
      dtype=object)
```

In [26]:
```
sns.countplot(data=df,x='dataset',color='#87CEFA')
```

Out[26]:
```
<Axes: xlabel='dataset', ylabel='count'>
```



4.4. cp

In [27]:
```
df['cp'].dtype
```

Out[27]:
```
dtype('O')
```

In [28]:
```
df['cp'].unique()
```

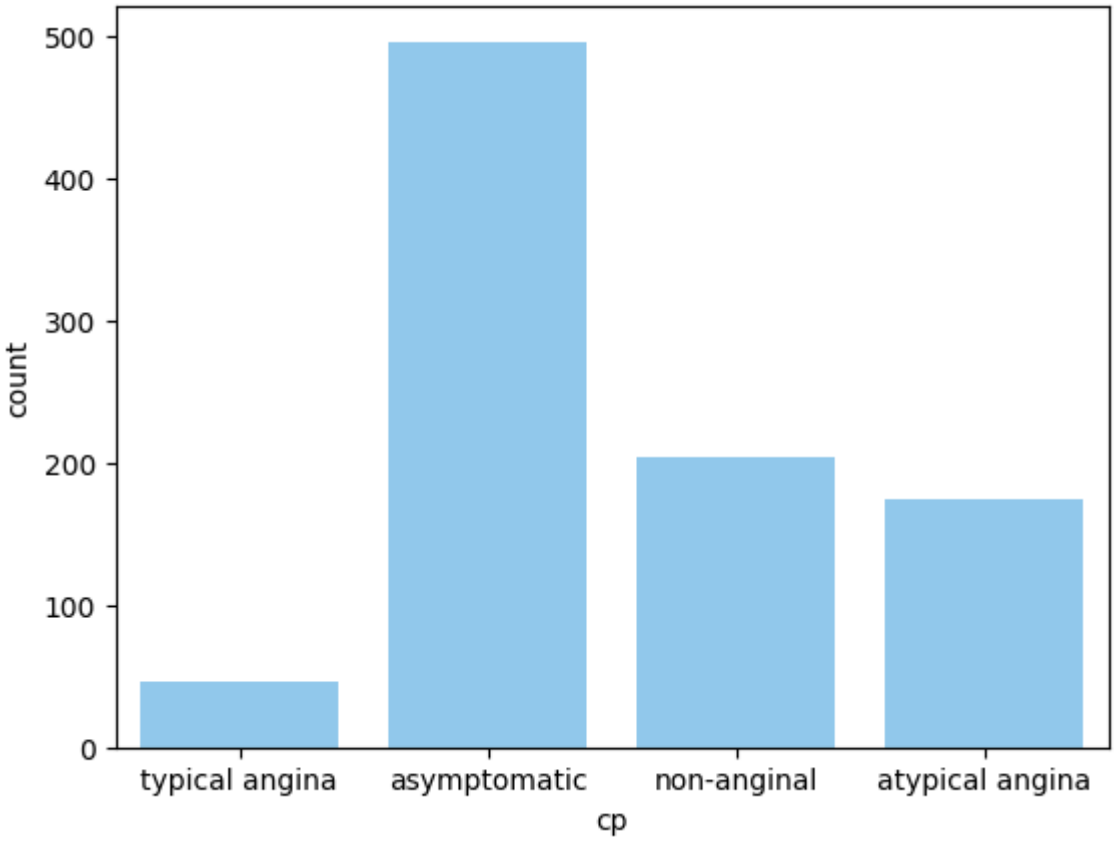
Out[28]:
```
array(['typical angina', 'asymptomatic', 'non-anginal', 'atypical angina'],
      dtype=object)
```

In [29]:
```
# Checking null values
df['cp'].isnull().sum()
```

Out[29]:
```
0
```

In [30]:
```
sns.countplot(data=df,x='cp',color='#87CEFA')
```

Out[30]:
```
<Axes: xlabel='cp', ylabel='count'>
```



In [31]:
```
df['cp'].value_counts()
```

Out[31]:
```
cp
asymptomatic       496
non-anginal        204
atypical angina    174
typical angina      46
Name: count, dtype: int64
```

In [32]:
```
print(f"Percentage of asymptotic chest pain is {df[df['cp']=='asymptomatic']['cp'].count()/le
print(f"Percentage of typical angina chest pain is {df[df['cp']=='typical angina']['cp'].coun
```

```
Percentage of asymptotic chest pain is 53.91304347826087
Percentage of typical angina chest pain is 5.0
```

4.5. trestbps

In [33]:
```
df['trestbps'].dtype
```

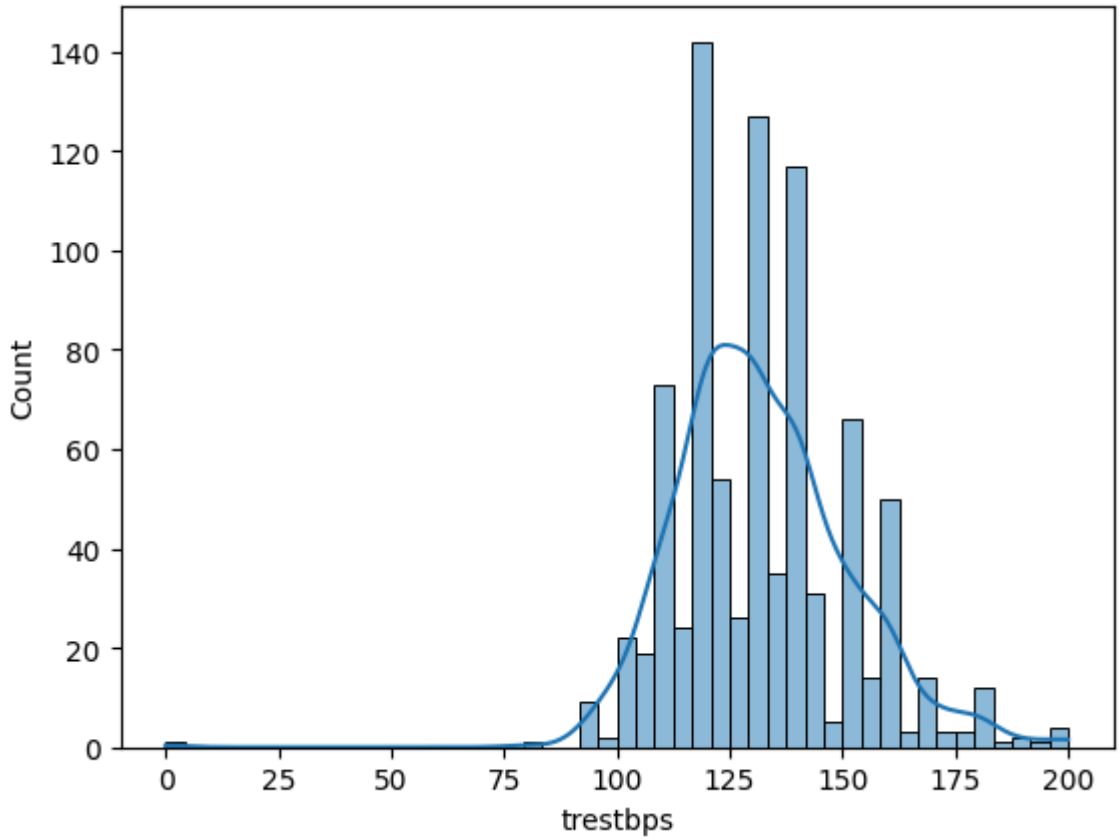
Out[33]:
```
dtype('float64')
```

In [34]:
```
df['trestbps'].isnull().sum()
```

Out[34]:
```
59
```

In [35]:
```
sns.histplot(data=df['trestbps'],kde=True)
```

Out[35]:
```
<Axes: xlabel='trestbps', ylabel='Count'>
```

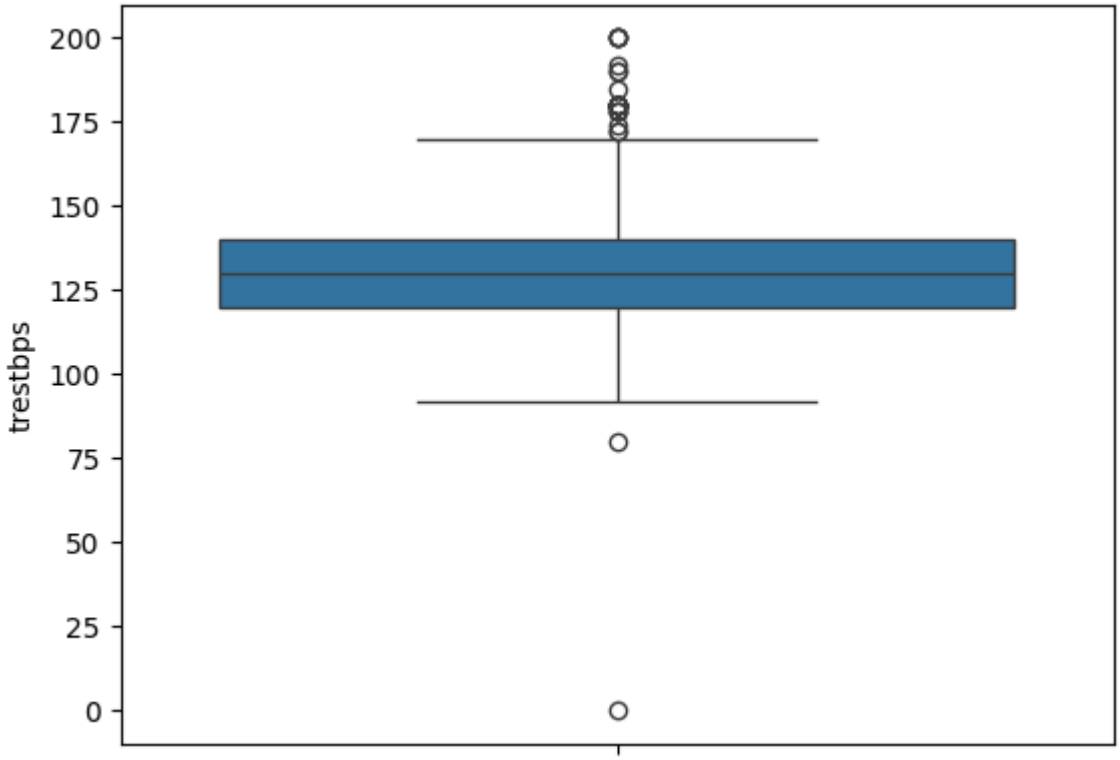


In [36]:
```
df['trestbps'].skew()
```

Out[36]: 0.21333446967212508

In [37]:
```
sns.boxplot(df['trestbps'])
```

Out[37]: <Axes: ylabel='trestbps'>



4.6. chol

In [38]:
```
df['chol'].dtype
```

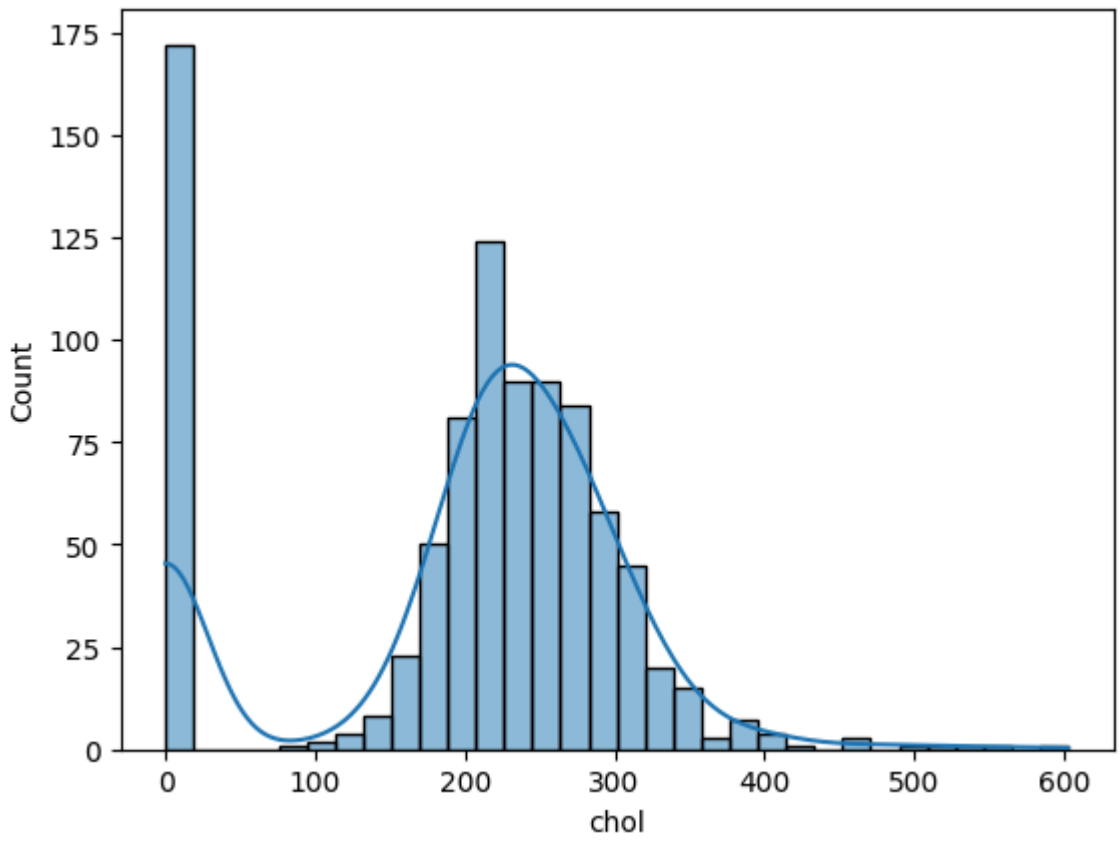
Out[38]: dtype('float64')

In [39]:
```
df['chol'].isnull().sum()
```

Out[39]: 30

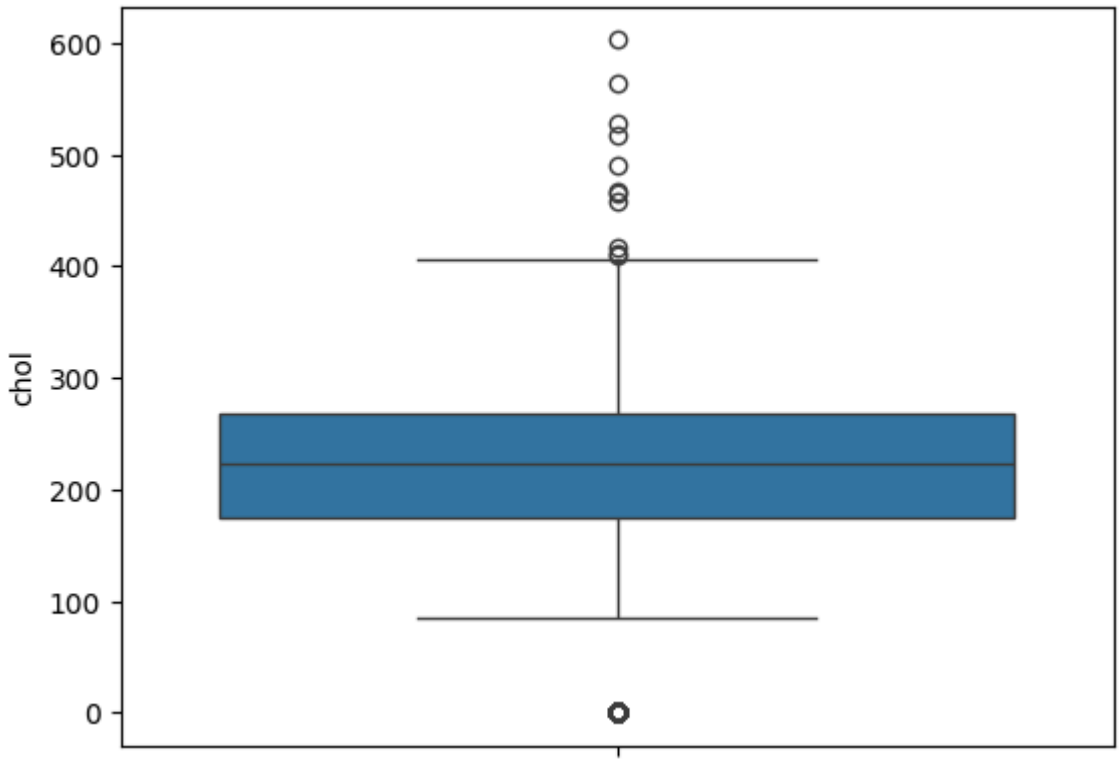
In [40]:
```
sns.histplot(data=df['chol'],kde=True)
```

Out[40]: <Axes: xlabel='chol', ylabel='Count'>



In [41]:
```
sns.boxplot(df['chol'])
```

Out[41]: <Axes: ylabel='chol'>



In [42]:
```
df[df['chol']==0]['chol'].count()
```

Out[42]: 172

4.7. fbs

In [43]:
```
df['fbs'].dtype
```

Out[43]: dtype('O')

In [44]:
```
df['fbs'].unique()
```

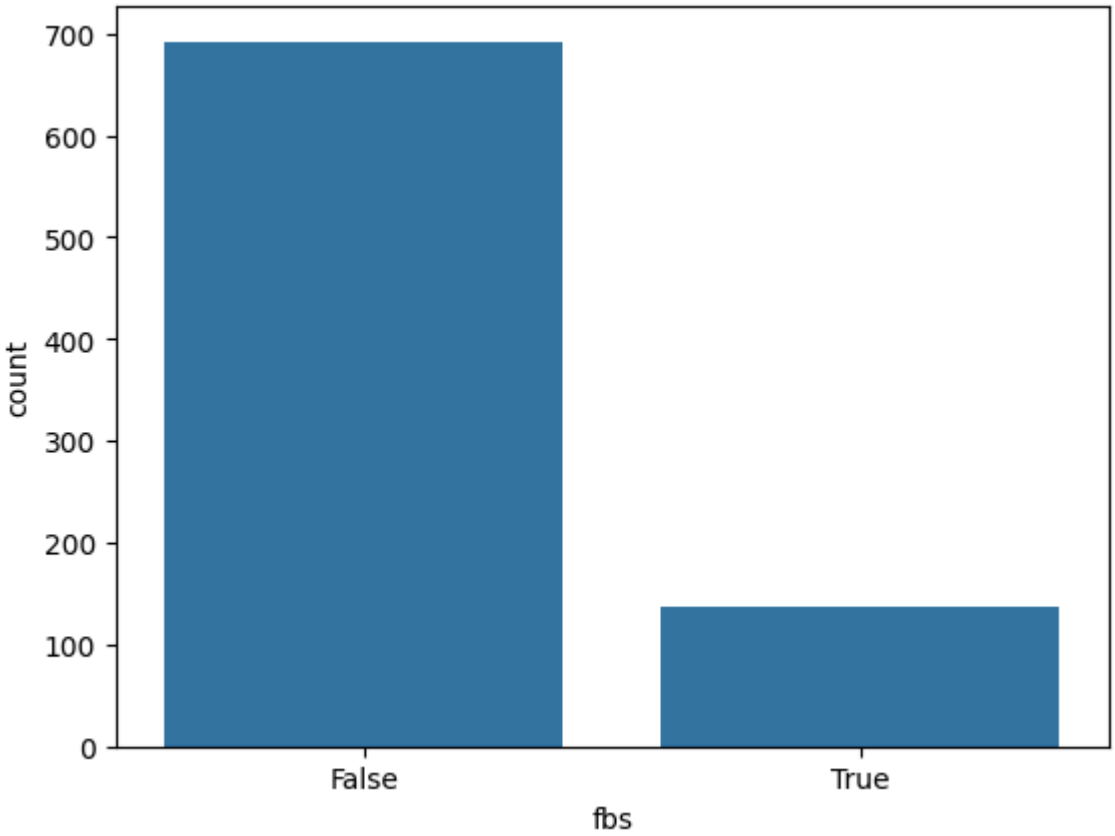
Out[44]: array([True, False, nan], dtype=object)

In [45]:
```
df['fbs'].isnull().sum()
```

Out[45]: 90

In [46]:
```
sns.countplot(x=df['fbs'])
```

Out[46]: <Axes: xlabel='fbs', ylabel='count'>



In [47]:
```
df[df['fbs']==False]['fbs'].count()/df[df['fbs']==True]['fbs'].count()
```

Out[47]: 5.0144927536231885

4.8. restecg

In [48]:
```
df['restecg'].dtype
```

Out[48]: dtype('O')

In [49]:
```
df['restecg'].unique()
```

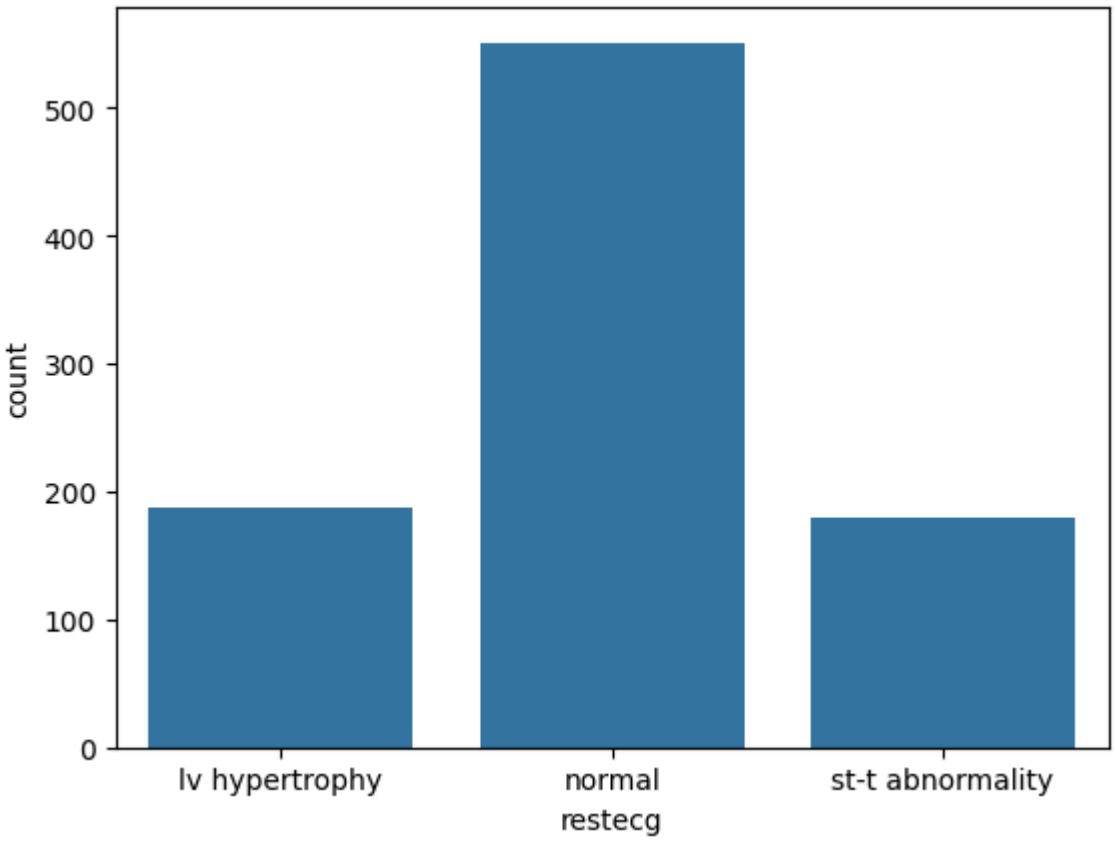
Out[49]: array(['lv hypertrophy', 'normal', 'st-t abnormality', nan], dtype=object)

In [50]:
```
df['restecg'].isnull().sum()
```

Out[50]: 2

In [51]:
```
sns.countplot(data=df,x='restecg')
```

Out[51]: <Axes: xlabel='restecg', ylabel='count'>



In [52]:
```
(df[df['restecg']=='normal']['restecg'].count()/len(df['restecg']))*100
```

Out[52]: 59.89130434782609

4.9. thalch

In [53]:
```
df['thalch'].dtype
```

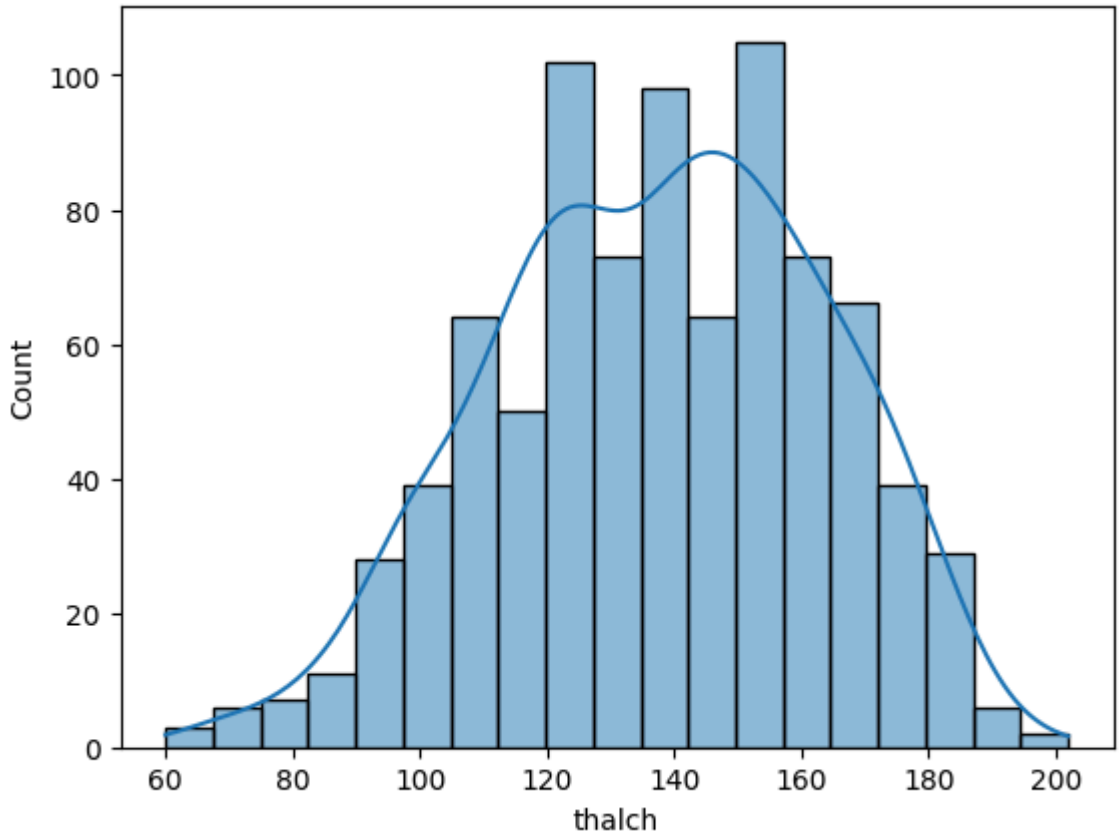
Out[53]: dtype('float64')

In [54]:
```
df['thalch'].isnull().sum()
```

Out[54]: 55

In [55]:
```
sns.histplot(data=df['thalch'],kde=True)
```

Out[55]: <Axes: xlabel='thalch', ylabel='Count'>

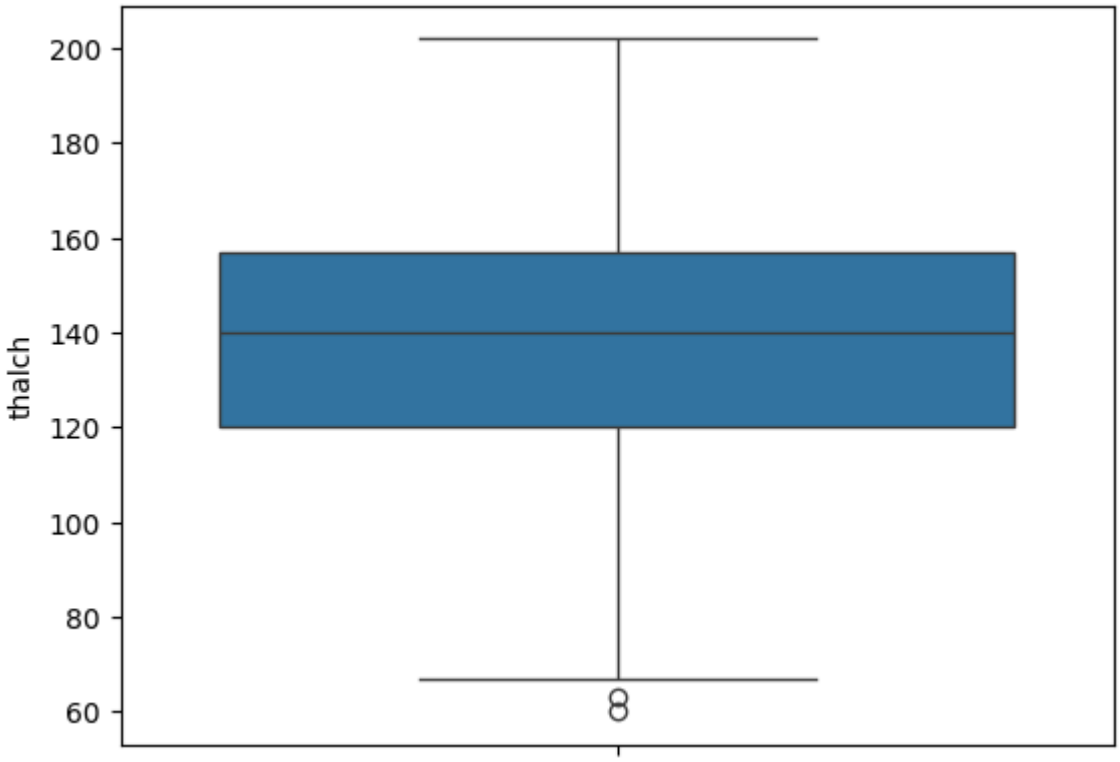


In [56]:
```
# statistical test for normality check
# H0: data is normal
# H1: data is not normal
stat,p=shapiro(df['thalch'])
if p>0.05:
    print('data is normal')
else:
    print('data is not normal')
```

```
data is not normal
```

In [57]:
```
sns.boxplot(df['thalch'])
```

Out[57]: <Axes: ylabel='thalch'>



4.10. exang

In [58]:
```
df['exang'].dtypes
```

Out[58]:
```
dtype('O')
```

In [59]:
```
df['exang'].isnull().sum()
```

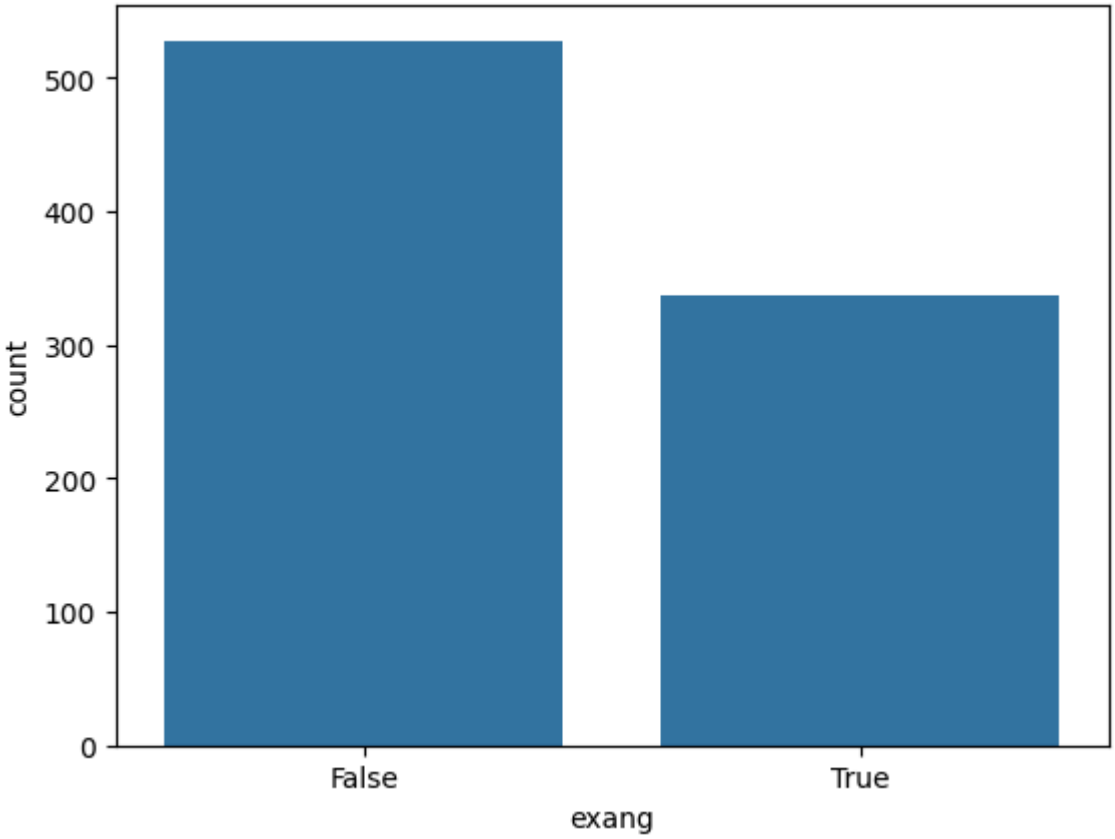
Out[59]:
```
55
```

In [60]:
```
df['exang'].unique()
```

Out[60]:
```
array([False, True, nan], dtype=object)
```

In [61]:
```
sns.countplot(data=df,x='exang')
```

Out[61]:
```
<Axes: xlabel='exang', ylabel='count'>
```



In [62]:
```
(len(df[df['exang']==True])/len(df['exang']))*100
```

Out[62]:
```
36.630434782608695
```

4.11. oldpeak

In [63]:
```
df['oldpeak'].dtype
```

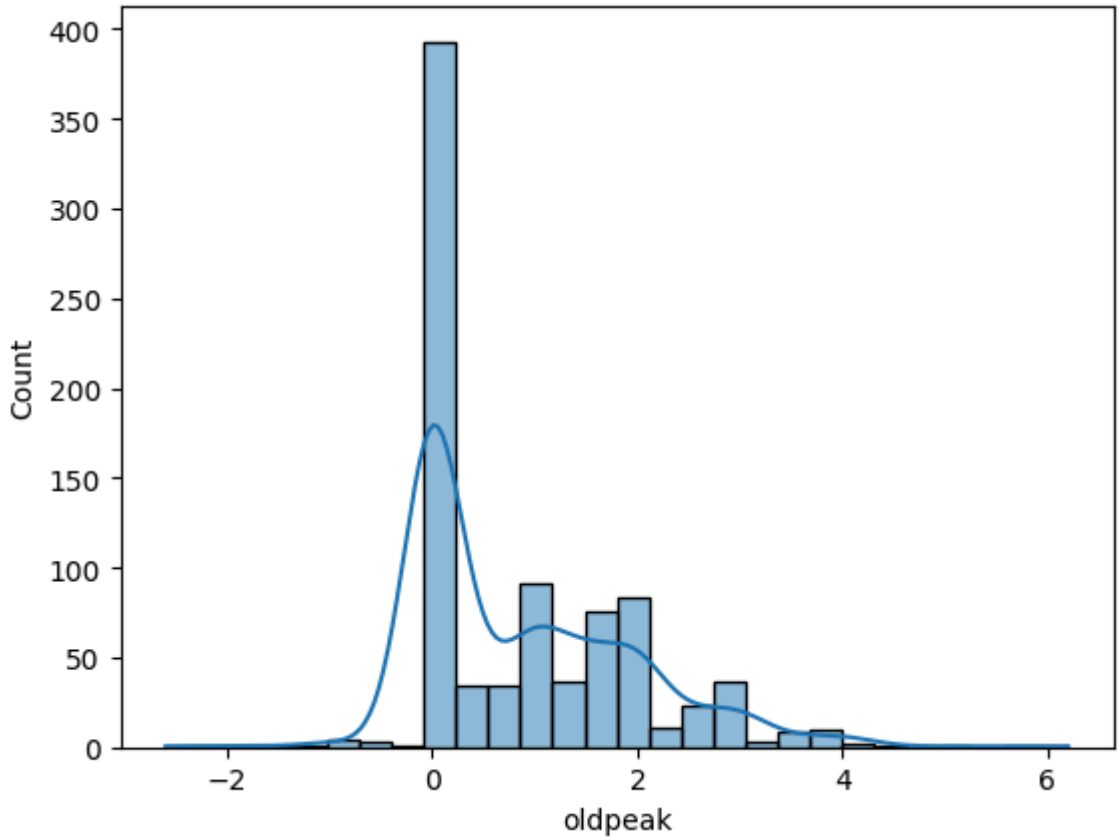
Out[63]:
```
dtype('float64')
```

In [64]:
```
df['oldpeak'].isnull().sum()
```

Out[64]:
```
62
```

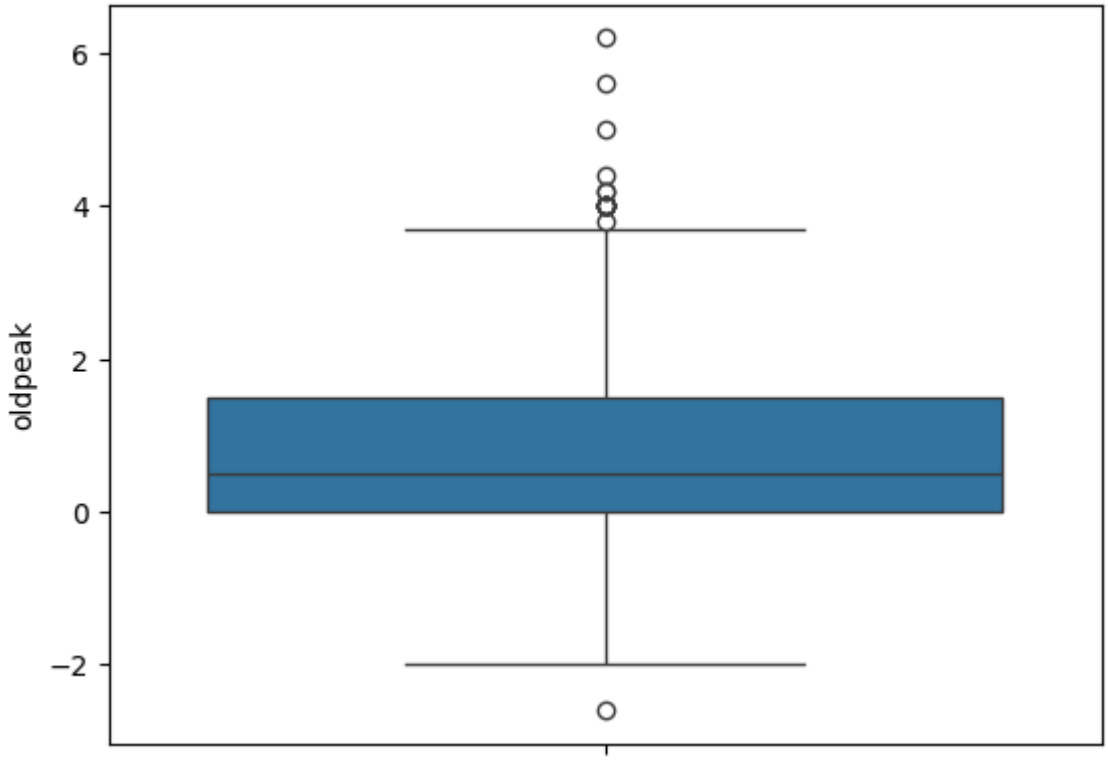
In [65]:
```
sns.histplot(data=df['oldpeak'],kde=True)
```

Out[65]:
```
<Axes: xlabel='oldpeak', ylabel='Count'>
```



In [66]:
```
sns.boxplot(df['oldpeak'])
```

Out[66]:
```
<Axes: ylabel='oldpeak'>
```



4.12. slope

In [67]:
```
df['slope'].dtype
```

Out[67]:
```
dtype('O')
```

In [68]:
```
df['slope'].isnull().sum()
```

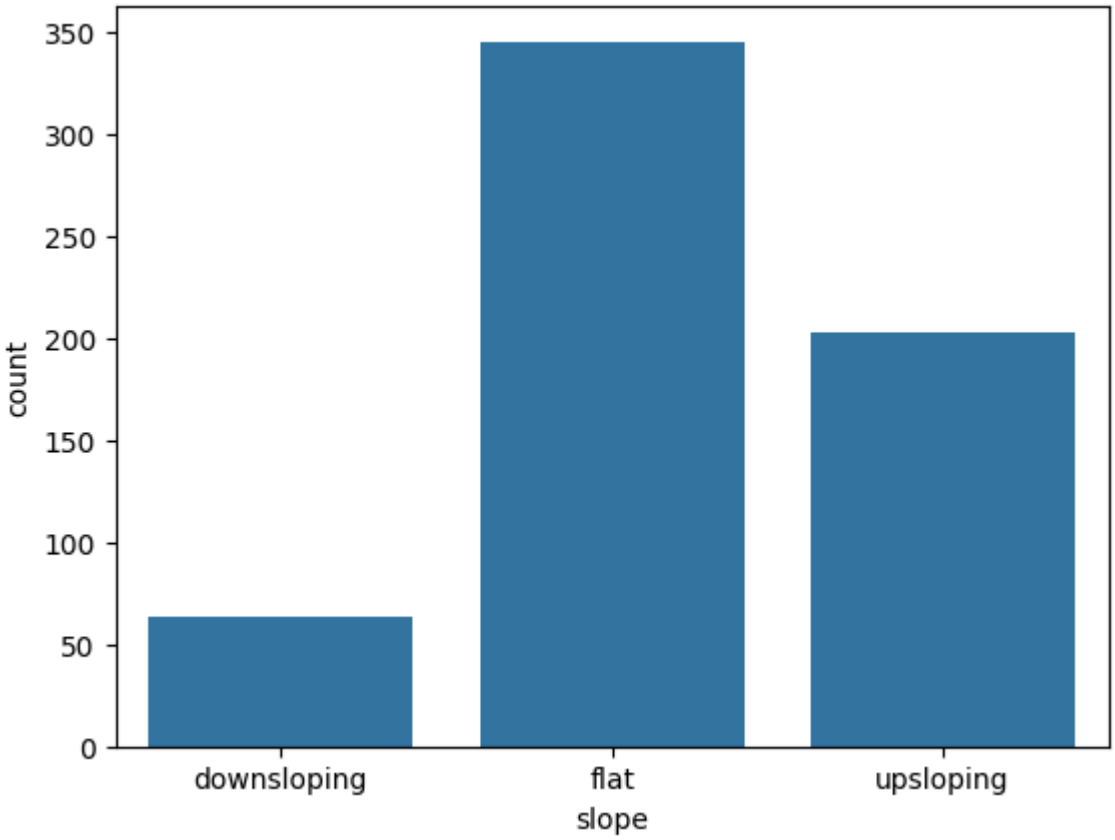
Out[68]:
```
309
```

In [69]:
```
df['slope'].unique()
```

Out[69]:
```
array(['downsloping', 'flat', 'upsloping', nan], dtype=object)
```

In [70]:
```
sns.countplot(data=df,x='slope')
```

Out[70]:
```
<Axes: xlabel='slope', ylabel='count'>
```



4.13. ca

In [71]:
```
df['ca'].dtype
```

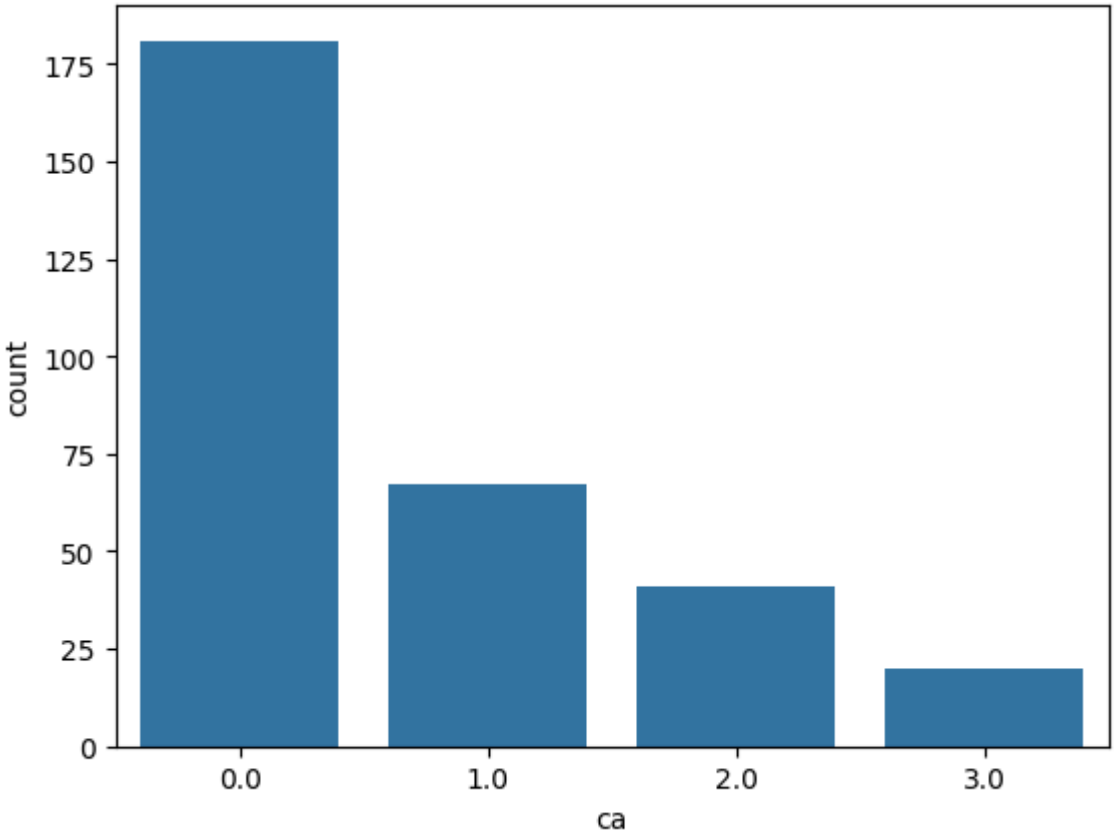
Out[71]:
```
dtype('float64')
```

In [72]:
```
df['ca'].unique()
```

Out[72]:
```
array([ 0.,  3.,  2.,  1., nan])
```

In [73]:
```
sns.countplot(data=df,x='ca')
```

Out[73]:
```
<Axes: xlabel='ca', ylabel='count'>
```



In [74]:
```
print(f"no vessel effected : {len(df[df['ca']==0.0])/len(df[df['ca'].notnull()])*100}%")
print(f"no vessel effected : {len(df[df['ca']==3.0])/len(df[df['ca'].notnull()])*100}%")
```

```
no vessel effected : 58.57605177993528%
no vessel effected : 6.472491909385113%
```

4.14. thal

In [75]:
```
df['thal'].dtype
```

Out[75]:
```
dtype('O')
```

In [76]:
```
df['thal'].isnull().sum()
```

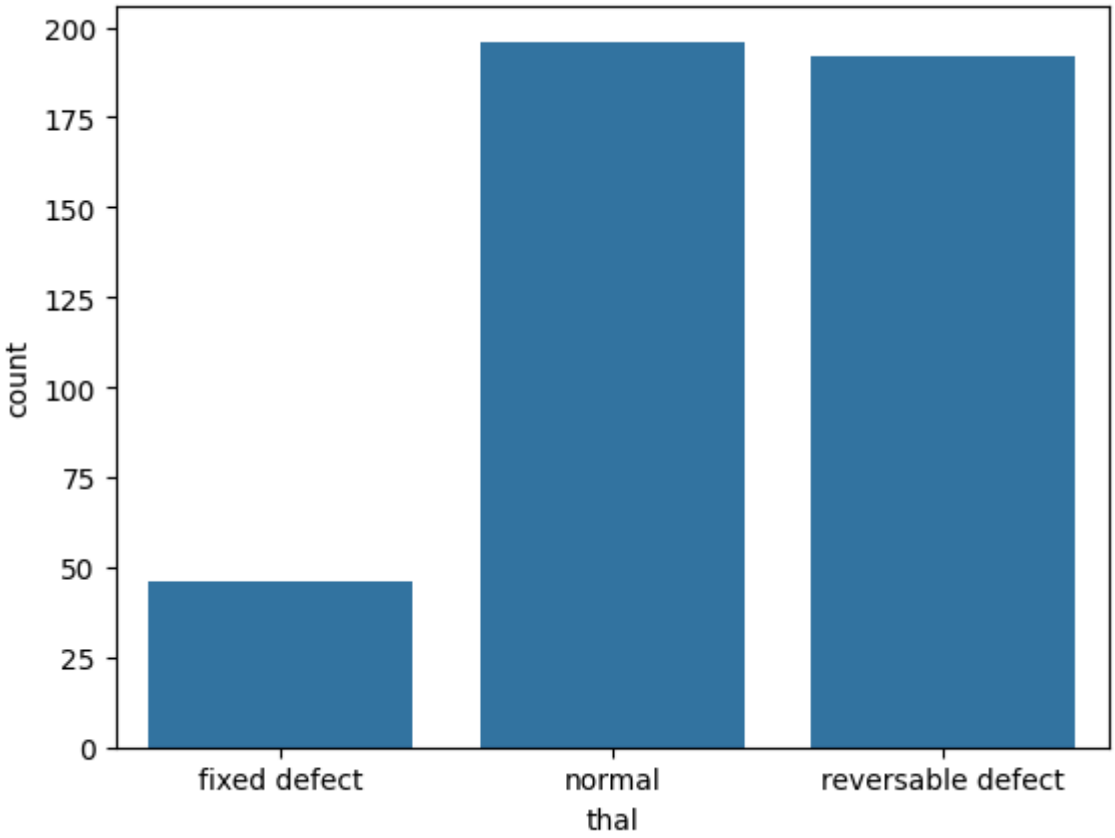
Out[76]:
```
486
```

In [77]:
```
df['thal'].unique()
```

Out[77]:
```
array(['fixed defect', 'normal', 'reversable defect', nan], dtype=object)
```

In [78]:
```
sns.countplot(data=df,x='thal')
```

Out[78]:
```
<Axes: xlabel='thal', ylabel='count'>
```



4.15. num

In [79]:
```
df['num'].dtype
```

Out[79]:
```
dtype('int64')
```

In [80]:
```
df['num'].isnull().sum()
```

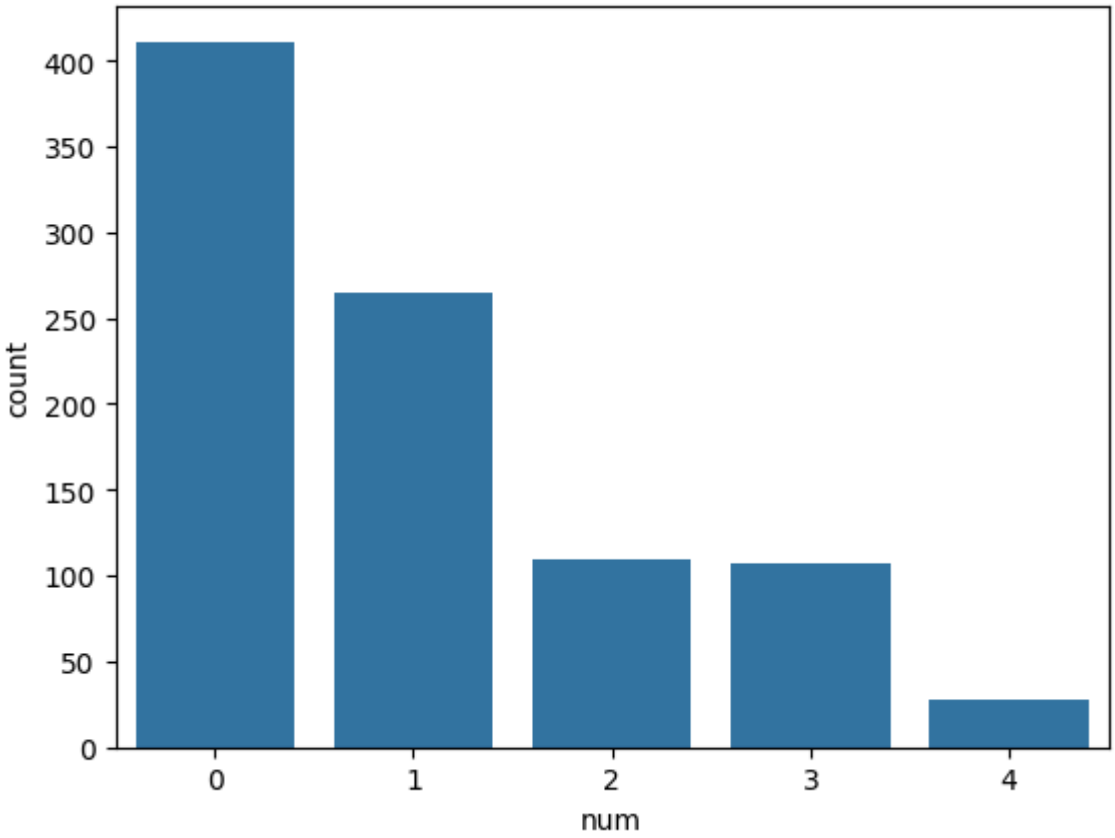
Out[80]:
```
0
```

In [81]:
```
df['num'].unique()
```

Out[81]:
```
array([0, 2, 1, 3, 4], dtype=int64)
```

In [82]:
```
sns.countplot(data=df,x='num')
```

Out[82]:
```
<Axes: xlabel='num', ylabel='count'>
```



Observations:

- Age is not normal, the best practice should be normalization of age.
- Men have 3.74 times more chances of heart disease as compared to women.
- Most of the dataset is from Cleveland while smallest part of dataset belongs to Switzerland.
- Highest number of chest pains are asymptotic(53.91%) while rare one is typical angina(5.0%).
- Trestbps is rightly skewed.
- 0 trestbps is considerable.
- As their are 172 number of people having chol at 0, hence it can't be an outlier.
- People with fasting blood sugar > 120 mg/dl are 5.01 times more in number as compared to people with fasting blood sugar < 120 mg/dl.
- `59.89%` patients have normal restecg.
- `36.63%` had angina induced during exercise.
- From not null ca values, `58.57%` had no vessel effected while `6.47%` had all the three effected.
- Count of people recieving blood to specific tissues of heart during rest but not stress, is too high.

Chapter 5: Data Cleaning

5.1 Dealing with Missing Values

In [83]:
```
# imputing missing values that are less than 10%

missing_threshold = 10  # 10% threshold for missing values
total_entries = len(df)
target_variable = 'num'

# list of numerical columns that have less than a certain threshold of missing values.
numerical_cols = []
for col in df.columns:
```

```
    if (df[col].dtype in ['int64', 'float64'] and
        (df[col].isnull().sum() / total_entries * 100) < missing_threshold and
        col != target_variable):
        numerical_cols.append(col)


# list of numerical columns that have less than a certain threshold of missing values
categorical_cols = []
for col in df.columns:
    if (df[col].dtype == 'object' and
        (df[col].isnull().sum() / total_entries * 100) < missing_threshold):
        categorical_cols.append(col)


# Apply simple imputation to numerical and categorical columns
num_imputer = SimpleImputer(strategy='median')
cat_imputer = SimpleImputer(strategy='most_frequent')

# Apply imputation to numerical and categorical columns
for col in numerical_cols:
    df.loc[:,col] = num_imputer.fit_transform(df[[col]])

for col in categorical_cols:
    df.loc[:, col] = cat_imputer.fit_transform(df[[col]])
# Display the features with missing values
print(f"The sum of all high missing values where simple imputation wasn't used, is:\n{df[['th
```

```
The sum of all high missing values where simple imputation wasn't used, is:
thal     486
ca       611
slope    309
dtype: int64
```

In [84]:

```
# Confirm there are no missing values left in features with less than 10% missing values

missing_values_sum = df[df.columns.difference(['thal', 'ca', 'slope'])].isnull().sum()

print(f"The sum of all missing values, excluding features ['thal', 'ca', 'slope'], is:\n{miss
```

```
The sum of all missing values, excluding features ['thal', 'ca', 'slope'], is:
age         0
chol        0
cp          0
dataset     0
exang       0
fbs         0
id          0
num         0
oldpeak     0
restecg     0
sex         0
thalch      0
trestbps    0
dtype: int64
```

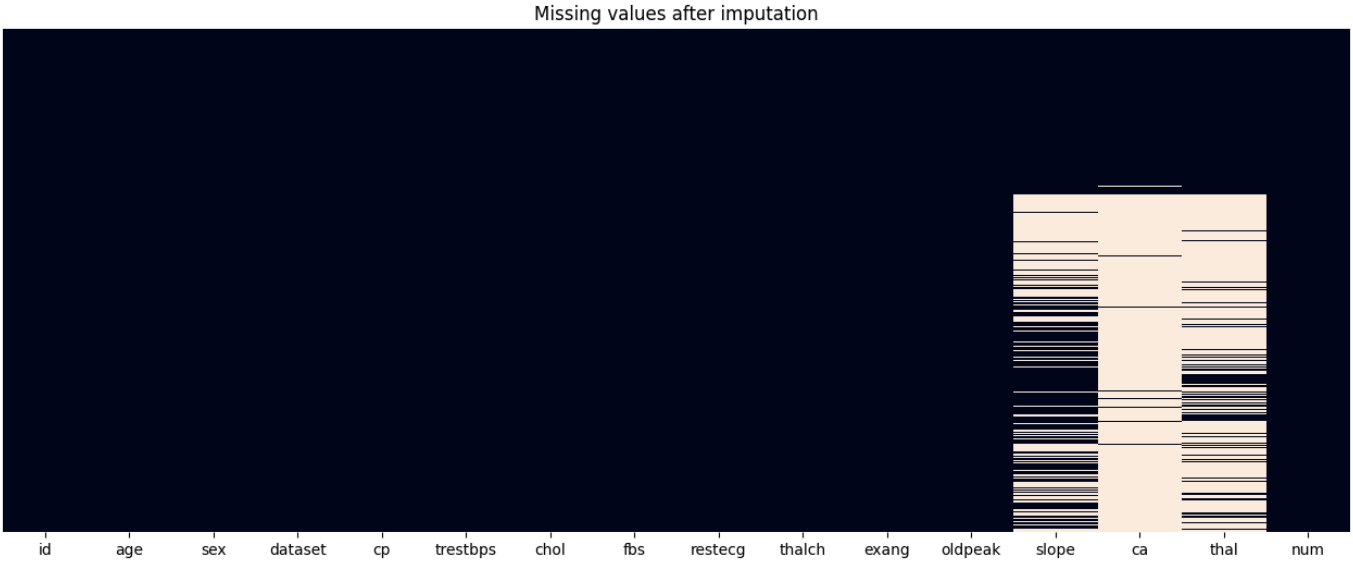
In [85]:

```
df.isnull().sum().sort_values(ascending=False).head()
```

Out[85]:

```
ca       611
thal     486
slope    309
id         0
age        0
dtype: int64
```

In [86]:
```
# Let's see heatmap to check missing values after imputation
plt.figure(figsize=(16, 6))
sns.heatmap(df.isnull(), cbar=False, yticklabels = False)
plt.title('Missing values after imputation')
plt.show()
```



In [87]:
```
# Split the dataset into two subset with one contaning complete data and the other containing
# Create a new dataframe which removes all the rows that nan values
df_clean = df.dropna()

# Create a new dataframe which contains all the rows that have missing values
df_missing = df[df.isna().any(axis=1)]

print(f"The shape of the complete dataframe is: {df_clean.shape}")
print("\n")
print(f"The shape of the dataframe with missing values is: {df_missing.shape}")
```

```
The shape of the complete dataframe is: (299, 16)


The shape of the dataframe with missing values is: (621, 16)
```

In [88]:
```
# Training and evaluating the model for the target variables 'slope', 'thanl', and 'ca'.
def predict_and_evaluate(target_variable, drop_columns):
    # Drop specified columns from the DataFrame
    X = df_clean.drop(drop_columns, axis=1)

    # Select target variable(s)
    y = df_clean[target_variable]

    # Initialize LabelEncoder
    le = LabelEncoder()
    for col in X.columns:
        if X[col].dtype == 'object':
            X[col] = le.fit_transform(X[col])

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize RandomForestClassifier
    model = RandomForestClassifier(n_estimators=10, random_state=42)

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)
```

```
    # Print accuracy score
    print(f"The accuracy of the model is: {round(accuracy_score(y_test, y_pred),2)}")

    # Print classification report
    print(f"Classification report:\n {classification_report(y_test, y_pred)}")

    # Return the trained model
    return model
```

In [89]:
```
# Train and evaluate a model for predicting the 'slope' feature, excluding 'slope', 'ca', and
rf_model_slope  = predict_and_evaluate('slope', ['slope', 'ca', 'thal'])
```

```
The accuracy of the model is: 0.72
Classification report:
               precision    recall  f1-score   support

 downsloping       0.00      0.00      0.00         3
        flat       0.70      0.76      0.73        25
   upsloping       0.75      0.75      0.75        32

    accuracy                           0.72        60
   macro avg       0.48      0.50      0.49        60
weighted avg       0.69      0.72      0.70        60
```

In [90]:
```
# Train and evaluate a model for predicting the 'ca' feature, excluding 'ca', and 'thal' from
rf_model_ca = predict_and_evaluate('ca', [ 'ca', 'thal'])
```

```
The accuracy of the model is: 0.72
Classification report:
               precision    recall  f1-score   support

         0.0       0.76      0.97      0.86        40
         1.0       0.67      0.15      0.25        13
         2.0       0.33      0.40      0.36         5
         3.0       0.00      0.00      0.00         2

    accuracy                           0.72        60
   macro avg       0.44      0.38      0.37        60
weighted avg       0.68      0.72      0.66        60

Classification report:
               precision    recall  f1-score   support

         0.0       0.76      0.97      0.86        40
         1.0       0.67      0.15      0.25        13
         2.0       0.33      0.40      0.36         5
         3.0       0.00      0.00      0.00         2

    accuracy                           0.72        60
   macro avg       0.44      0.38      0.37        60
weighted avg       0.68      0.72      0.66        60
```

In [91]:
```
# Train and evaluate a model for predicting the 'thal' feature.
rf_model_thal = predict_and_evaluate('thal', ['thal'])
```

```
The accuracy of the model is: 0.7
Classification report:
                    precision    recall  f1-score   support

     fixed defect       0.00      0.00      0.00         4
           normal       0.67      0.94      0.78        31
reversable defect       0.76      0.52      0.62        25

         accuracy                           0.70        60
        macro avg       0.48      0.49      0.47        60
     weighted avg       0.67      0.70      0.66        60
```

In [92]:
```
# create a new dataframe and drop the target variables
df_encoded_pred = df_missing.drop(['slope', 'ca', 'thal'], axis=1)

# Encode categorical features and predict the missing values
def encode_and_predict(df, target_variable, model):

    # Initialize label encoder
    le = LabelEncoder()

    for col in df.columns:
        if df[col].dtype == 'object':
            df[col] = le.fit_transform(df[col])
    df[target_variable] = model.predict(df)

# Encode categorical features and predict the 'slope' column in df_encoded_pred
encode_and_predict(df_encoded_pred, 'slope', rf_model_slope)
# Update the 'slope' column in df_missing with the predicted values
df_missing.loc[:,'slope'] = df_encoded_pred['slope']
```

In [93]:
```
# Create a new dataframe and drop the target variables
df_encoded_pred = df_missing.drop(['ca', 'thal'], axis=1)
# Encode categorical features and predict the 'ca' column in df_encoded_pred
encode_and_predict(df_encoded_pred, 'ca', rf_model_ca)
# Update the 'ca' column in df_missing with the predicted values
df_missing.loc[:, 'ca'] = df_encoded_pred['ca']
```

In [94]:
```
# Create a new dataframe and drop the target variables
df_encoded_pred = df_missing.drop(['thal'], axis=1)
# Encode categorical features and predict the 'thal' column in df_encoded_pred
encode_and_predict(df_encoded_pred, 'thal', rf_model_thal)
# Update the 'thal' column in df_missing with the predicted values
df_missing.loc[:, 'thal'] = df_encoded_pred['thal']
```

In [95]:
```
# Merge df_clean and df_missing
df = pd.concat([df_clean, df_missing])

print(f"The shape of the dataset is: {df.shape[0]} rows and {df.shape[1]} columns.")
```

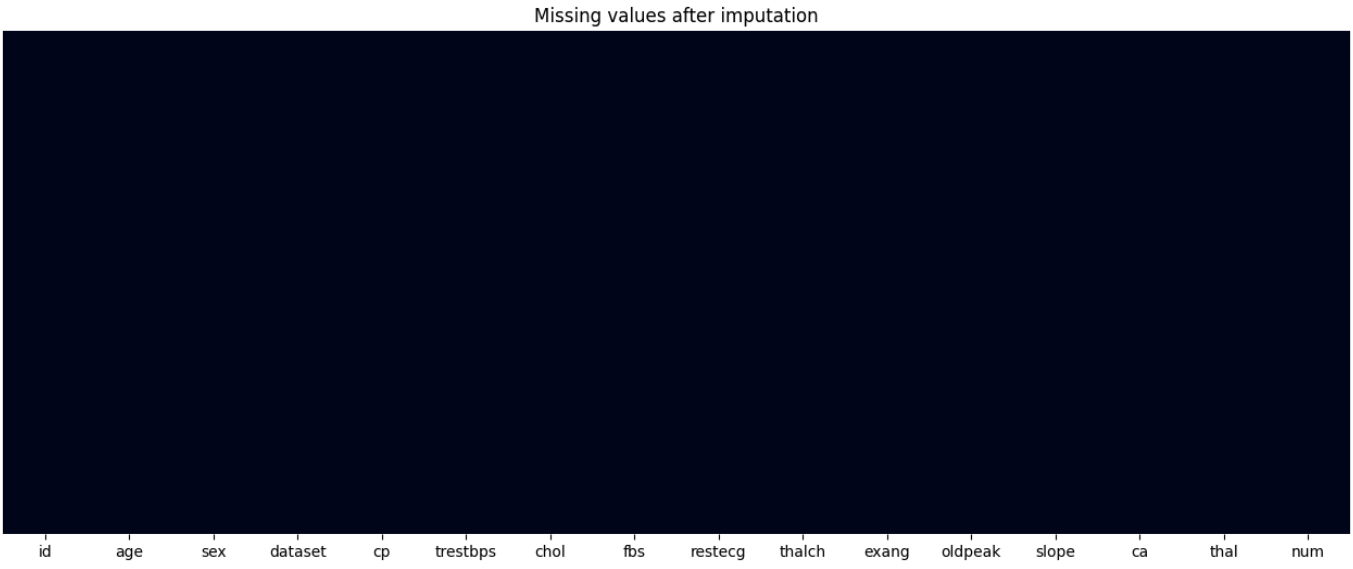
```
The shape of the dataset is: 920 rows and 16 columns.
```

In [96]:
```
# Display the missing values if any in the dataset for verification
if df.isnull().sum().sum() == 0:
    print("There are no missing values in the dataset.")
else:
    print("There are missing values in the dataset.")
```

```
There are no missing values in the dataset.
```

In [97]:
```
# Create a heatmap for df to check missing values
plt.figure(figsize=(16, 6))
sns.heatmap(df.isnull(), cbar=False, yticklabels = False)
```

```
plt.title('Missing values after imputation')
plt.show()
```



- Finally we got rid of missing values 🎉

Observations:

- For all the features with less than 10% missing values, we can impute them using mean, median, or mode.
- But for large missing values, we can't impute them with any of the above methods because it will cause biasness. So, we can impute them using Random Forest Classifier(because all three are categorical).

5.2 Outliers

trestbps was spotted as outlier, let's check it again and drop if it is actually.

In [98]:
```
# checking features with 0 trestbps
df[df['trestbps']==0]
```

Out[98]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope
753	754	55	Male	VA Long Beach	non-anginal	0.0	0.0	False	normal	155.0	False	1.5	flat

In [99]:
```
df.drop(753, inplace=True,axis=0)
```

In [100…
```
# Check is it dropped
df[df['trestbps']==0]
```

Out[100…

id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	nu

Observation:

- It has 0 resting bps and num=3. It means patien may had severe heart attack or died.
- So, I have decided to drop this feature from the dataset.

Chapter 6: Bivariate and Multivariate analysis

- As we have num as target variable,so we will check bivariate and multivariate analysis only with num.

In [101…

```
df.head
```

Out[101…

```
<bound method NDFrame.head of       id  age     sex        dataset               cp  trestbps
chol    fbs  \
0      1   63    Male      Cleveland   typical angina     145.0  233.0   True
1      2   67    Male      Cleveland     asymptomatic     160.0  286.0  False
2      3   67    Male      Cleveland     asymptomatic     120.0  229.0  False
3      4   37    Male      Cleveland      non-anginal     130.0  250.0  False
4      5   41  Female      Cleveland  atypical angina     130.0  204.0  False
..   ...  ...     ...            ...              ...       ...    ...    ...
915  916   54  Female  VA Long Beach     asymptomatic     127.0  333.0   True
916  917   62    Male  VA Long Beach   typical angina     130.0  139.0  False
917  918   55    Male  VA Long Beach     asymptomatic     122.0  223.0   True
918  919   58    Male  VA Long Beach     asymptomatic     130.0  385.0   True
919  920   62    Male  VA Long Beach  atypical angina     120.0  254.0  False

              restecg  thalch  exang  oldpeak        slope   ca  \
0      lv hypertrophy   150.0  False      2.3  downsloping  0.0
1      lv hypertrophy   108.0   True      1.5         flat  3.0
2      lv hypertrophy   129.0   True      2.6         flat  2.0
3              normal   187.0  False      3.5  downsloping  0.0
4      lv hypertrophy   172.0  False      1.4    upsloping  0.0
..                ...     ...    ...      ...          ...  ...
915  st-t abnormality   154.0  False      0.0         flat  0.0
916  st-t abnormality   140.0  False      0.5         flat  0.0
917  st-t abnormality   100.0  False      0.0         flat  1.0
918    lv hypertrophy   140.0  False      0.5         flat  0.0
919    lv hypertrophy    93.0   True      0.0         flat  1.0

                  thal  num
0         fixed defect    0
1               normal    2
2    reversable defect    1
3               normal    0
4               normal    0
..                 ...  ...
915             normal    1
916  reversable defect    0
917             normal    2
918             normal    0
919             normal    1

[919 rows x 16 columns]>
```

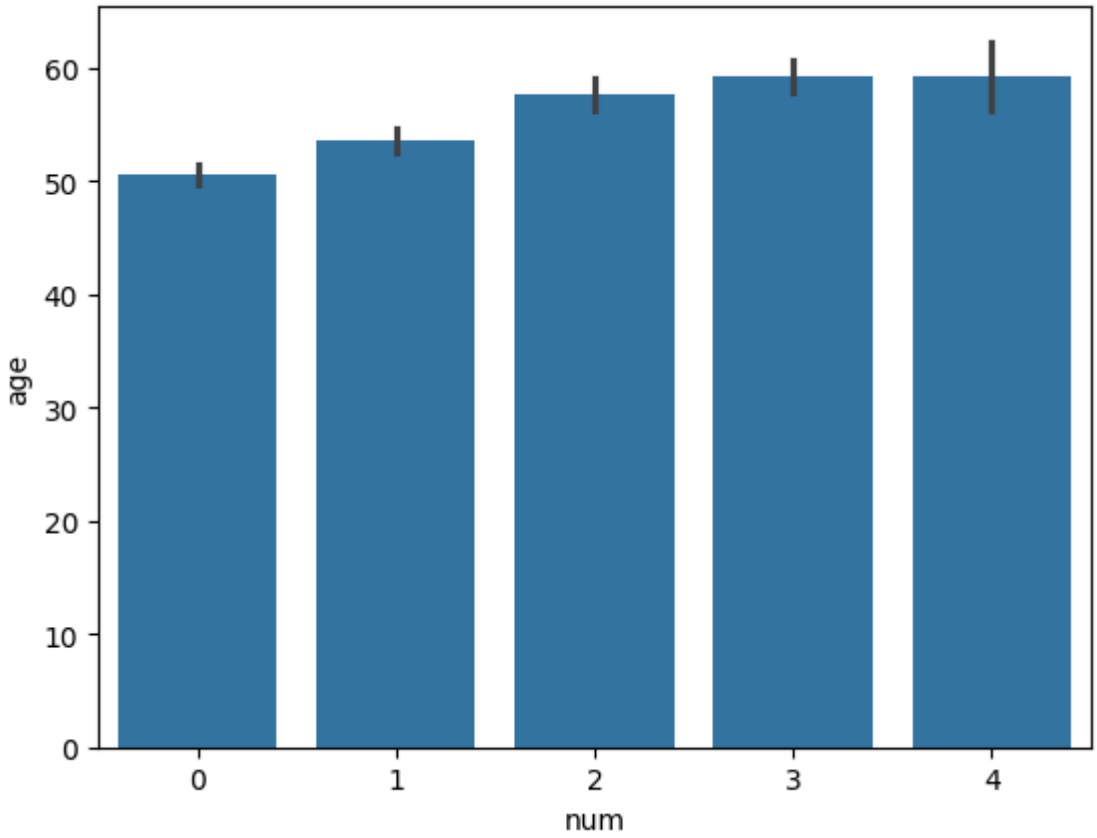
6.1 num vs age

In [102…

```
sns.barplot(x='num',y='age',data=df)
```

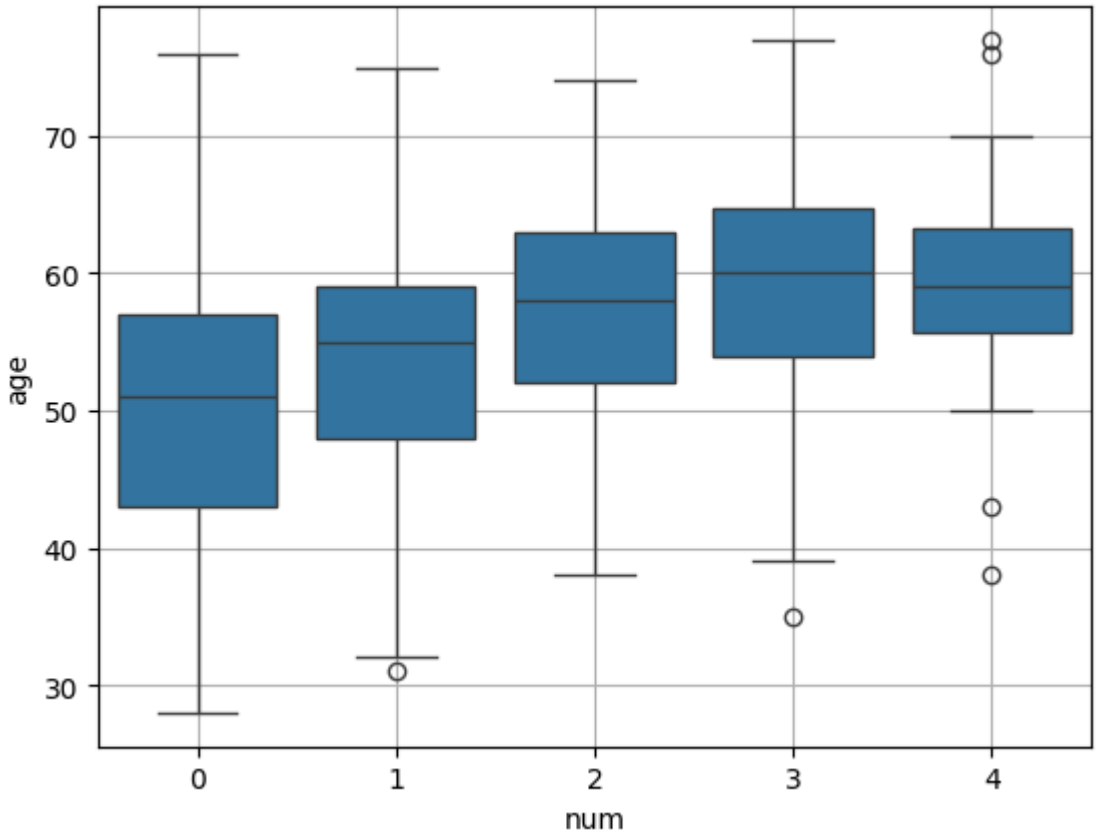
Out[102…

```
<Axes: xlabel='num', ylabel='age'>
```



In [103…

```
sns.boxplot(x='num',y='age',data=df)
plt.grid()
```



In [104…

```
df.groupby('num')['age'].describe()
```

Out[104…

num	count	mean	std	min	25%	50%	75%	max
0	411.0	50.547445	9.433700	28.0	43.00	51.0	57.00	76.0
1	265.0	53.528302	8.740371	31.0	48.00	55.0	59.00	75.0
2	109.0	57.577982	7.786852	38.0	52.00	58.0	63.00	74.0
3	106.0	59.254717	8.017910	35.0	54.00	60.0	64.75	77.0
4	28.0	59.214286	8.283661	38.0	55.75	59.0	63.25	77.0

6.2. num vs sex

In [105…

```
# making contigency table
cotigency_table=pd.crosstab(df['num'],df['sex'])
cotigency_table
```

Out[105…

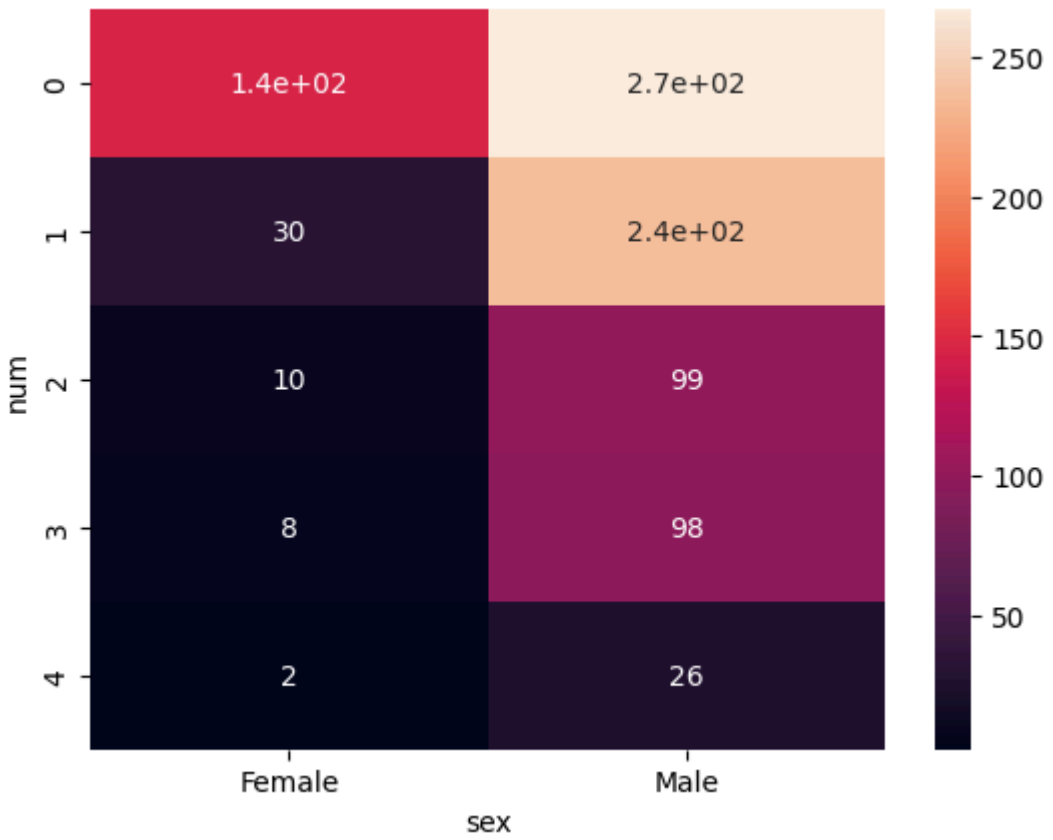
num	sex Female	Male
0	144	267
1	30	235
2	10	99
3	8	98
4	2	26

In [106…

```
sns.heatmap(cotigency_table,annot=True)
```

Out[106…

```
<Axes: xlabel='sex', ylabel='num'>
```



In [107…

```
# chi-squared contigency test
# H0: There is no association between the variables
# H1: there is association between the variables
stat,p,dof,expected=chi2_contingency(cotigency_table)
if p>0.05:
    print('there is no association.')
else:
    print('Both have association.')
print(f'p value is: {p}')
```

```
Both have association.
p value is: 4.63523526517916e-18
```

6.3. num vs dataset

In [108…

```
# making contigency table
cotigency_table=pd.crosstab(df['num'],df['dataset'])
cotigency_table
```

Out[108…

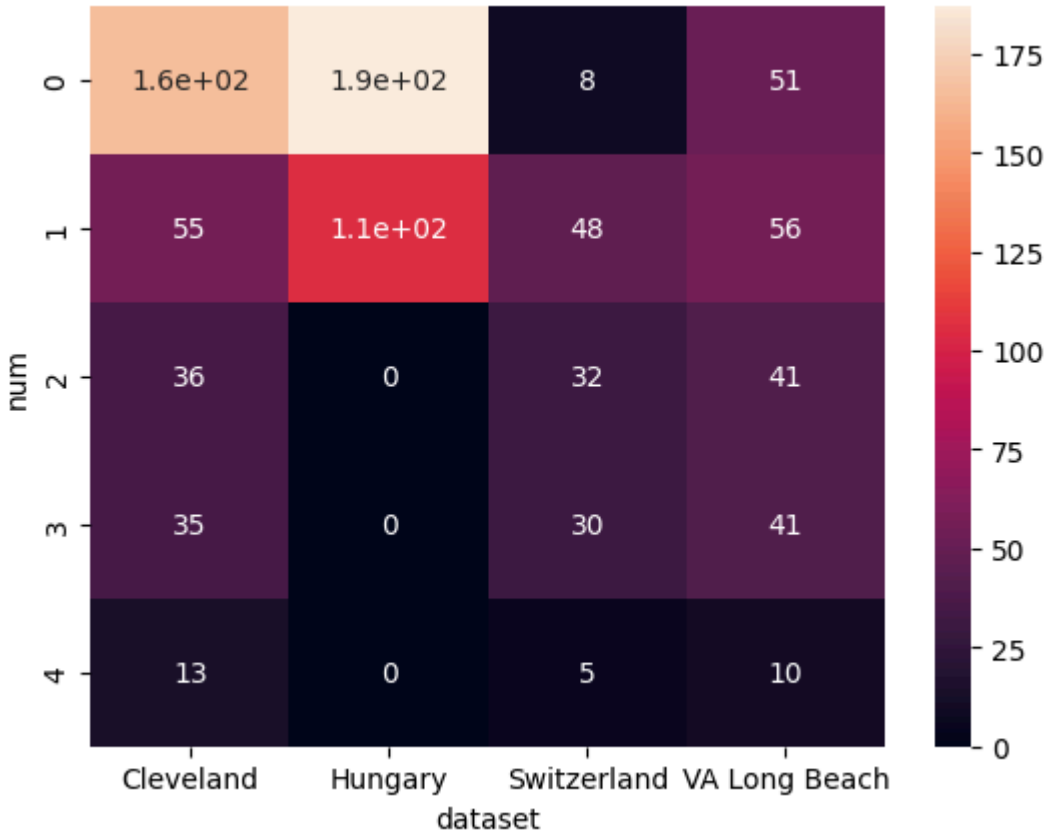
dataset	Cleveland	Hungary	Switzerland	VA Long Beach
num				
0	165	187	8	51
1	55	106	48	56
2	36	0	32	41
3	35	0	30	41
4	13	0	5	10

In [109…

```
sns.heatmap(cotigency_table,annot=True)
```

Out[109…

```
<Axes: xlabel='dataset', ylabel='num'>
```



In [110…

```
# chi-squared contigency test
# H0: There is no association between the variables
```

```
# H1: there is association between the variables
stat,p,dof,expected=chi2_contingency(cotigency_table)
if p>0.05:
    print('there is no association.')
else:
    print('Both have association.')
print(f'p value is: {p}.')
```

```
Both have association.
p value is: 7.727220653424762e-48.
```

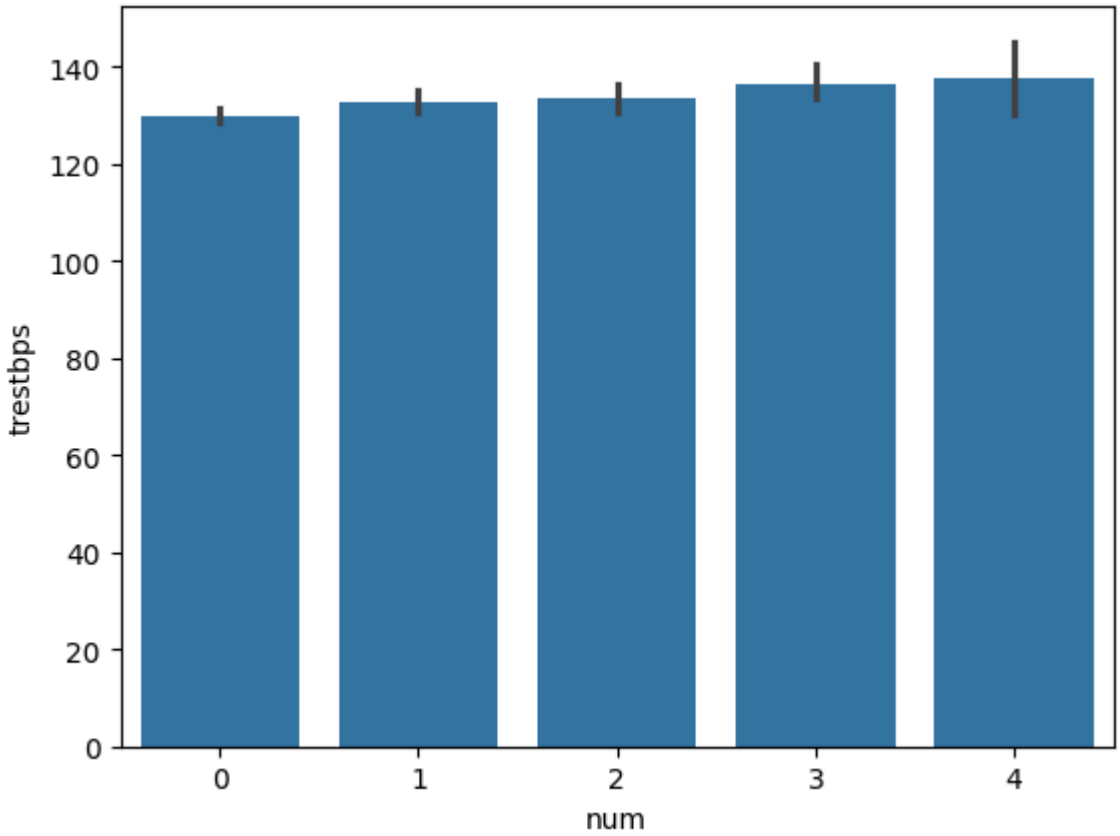
6.4. num vs trestbps

In [111…

```
sns.barplot(x='num',y='trestbps',data=df)
```

Out[111…

```
<Axes: xlabel='num', ylabel='trestbps'>
```

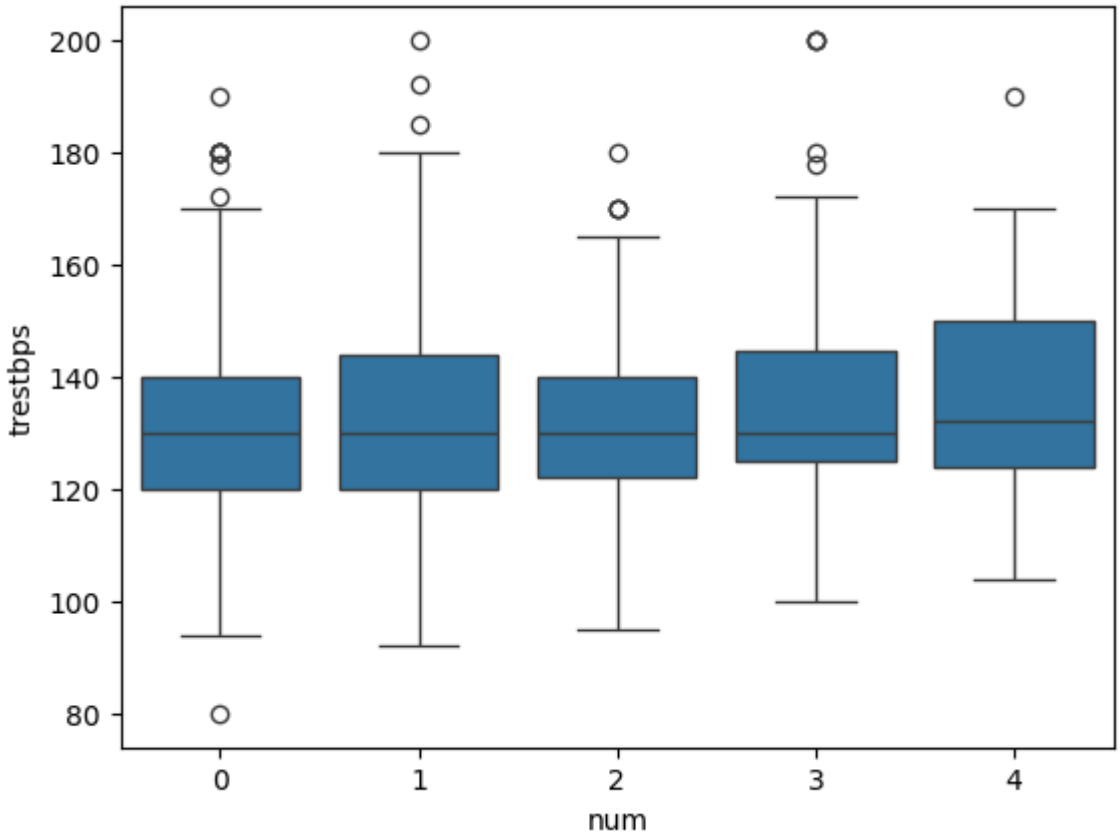


In [112…

```
sns.boxplot(x='num',y='trestbps',data=df)
```

Out[112…

```
<Axes: xlabel='num', ylabel='trestbps'>
```

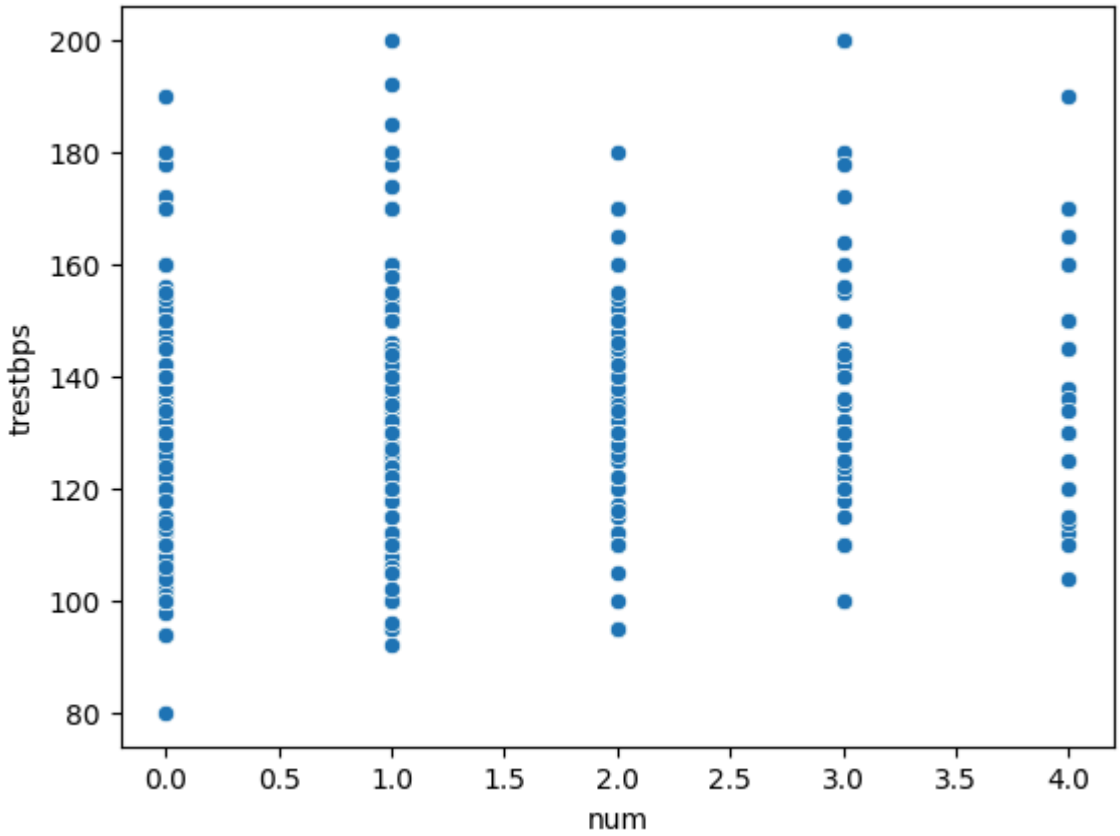


In [113…

```
sns.scatterplot(x='num',y='trestbps',data=df)
```

Out[113…

```
<Axes: xlabel='num', ylabel='trestbps'>
```



In [114…

```
df.groupby('num')['trestbps'].describe()
```

Out[114…

num	count	mean	std	min	25%	50%	75%	max
0	411.0	129.917275	16.453274	80.0	120.00	130.0	140.00	190.0
1	265.0	132.720755	19.363482	92.0	120.00	130.0	144.00	200.0
2	109.0	133.348624	16.669098	95.0	122.00	130.0	140.00	180.0
3	106.0	136.566038	19.067309	100.0	125.00	130.0	144.75	200.0
4	28.0	137.785714	21.103549	104.0	123.75	132.0	150.00	190.0

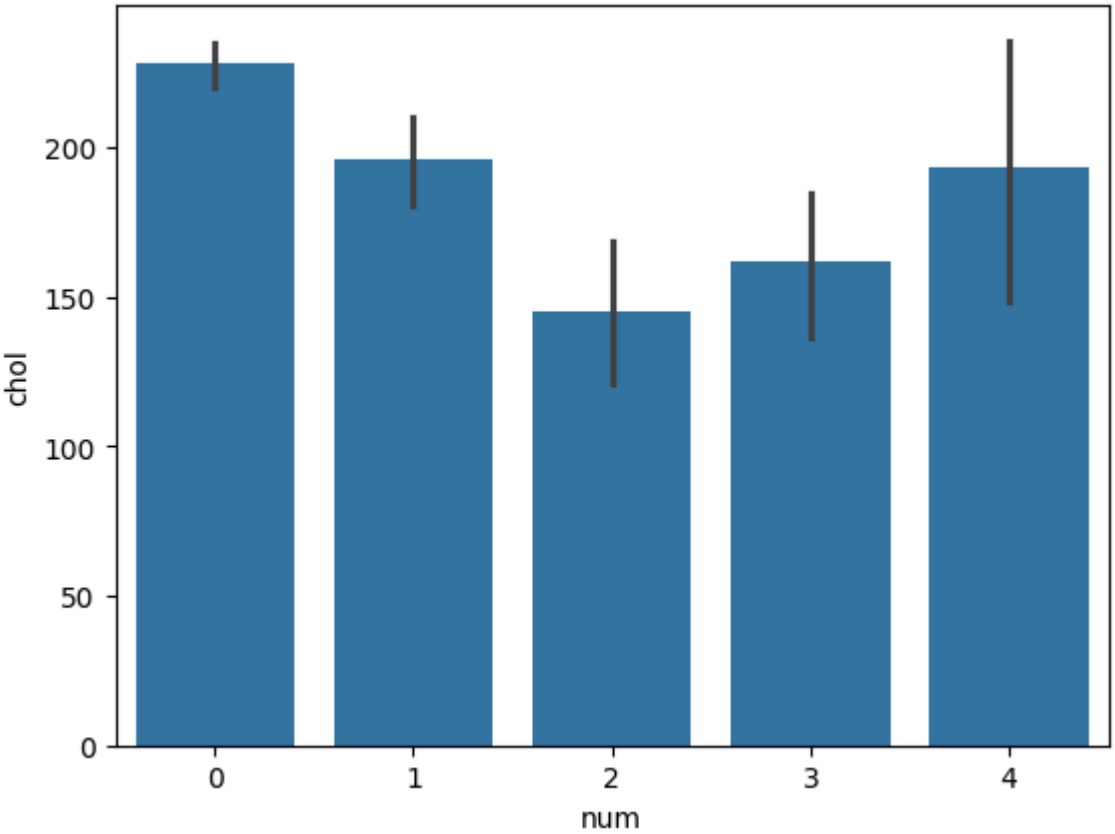
6.5. num vs chol

In [115…

```
sns.barplot(x='num',y='chol',data=df)
```

Out[115…

```
<Axes: xlabel='num', ylabel='chol'>
```

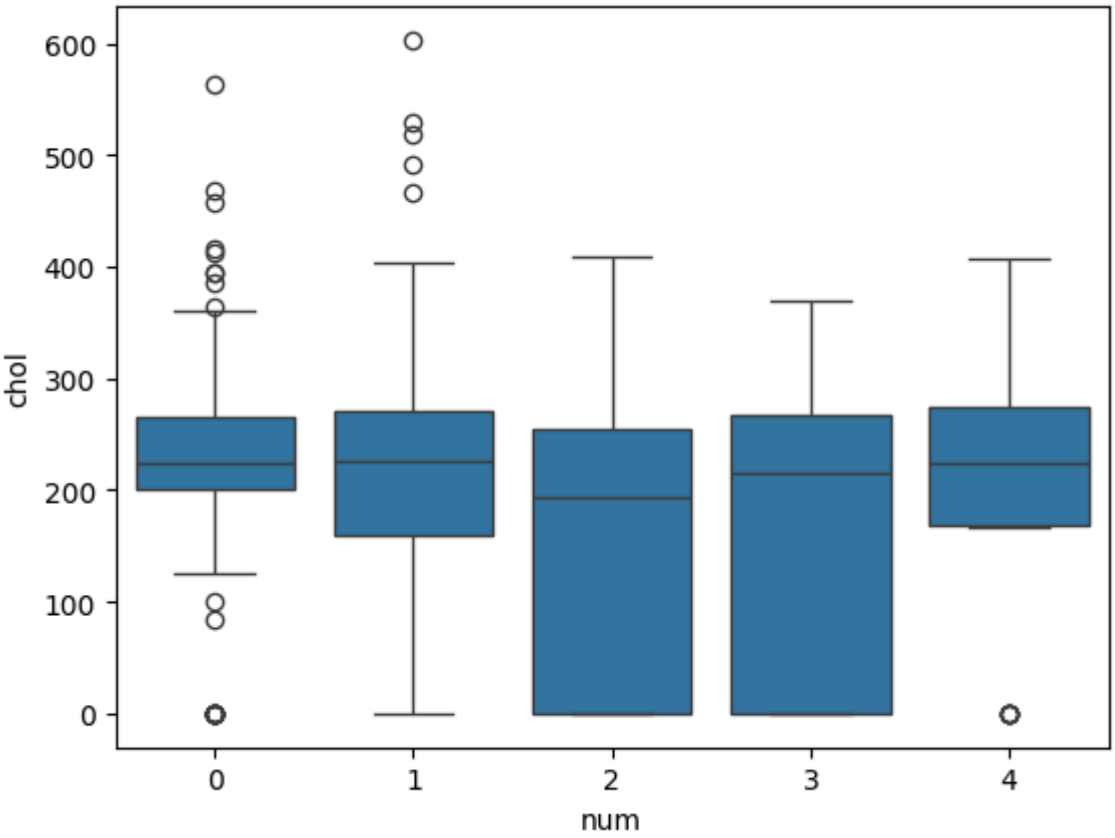


In [116…

```
sns.boxplot(x='num',y='chol',data=df)
```

Out[116…

```
<Axes: xlabel='num', ylabel='chol'>
```



6.6. num vs fbs

In [117…

```
# making contigency table
cotigency_table=pd.crosstab(df['num'],df['fbs'])
cotigency_table
```

Out[117…

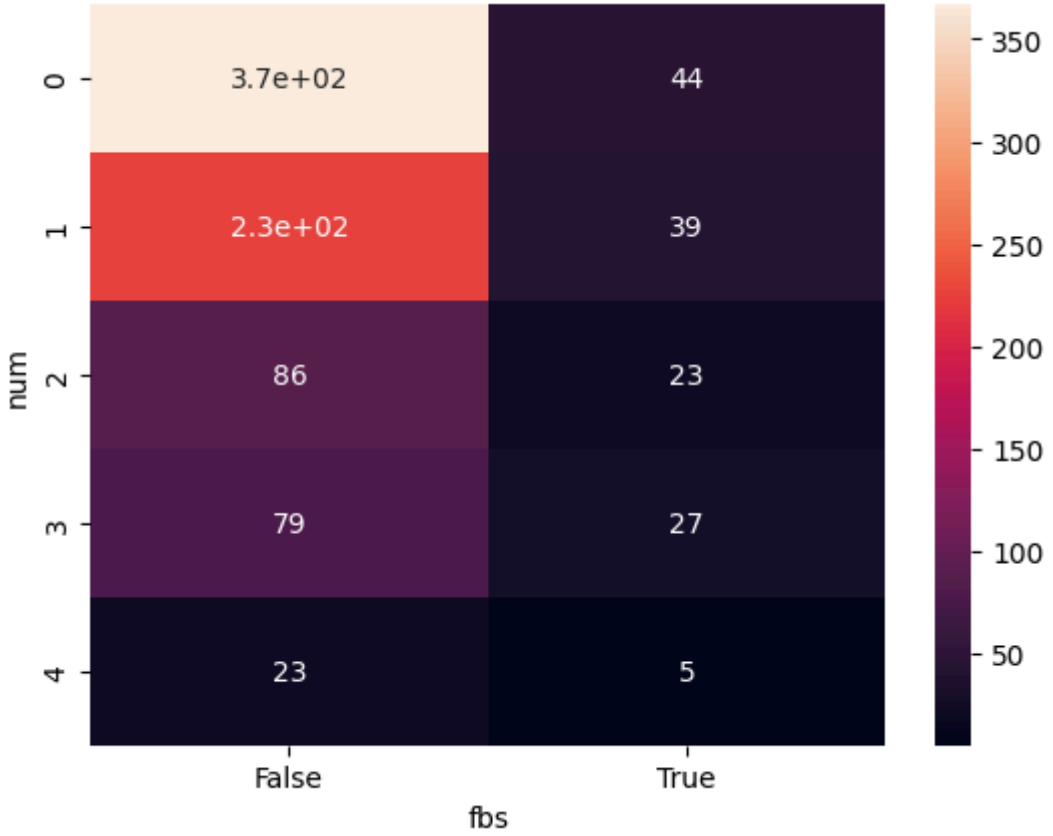
fbs	False	True
num		
0	367	44
1	226	39
2	86	23
3	79	27
4	23	5

In [118…

```
sns.heatmap(cotigency_table,annot=True)
```

Out[118…

```
<Axes: xlabel='fbs', ylabel='num'>
```



In [119…

```
# chi-squared contigency test
# H0: There is no association between the variables
# H1: there is association between the variables
stat,p,dof,expected=chi2_contingency(cotigency_table)
if p>0.05:
    print('there is no association.')
else:
    print('Both have association.')
print(f'p value is: {p}')
```

```
Both have association.
p value is: 0.0010201461964449738
```

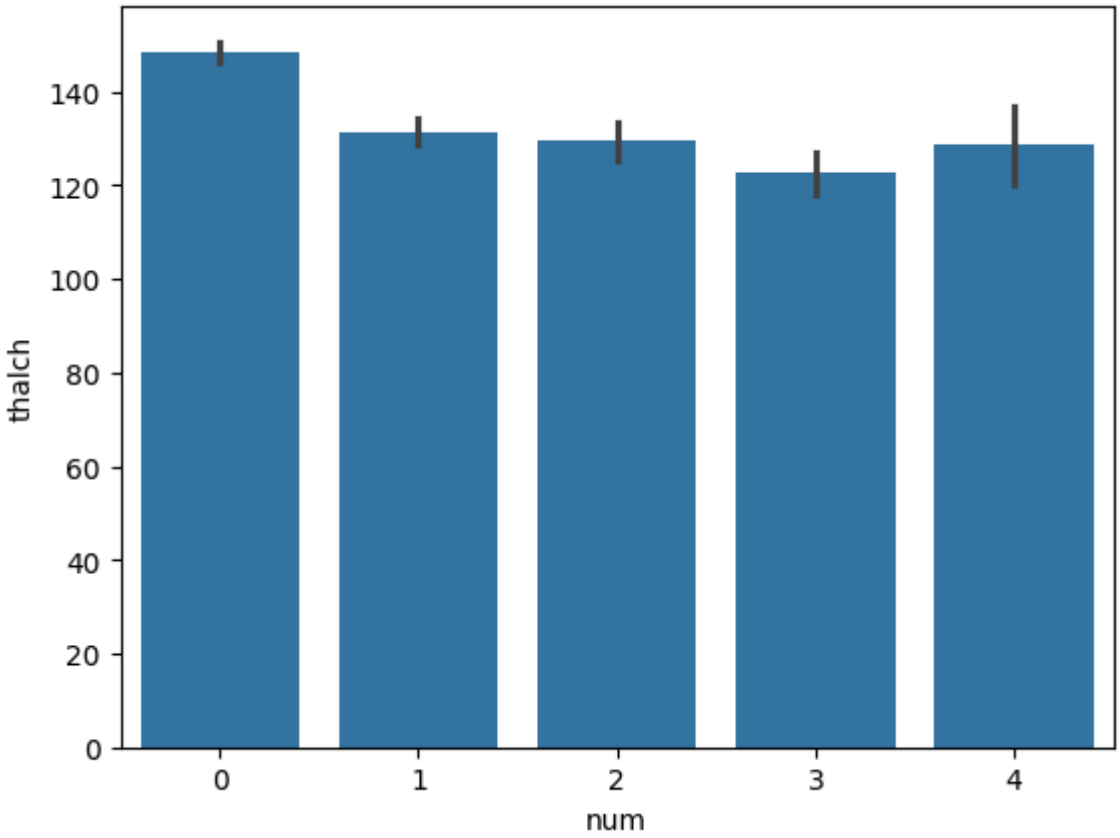
6.7. num vs thalch

In [120…

```
sns.barplot(x='num',y='thalch',data=df)
```

Out[120…

```
<Axes: xlabel='num', ylabel='thalch'>
```

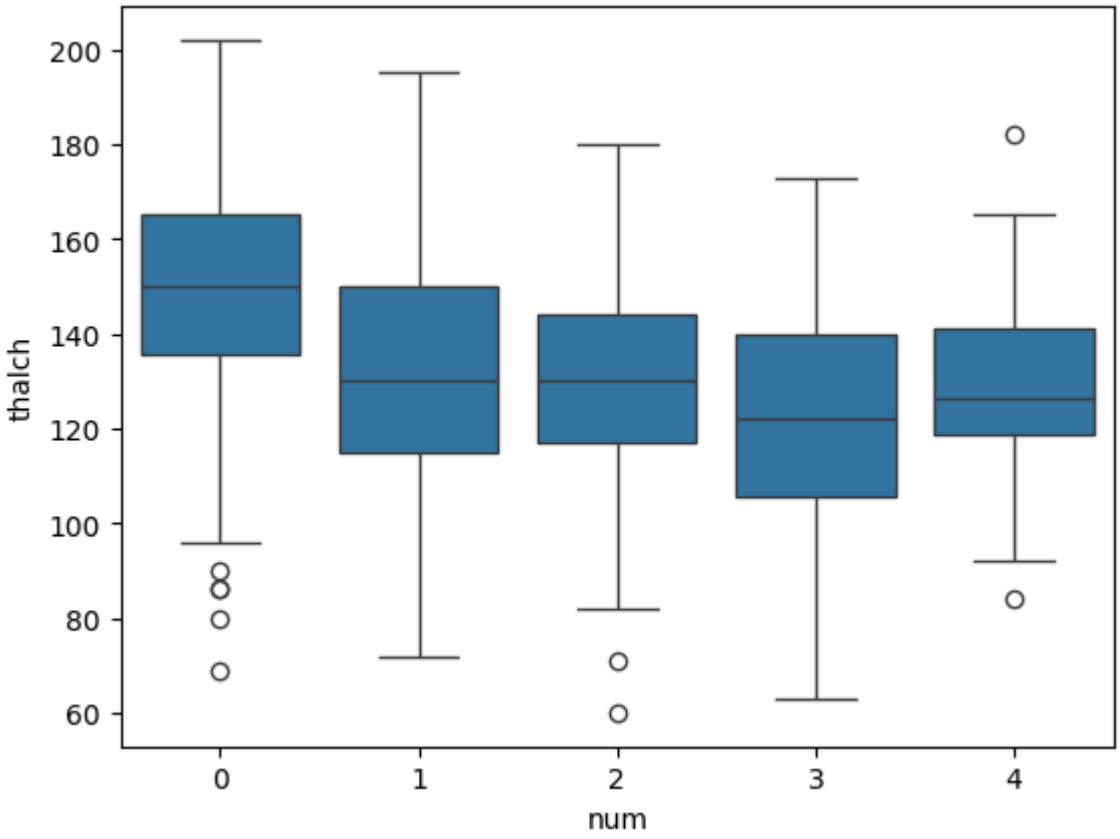


In [121…

```
sns.boxplot(x='num',y='thalch',data=df)
```

Out[121…

```
<Axes: xlabel='num', ylabel='thalch'>
```



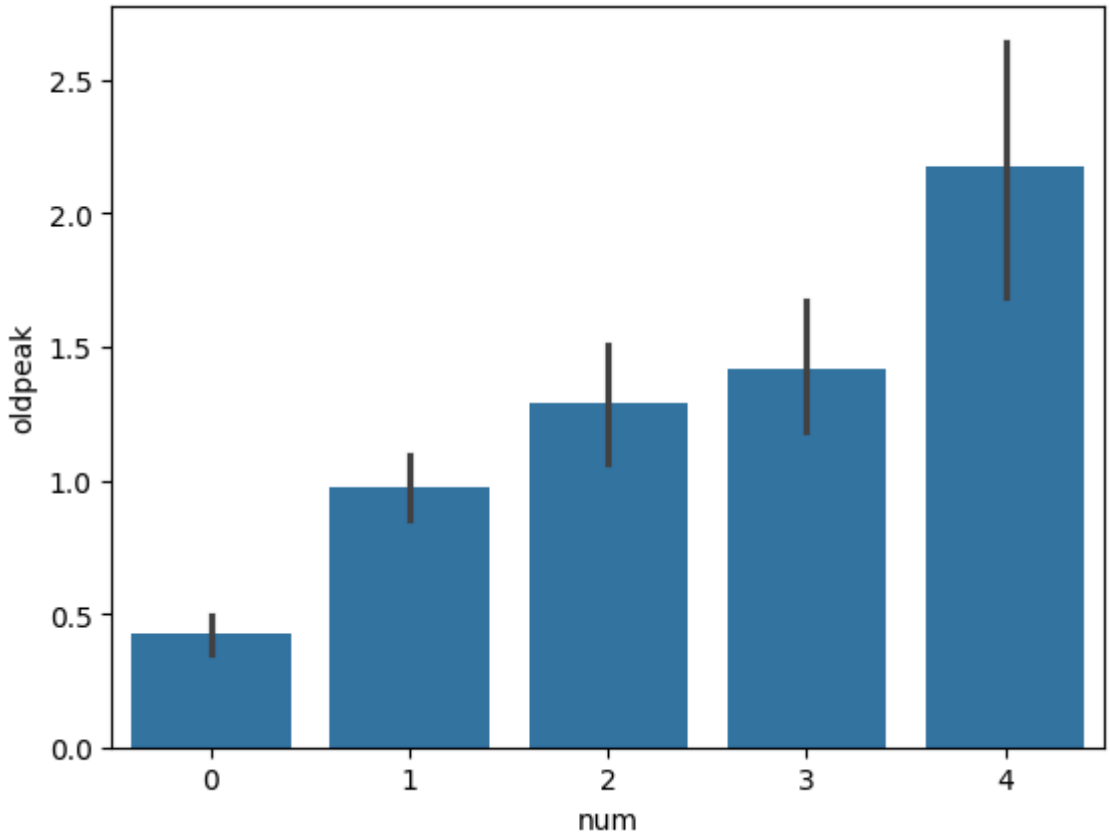
6.8. num vs oldpeak

In [122…

```
sns.barplot(x='num',y='oldpeak',data=df)
```

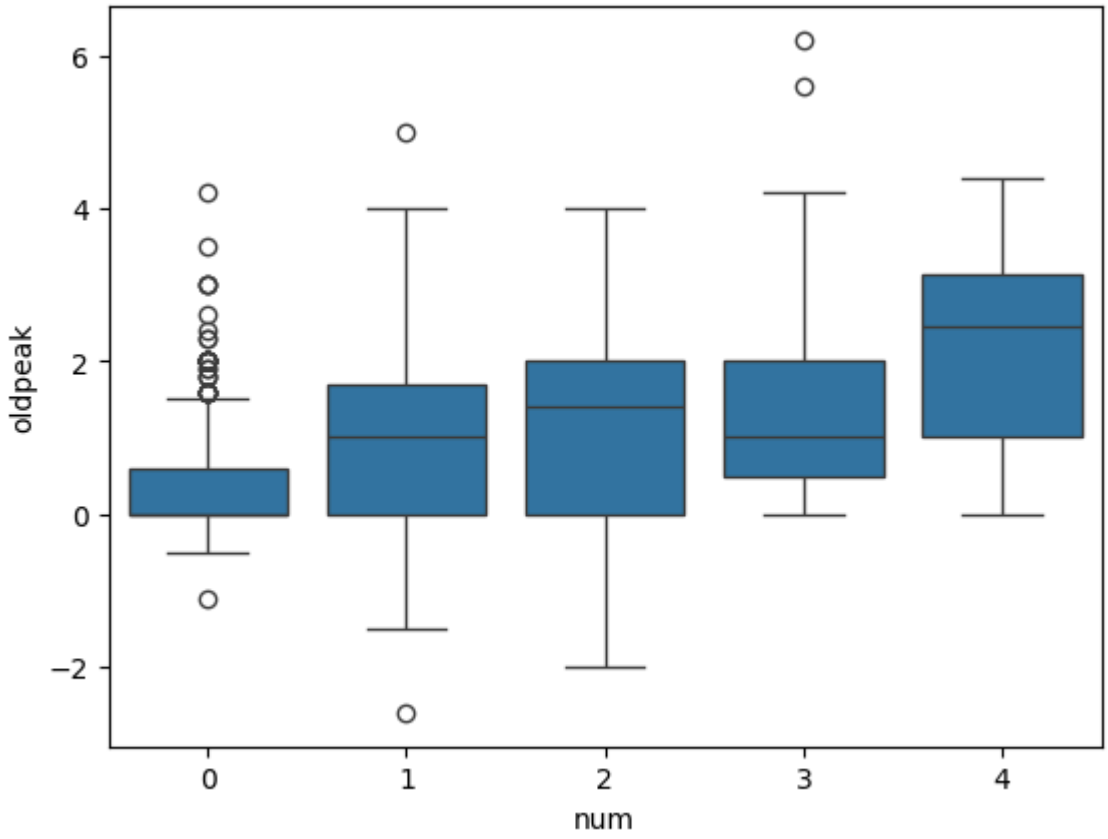
Out[122…

```
<Axes: xlabel='num', ylabel='oldpeak'>
```



In [123…
```
sns.boxplot(x='num',y='oldpeak',data=df)
```

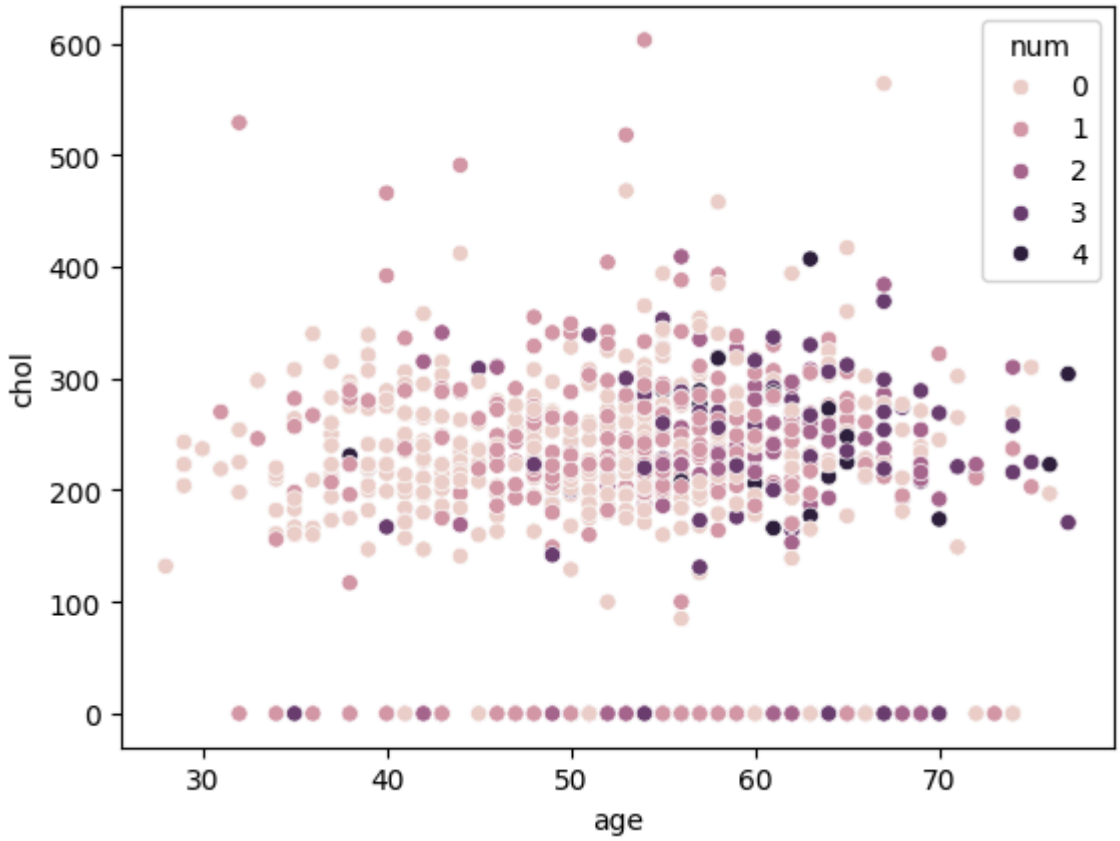
Out[123…
```
<Axes: xlabel='num', ylabel='oldpeak'>
```



6.9. age vs chol vs num

In [124…
```
sns.scatterplot(data=df,x='age',y='chol',hue='num')
```

Out[124…
```
<Axes: xlabel='age', ylabel='chol'>
```



In [125…

```
df[['age','chol']].corr()
```

Out[125…

	age	chol
age	1.000000	-0.085853
chol	-0.085853	1.000000

Observation:

- Generally chances of severity of heart disease increase with age.
- Second and third level of severity may have be because of deficiency of chol.
- Patients with no chest pain are more likely to have a heart disease.
- increase of oldpeak is also a justification for increasing severity of heart disease.
- no any relaionship was found between age and chol.

Chapter 7: Label Encoding

In [126…

```
# Dictionary to store the LabelEncoders
label_encoders = {}

# Iterate over the columns of the DataFrame
for col in df.columns:
    if df[col].dtype == 'object' or df[col].dtype == 'category':
        lab_enc = LabelEncoder()
        df[col] = lab_enc.fit_transform(df[col])
        label_encoders[col] = lab_enc
```

Chapter 08: Scalling numeric features

In [127…
```
# separating numeric features
numeric_cols = ['oldpeak', 'thalch', 'chol', 'trestbps', 'age']
```

In [128…
```
scalers={}
for i in numeric_cols:
    scaler=MinMaxScaler()
    df[i]=scaler.fit_transform(df[[i]])
    scalers[i]=scaler
scalers
```

Out[128…
```
{'oldpeak': MinMaxScaler(),
 'thalch': MinMaxScaler(),
 'chol': MinMaxScaler(),
 'trestbps': MinMaxScaler(),
 'age': MinMaxScaler()}
```

Chapter 09: ML Models Building

9.1. split into x and y

In [129…
```
# Let's split the data into Features (x) and Labels (y)
x = df.drop(['id', 'num'], axis = 1)
y = df['num']
```

9.2. split into train and test

In [130…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

9.3. Applying the functions to find best model:

In [131…
```
# creating function to find best model with best parameters:
def find_best_model(X, y, X_test, y_test):
    # Creating the global variables that we will use inside and the outside of the function as
    global y_pred,best_model_name
    best_model_name=None
    best_accuracy=0.0
    np.random.seed(42)
    models = {
        'LogisticRegression': (LogisticRegression(random_state=42), {}),

        'SVC': (SVC(random_state=42), {'kernel': ['rbf', 'poly', 'sigmoid'], 'C': [0.1, 1, 10

        'DecisionTreeClassifier': (DecisionTreeClassifier(random_state=42), {'max_depth': [No

        'RandomForestClassifier': (RandomForestClassifier(random_state=42), {'n_estimators':

        'KNeighborsClassifier': (KNeighborsClassifier(), {'n_neighbors': np.arange(3, 100, 2)
```

```
        'GradientBoostingClassifier': (GradientBoostingClassifier(random_state=42), {'n_estim

        'XGBClassifier': (XGBClassifier(random_state=42), {'n_estimators': [10, 100, 1000], '

        'AdaBoostClassifier': (AdaBoostClassifier(random_state=42), {'n_estimators': [50, 100
    }


    for name, (model, params) in models.items():
        try:
            grid_search = GridSearchCV(model, params, cv=5, scoring='accuracy')
            grid_search.fit(X, y)

            # You need to evaluate the model on the test set to get accuracy
            y_pred = grid_search.best_estimator_.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)

            # Print the performance metrics
            print("Model:", name)
            print("Cross-validation Accuracy:", grid_search.best_score_)
            print("Test Accuracy:", accuracy)
            print("\n_____________________________________________________________\n")

            # Check if the current model has the best accuracy
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_model_name = name
        except Exception as e:
            print(f"Error for model {name}: {e}")
    # Retrieve the best model
    print("Best Model:", best_model_name)
    print('Test Accuracy of the Best Model:', (best_accuracy)*100)
```

In [132…

```
%%time

# Calling function:
find_best_model(x, y, x_test, y_test)
```

```
Model: LogisticRegression
Cross-validation Accuracy: 0.5133998574483251
Test Accuracy: 0.6739130434782609

_____________________________________________________________

Model: SVC
Cross-validation Accuracy: 0.5526847232121644
Test Accuracy: 0.6413043478260869

_____________________________________________________________

Model: DecisionTreeClassifier
Cross-validation Accuracy: 0.47637205987170345
Test Accuracy: 0.9130434782608695

_____________________________________________________________

Model: RandomForestClassifier
Cross-validation Accuracy: 0.5298289379900214
Test Accuracy: 0.75

_____________________________________________________________

Model: KNeighborsClassifier
Cross-validation Accuracy: 0.5538132573057734
Test Accuracy: 0.625

_____________________________________________________________

Model: GradientBoostingClassifier
Cross-validation Accuracy: 0.46227726300784033
Test Accuracy: 0.7554347826086957

_____________________________________________________________

Model: XGBClassifier
Cross-validation Accuracy: 0.4633167023045853
Test Accuracy: 0.875

_____________________________________________________________

Model: AdaBoostClassifier
Cross-validation Accuracy: 0.5286944642432883
Test Accuracy: 0.6739130434782609

_____________________________________________________________

Best Model: DecisionTreeClassifier
Test Accuracy of the Best Model: 91.30434782608695
CPU times: total: 7min 8s
Wall time: 4min 42s
```

9.4. Saving the best model:

In [133…

```
pickle.dump(best_model_name, open('heart_prediction_model_best', 'wb'))
```

Chapter 10: Conclusion

- A comparative analysis of nine machine learning algorithms was conducted to predict heart disease. Results indicate that the Decision Tree Classifier exhibited superior performance, achieving an accuracy of `91.30%`. This model will serve as the foundation for further development and refinement in heart disease prediction.

Contact Details

Click on link below to contact/follow/correct me:

- **Email:** waqasliaqat630@gmailcom
- Linkedin
- Github
- kaggle