In [1]:
```
import pandas as pd
import numpy as np


import os

dirname = r"C:\Users\ANITHA\Downloads"
filename = "heart.csv"

print(os.path.join(dirname, filename))
```

```
C:\Users\ANITHA\Downloads\heart.csv
```

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as st
%matplotlib inline
sns.set(style="whitegrid")
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
df=pd.read_csv(r"C:\Users\ANITHA\Downloads\25th, 26th- Advanced EDA project\25th, 2
```

In [5]:
```
df
```

Out[5]:

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

303 rows × 14 columns

In [6]:
```
df.shape
```

Out[6]: (303, 14)

In [7]:
```
df.head()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	t
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	

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

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

In [9]:
```
df.describe()
```

Out[9]:

	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

In [10]:
```
df.columns
```

Out[10]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

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

Out[11]:
```
2
```

In [12]:
```
df['target'].unique()
```

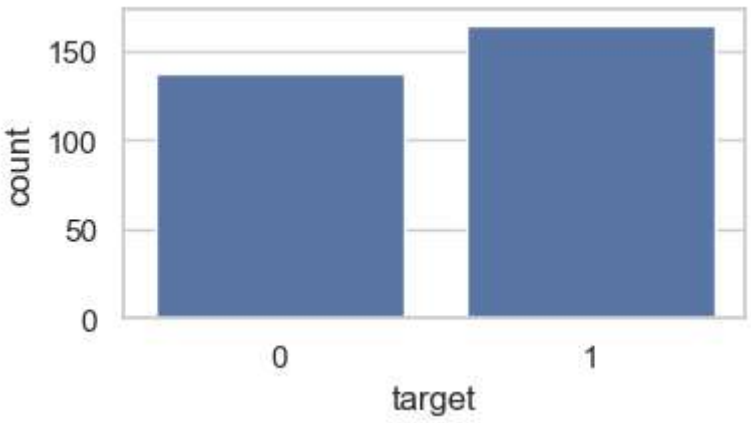
Out[12]:
```
array([1, 0])
```

In [13]:
```
df['target'].value_counts()
```

Out[13]:
```
target
1    165
0    138
Name: count, dtype: int64
```

***Frequency Distribution of Target Variable**

In [14]:
```
f,ax = plt.subplots(figsize=(4,2))
ax = sns.countplot(x="target",data=df)
plt.show()
```



In [15]:
```
df
```

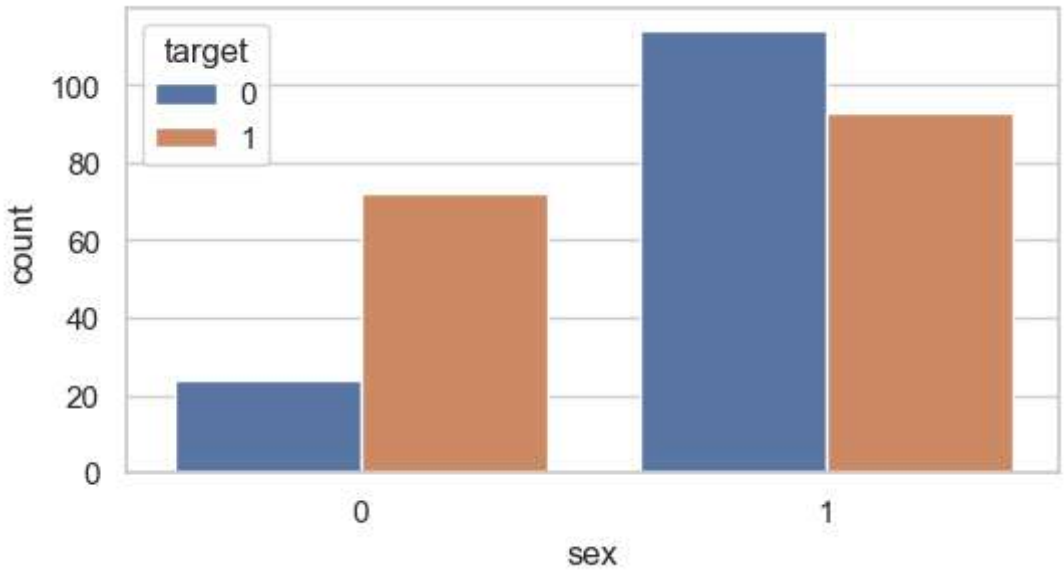
Out[15]:

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

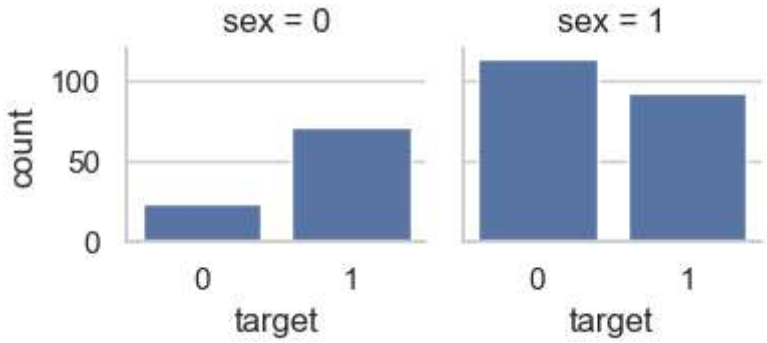
303 rows × 14 columns

visualizing the value counts of sex variable with-respect=to target

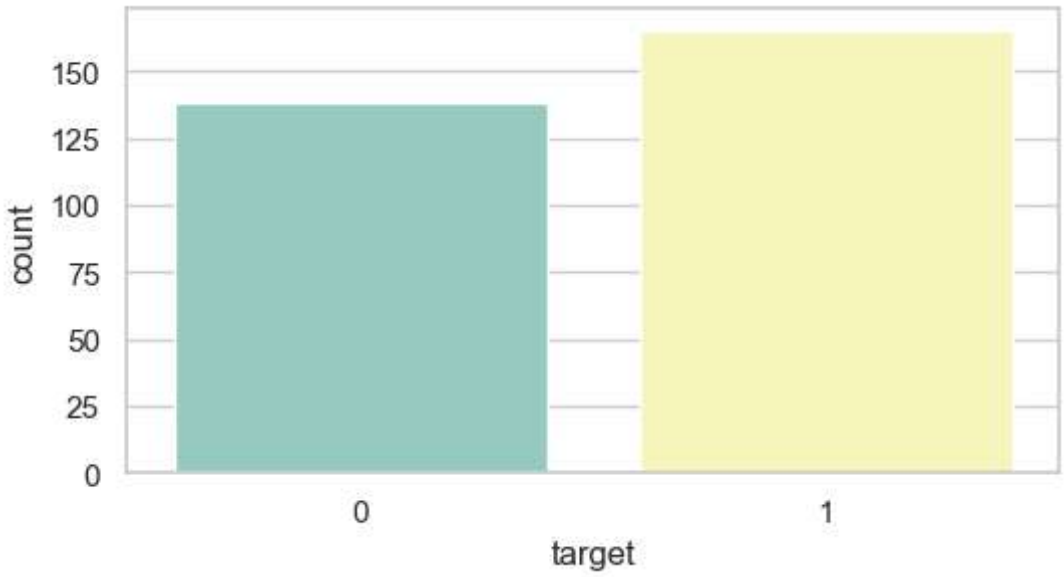
In [16]:
```
f,ax = plt.subplots(figsize=(6,3))
ax = sns.countplot(x="sex", hue="target",data=df)
plt.show()
```



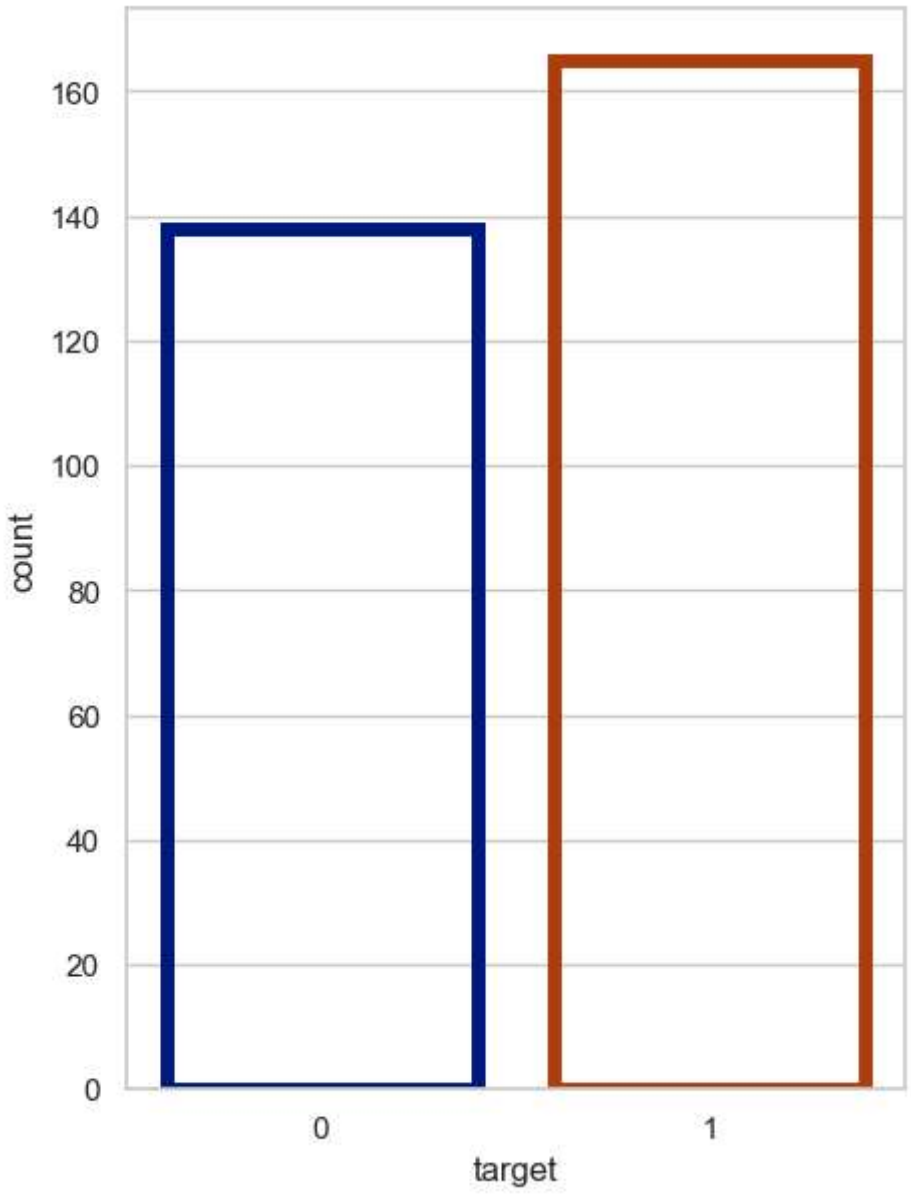
In [17]:
```
ax = sns.catplot(x="target",col="sex",data=df,kind="count",height=2,aspect=1)
plt.show()
```



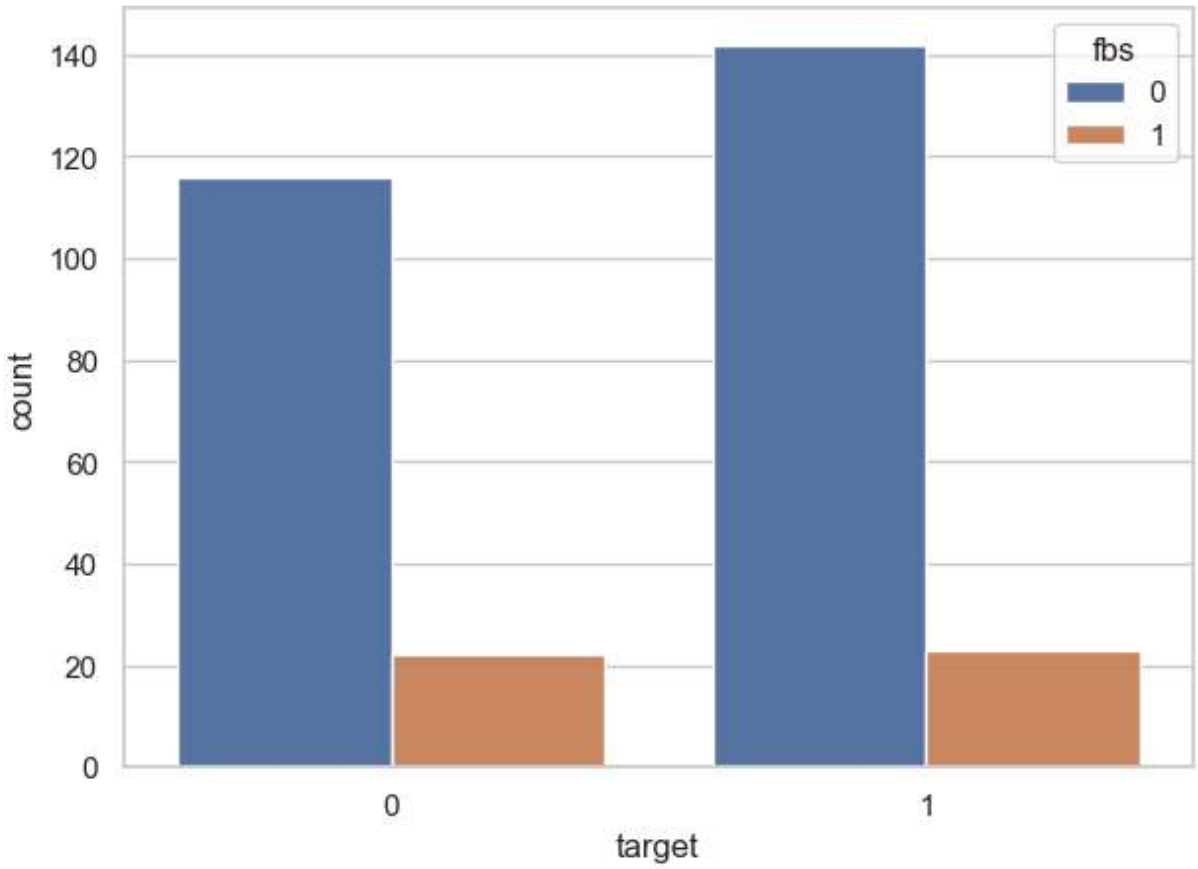
In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns

f, ax = plt.subplots(figsize=(6, 3))
sns.countplot(x='target', data=df, palette='Set3', ax=ax)
plt.show()
```



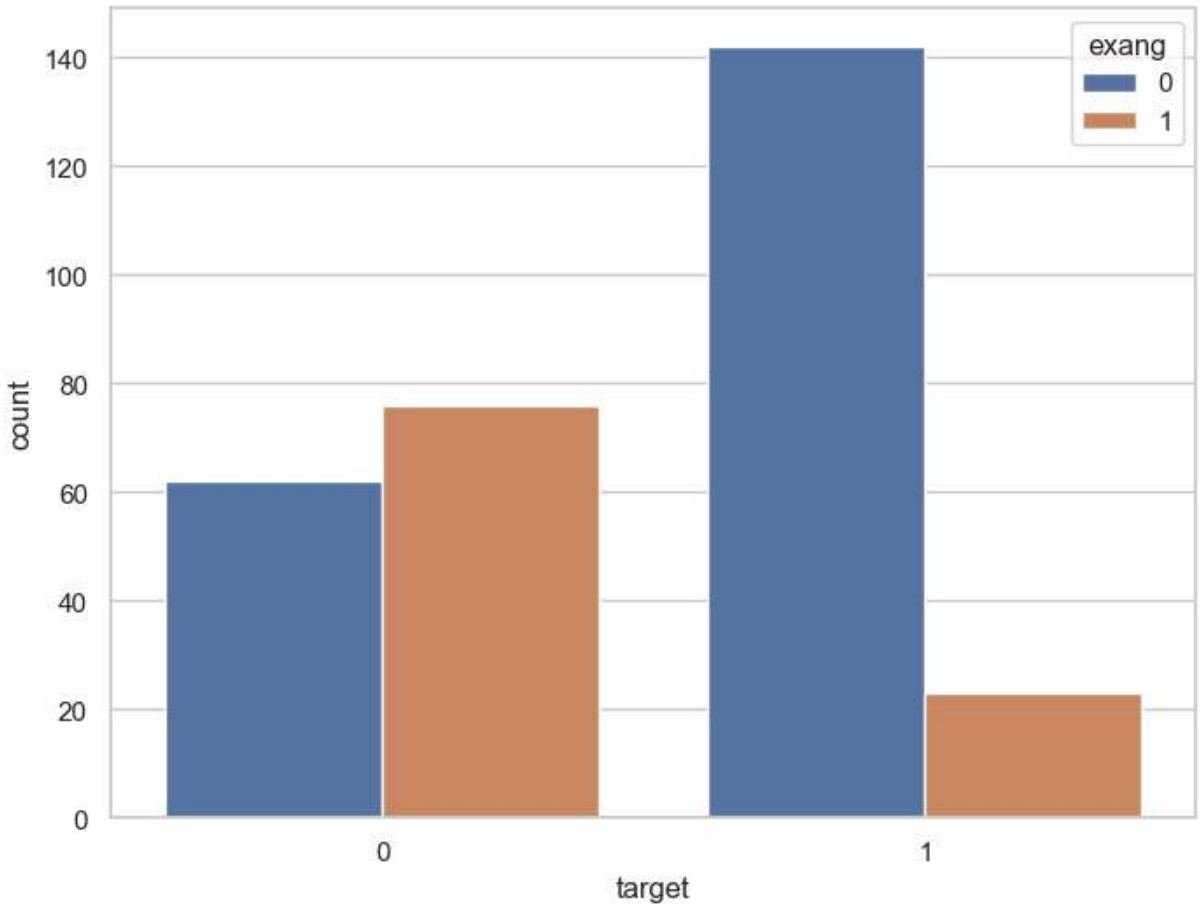
In [19]:
```
f, ax = plt.subplots(figsize =(5,7))
ax = sns.countplot(x = "target",data = df,facecolor=(0,0,0,0),linewidth=5,edgecolor
plt.show()
```



In [20]:
```
f,ax = plt.subplots(figsize=(7,5))
ax = sns.countplot(x="target",hue = "fbs",data =df)
plt.show()
```



In [21]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x="target",hue = "exang",data =df)
plt.show()
```



Bivariate Analysis

In [22]:
```
correlation = df.corr()
```

In [23]:
```
correlation["target"].sort_values(ascending = False)
```

Out[23]:
```
target      1.000000
cp          0.433798
thalach     0.421741
slope       0.345877
restecg     0.137230
fbs        -0.028046
chol       -0.085239
trestbps   -0.144931
age        -0.225439
sex        -0.280937
thal       -0.344029
ca         -0.391724
oldpeak    -0.430696
exang      -0.436757
Name: target, dtype: float64
```

Analysis of target and cp variable

cp stands for chest pain type

In [24]:
```
df ['cp']. nunique()
```

Out[24]: 4

so, there are 4 unique values in cp variable.Hence, it is a categorical variable.

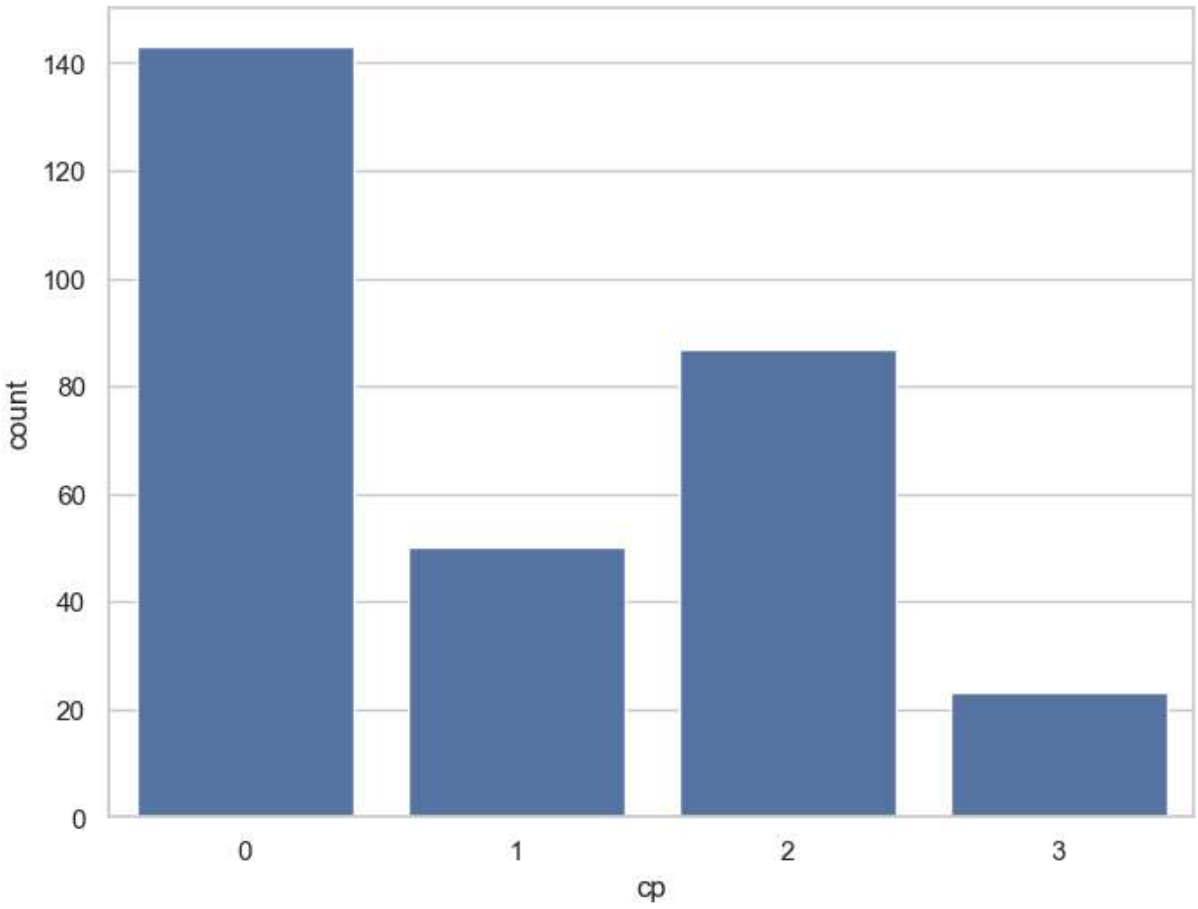
In [ ]:
```
```

In [25]:
```
df['cp'].value_counts()
```

Out[25]:
```
cp
0    143
2     87
1     50
3     23
Name: count, dtype: int64
```

Visualize the frequecy distribution of cp variable

In [26]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="cp",data=df)
plt.show()
```



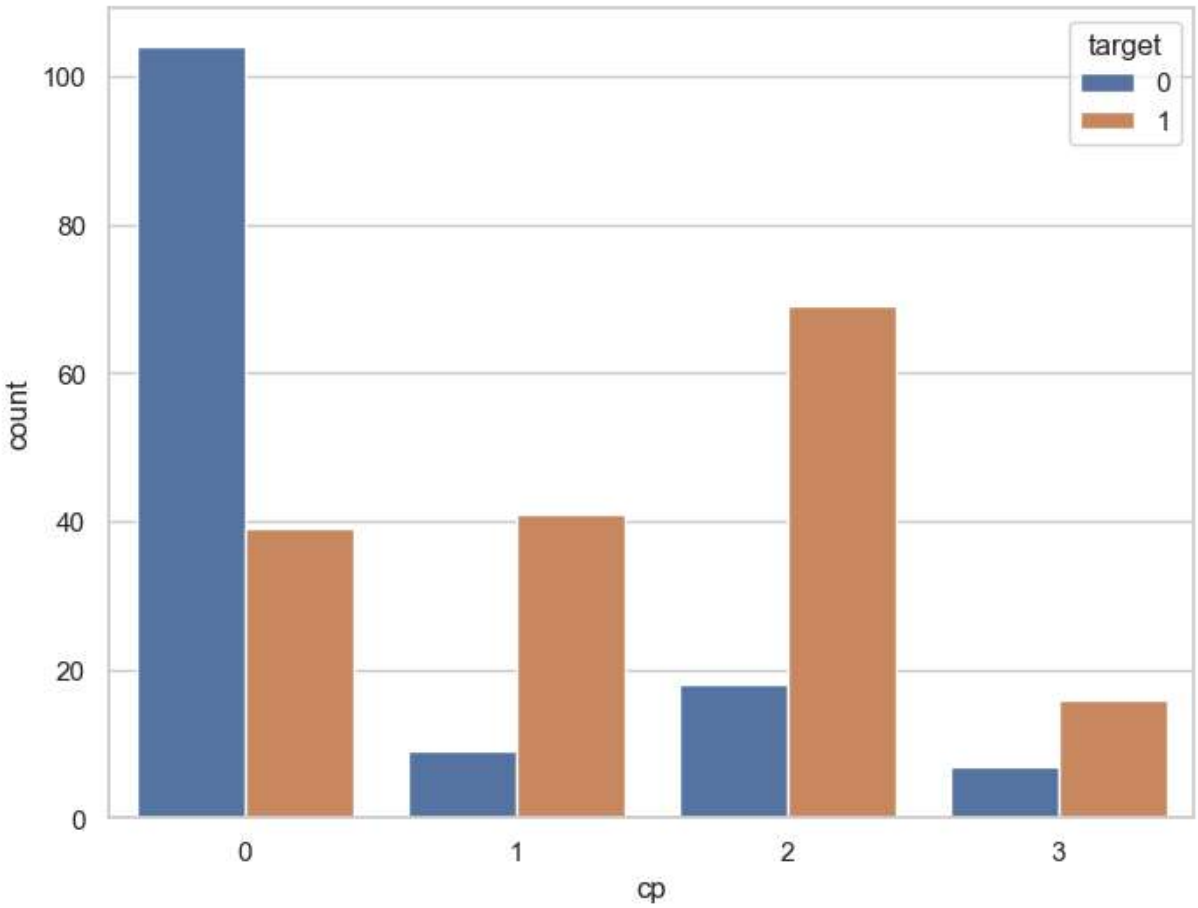
Frequency distribution of "target" variable wrt "cp"

In [27]:
```
df.groupby('cp')['target'].value_counts()
```

Out[27]:
```
cp  target
0   0         104
    1          39
1   1          41
    0           9
2   1          69
    0          18
3   1          16
    0           7
Name: count, dtype: int64
```

We can visualize the value counts of the `cp` variable wrt `target` as follows

In [28]:
```
f, ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x="cp",hue="target",data=df)
plt.show()
```



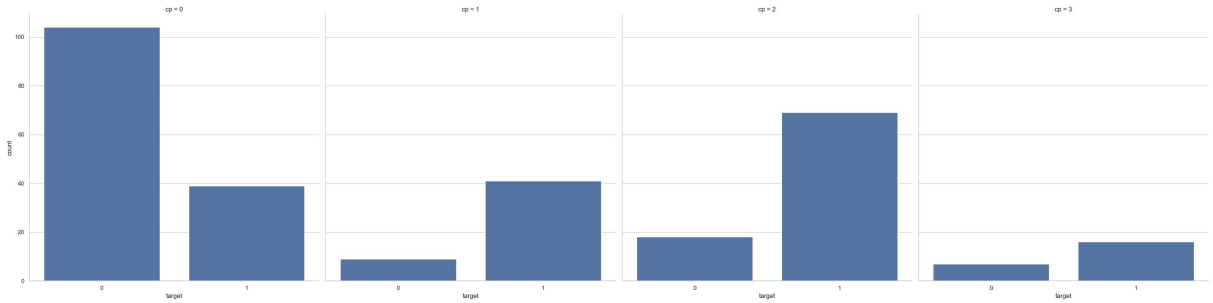
Interpretation

- We can see that the values of `target` variable are plotted wrt `cp`.

- `target` variable contains two integer values 1 and 0 : (1 = Presence of heart disease; 0 = Absence of heart disease)
- The above plot confirms our above findings,

Alternatively, we can visualize the same information as follows:

In [29]:
```
ax = sns.catplot(x="target",col="cp", data=df,kind="count",height=8,aspect=1)
plt.show()
```



Analysis of target and thalach variable

Explore thalach variable

- thalach stands for maximum heart rate achieved.
- I will check number of unique values in thalach variable as follows:

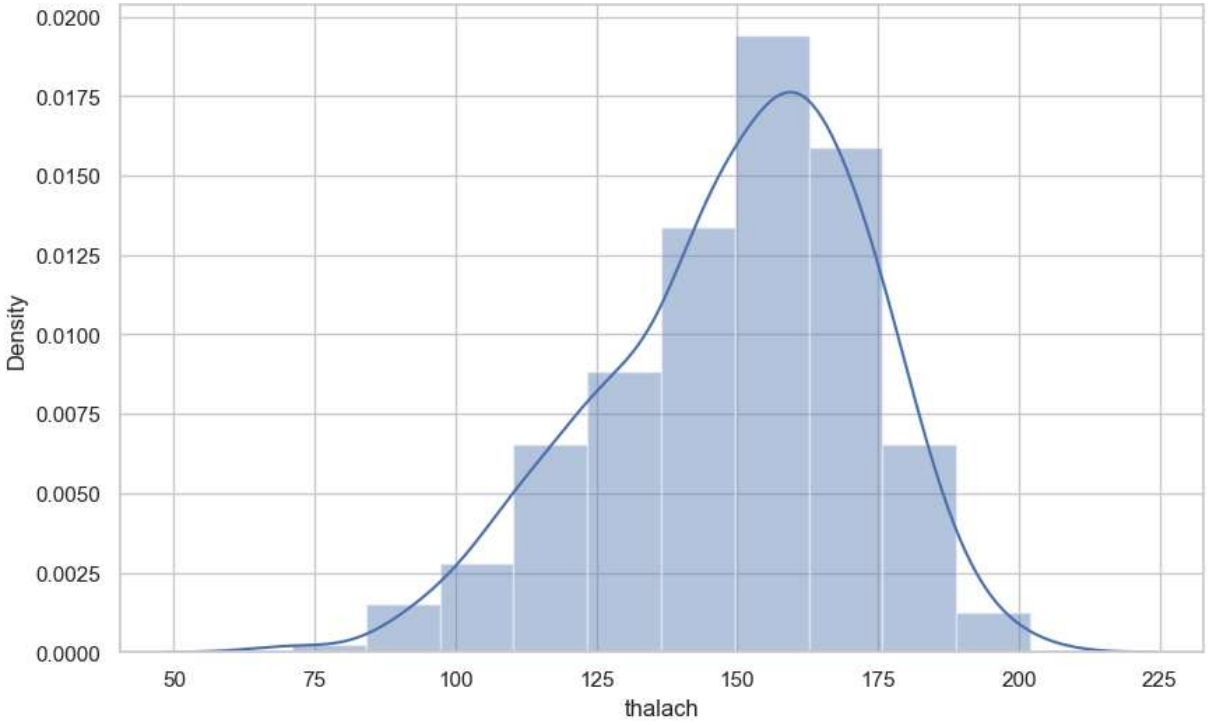
In [30]:
```
df['thalach'].nunique()
```

Out[30]: 91

- So, number of unique values in `thalach` variable is 91. Hence, it is numerical variable.
- I will visualize its frequency distribution of values as follows :

Visualize the frequency distribution of thalach variable

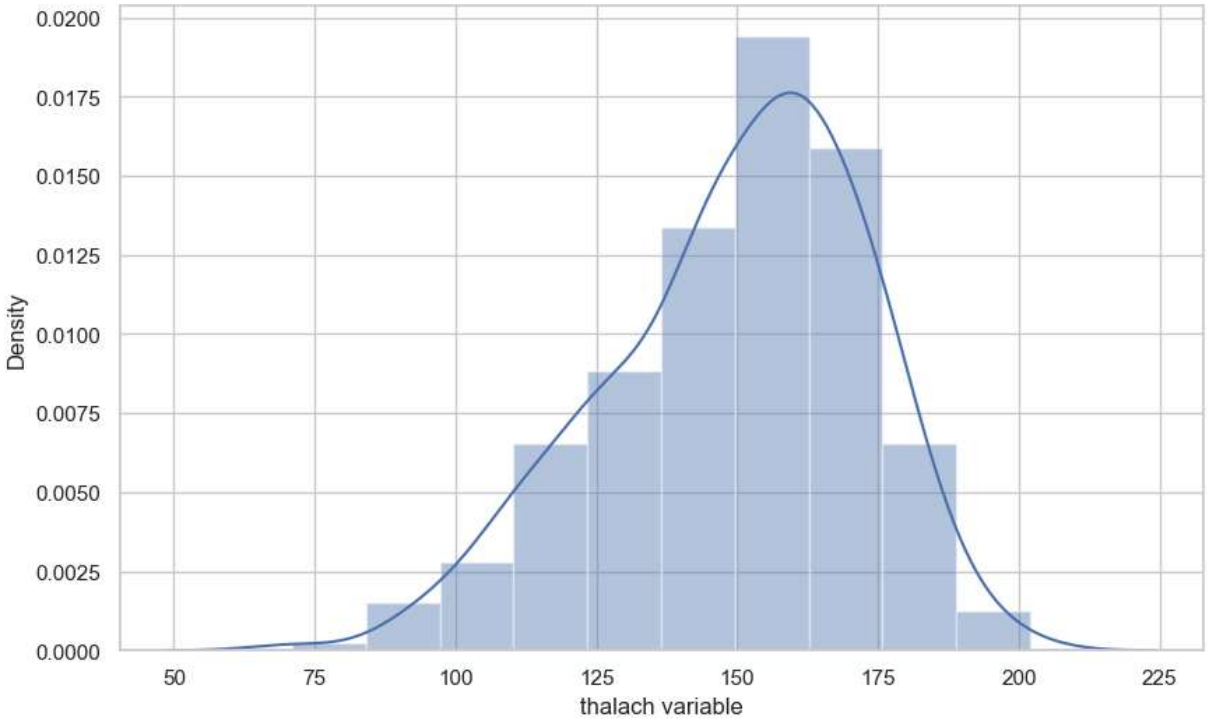
In [31]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
ax = sns.distplot(x, bins=10)
plt.show()
```



- We can see that the thalach variable is slightly negatively skewed

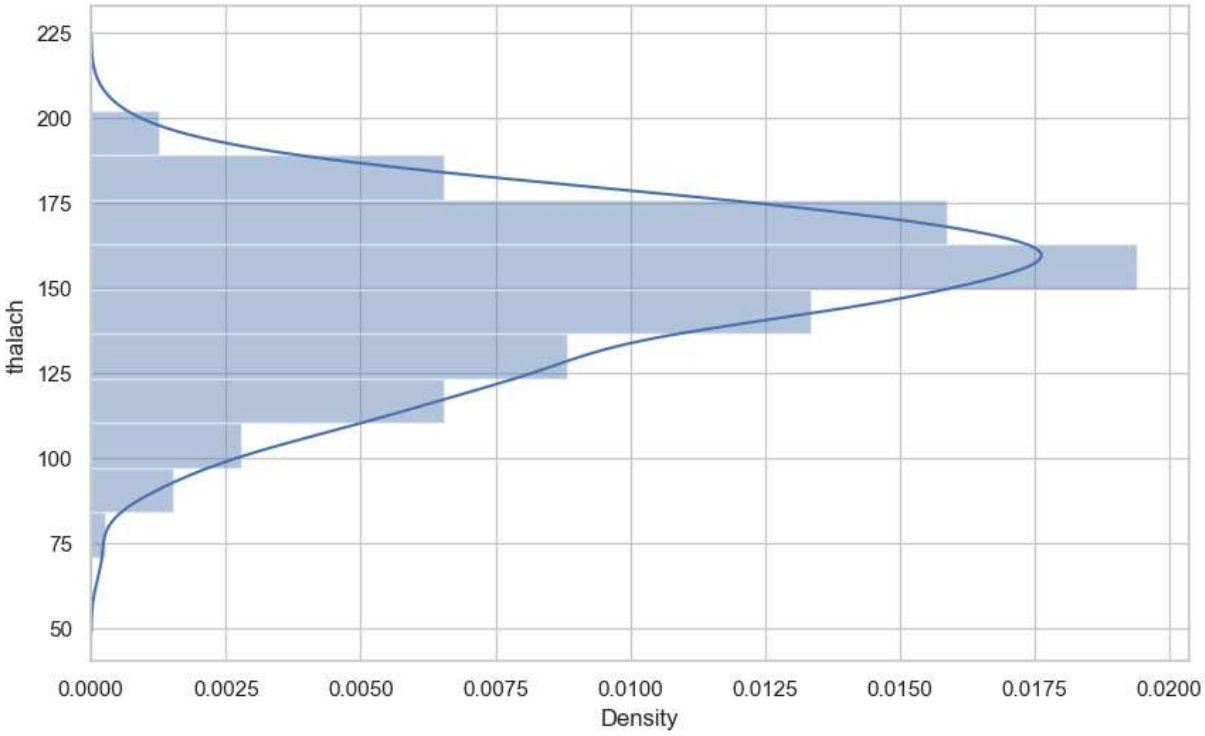
we can use pandas series object to get an information axis label as follows:

In [32]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x, name="thalach variable")
ax = sns.distplot(x, bins=10)
plt.show()
```



We can plot the distribution on the vertical axis as follows:-

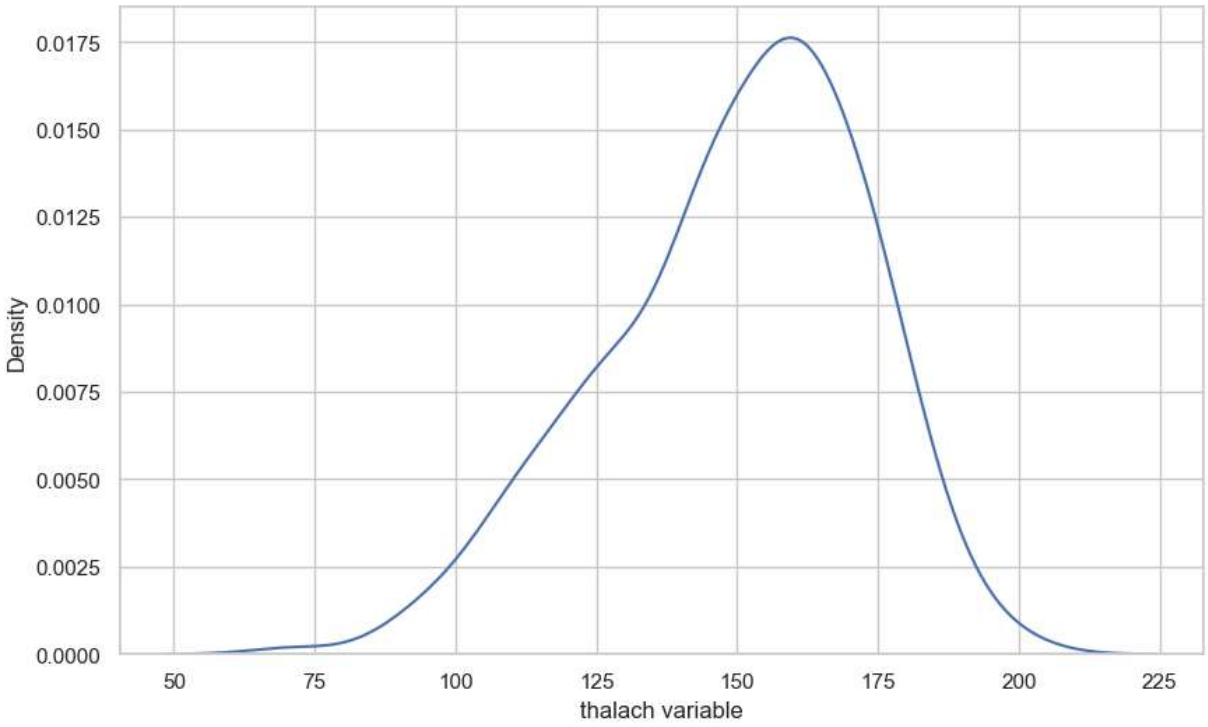
In [33]:
```
f, ax = plt.subplots(figsize=(10,6))
x =df['thalach']
ax = sns.distplot(x, bins=10, vertical=True)
plt.show()
```



Seaborn Kernel Density Estimation (KDE) Plot

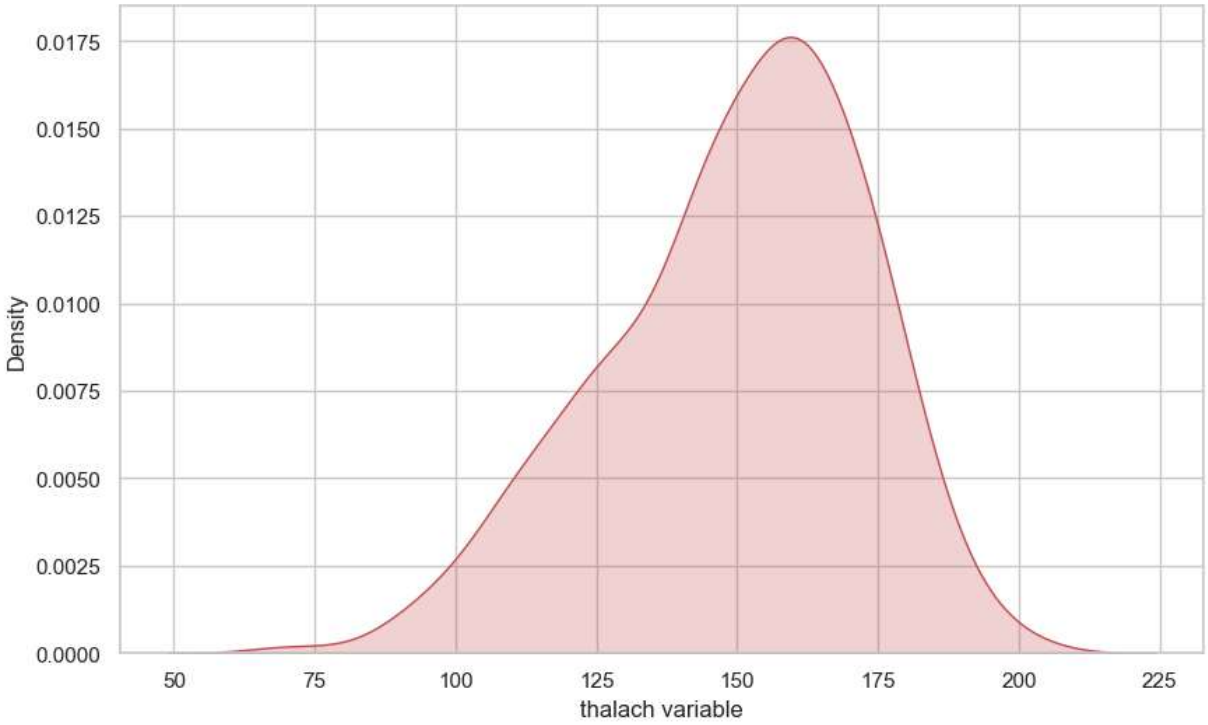
- The kernel density estimate (KDE) plot is a useful tool for plotting the shape of a distribution.
- The KDE plot plots the density of observations on one axis with height along the other axis.
- We can plot a KDE plot as follows :

In [34]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x, name="thalach variable")
ax = sns.kdeplot(x)
plt.show()
```



We can shade under the density curve and use a different color as follows:

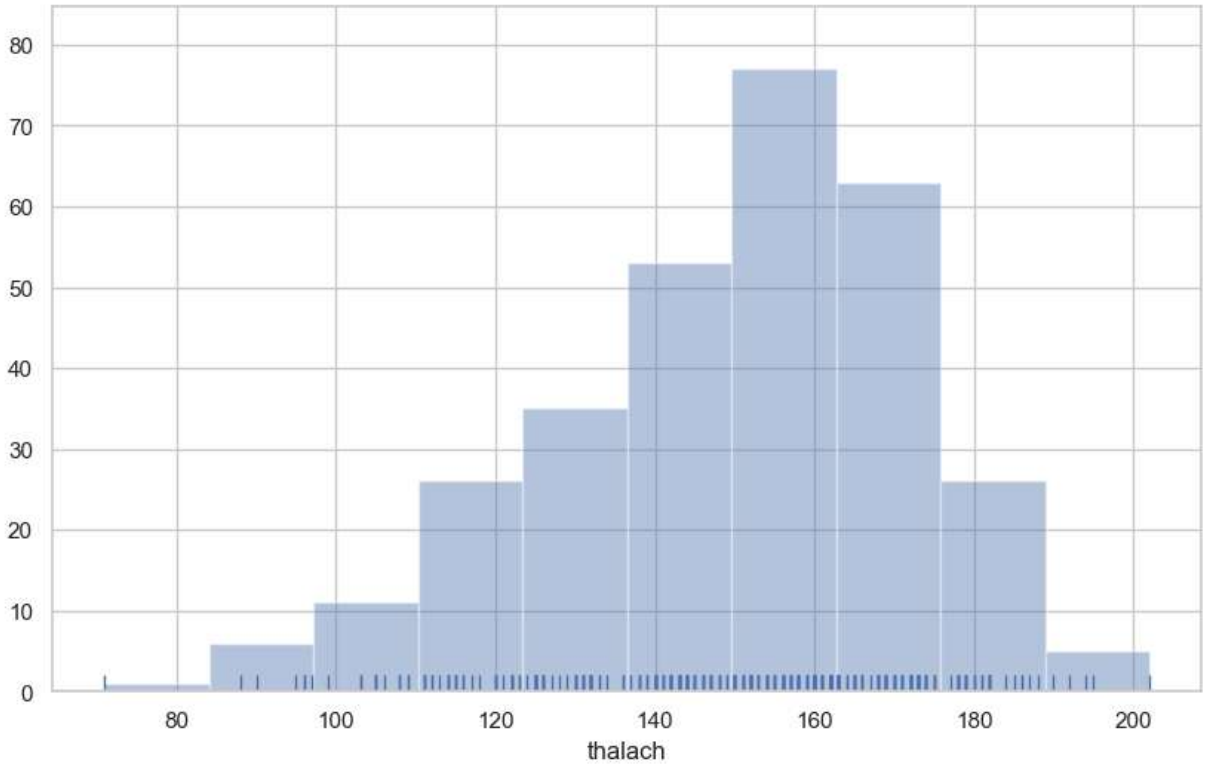
In [35]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x, name="thalach variable")
ax = sns.kdeplot(x,shade=True,color='r')
plt.show()
```



Histogram

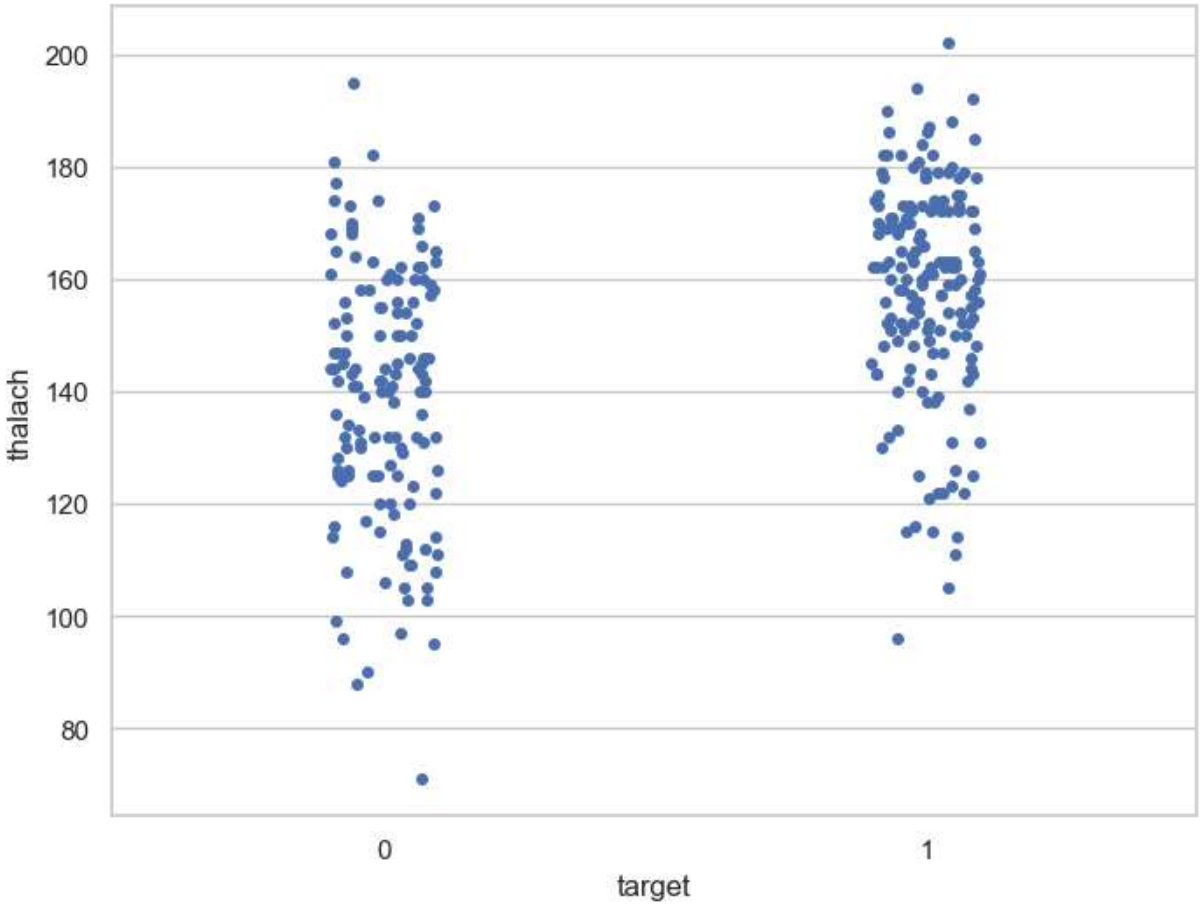
- A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.
- We can plot a histogram as follows :

In [36]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
ax = sns.distplot(x, kde=False, rug=True, bins=10)
plt.show()
```



Visualize frequency distribution of `thalach` variable wrt `target`

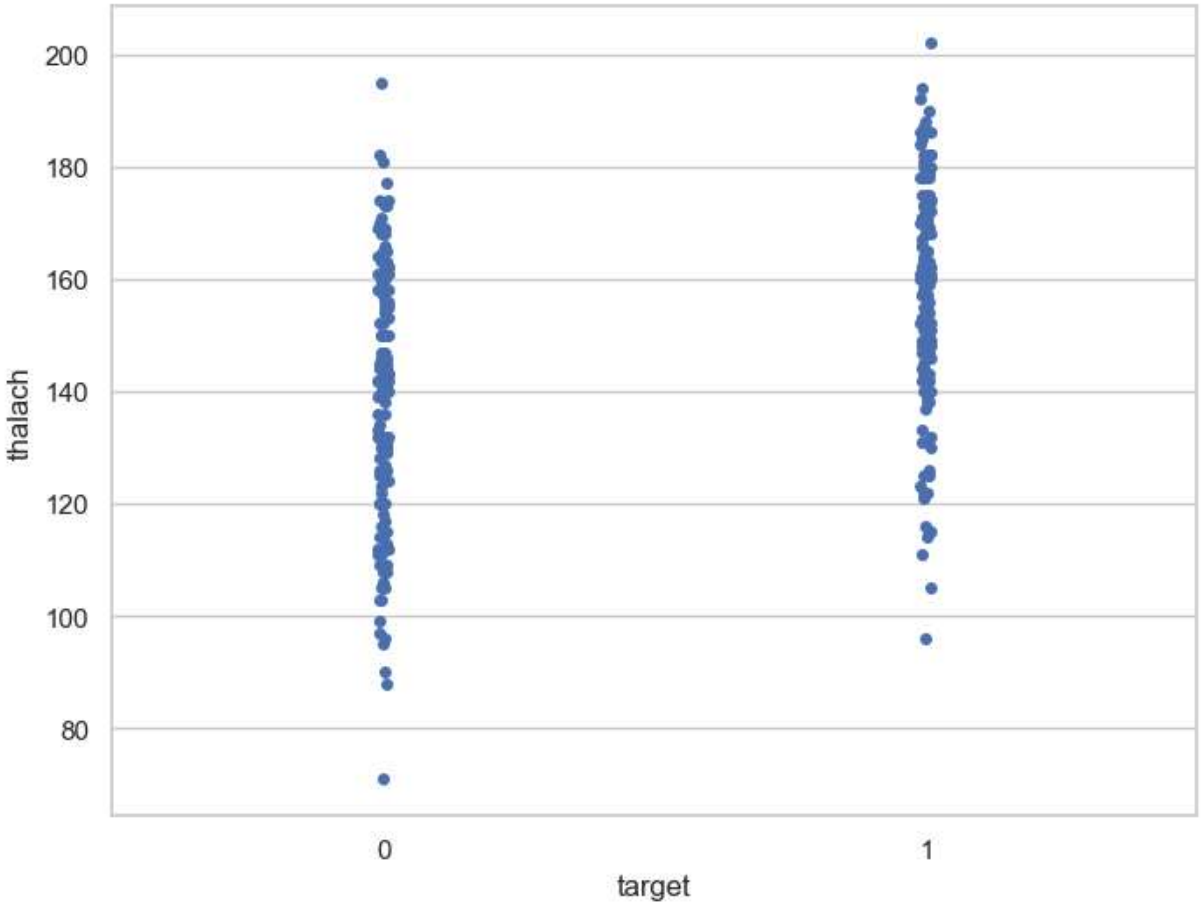
In [37]:
```
f, ax = plt.subplots(figsize=(8,6))
sns.stripplot(x="target",y="thalach",data=df)
plt.show()
```



- We can see that those people suffering from heart disease (target = 1) have relatively higher heart rate (thalach) as compared to people who are not suffering from heart disease (target = 0).

We can add jitter to bring out the distribution of values as follows:

In [38]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="target", y="thalach", data=df, jitter = 0.01)
plt.show()
```



Visualize distribution of `thalach` variable wrt `target` with boxplot

f, ax =plt.subplots(figsize=(8,6)) sns.boxplot(x="target", y="thalach",data=df) plt.show()

Interpretation

The above boxplot confirms our finding that people suffering from heart disease (target = 1) have relatively higher heart rate (thalach) as compared to people who are not suffering from heart disease (target = 0).

Findings of Bivariate Analysis

Findings of Bivariate Analysis are as follows –

- There is no variable which has strong positive correlation with `target` variable.
- There is no variable which has strong negative correlation with `target` variable.
- There is no correlation between `target` and `fbs`.
- The `cp` and `thalach` variables are mildly positively correlated with `target` variable.
- We can see that the `thalach` variable is slightly negatively skewed.

- The people suffering from heart disease (target = 1) have relatively higher heart rate (thalach) as compared to people who are not suffering from heart disease (target = 0).

- The people suffering from heart disease (target = 1) have relatively higher heart rate (thalach) as compared to people who are not suffering from heart disease (target = 0).

Multivariate analysis

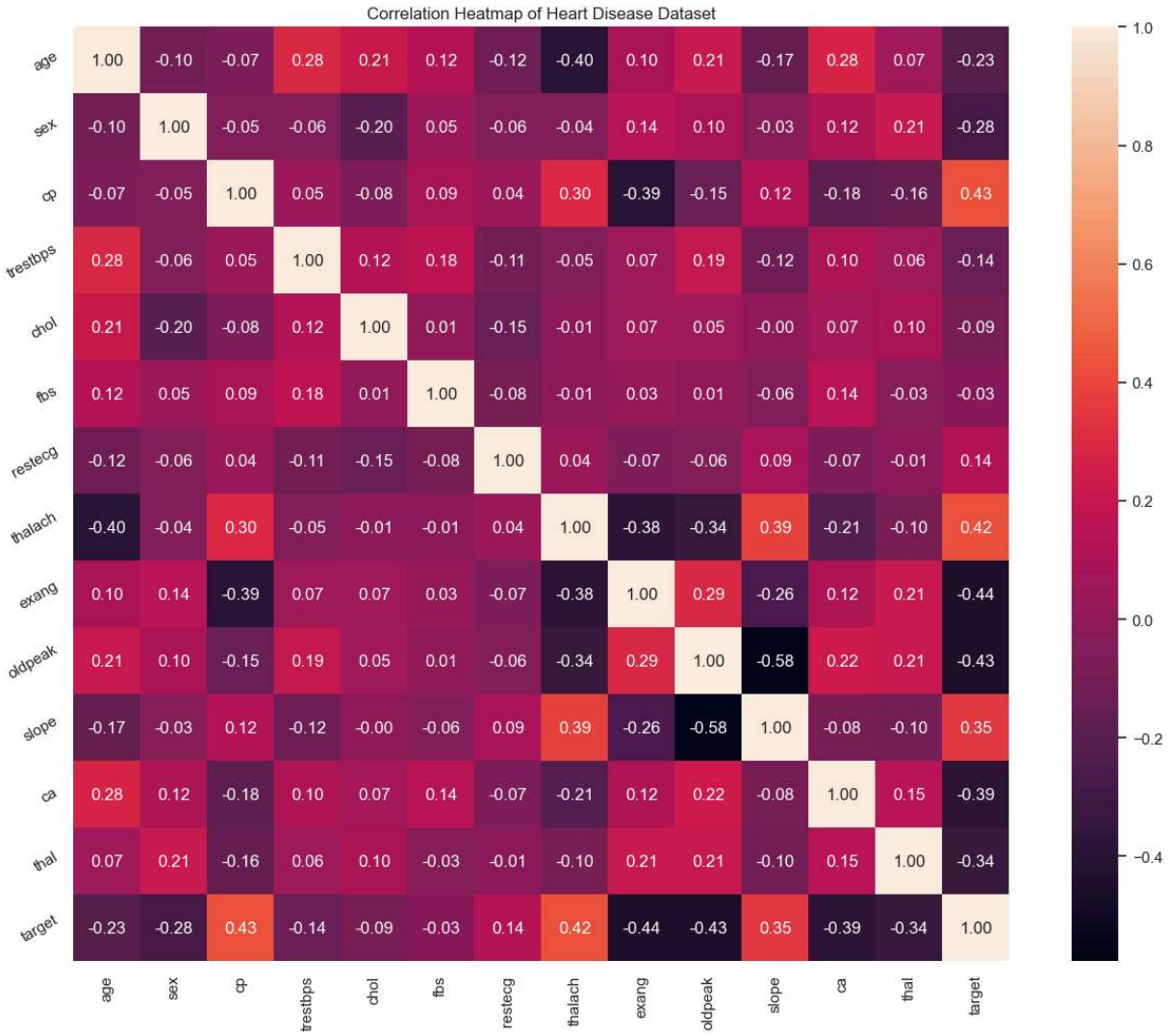
- The objective of the multivariate analysis is to discover patterns and relationships in the dataset.

Discover patterns and relationships

- An important step in EDA is to discover patterns and relationships between variables in the dataset.
- I will use `heat map` and `pair plot` to discover the patterns and relationships in the dataset.
- First of all, I will draw a `heat map`.

Heat Map

In [39]:
```
plt.figure(figsize=(16,12))
plt.title('Correlation Heatmap of Heart Disease Dataset')
a = sns.heatmap(correlation, square=True, annot=True, fmt='.2f', linecolor='white')
a.set_xticklabels(a.get_xticklabels(), rotation=90)
a.set_yticklabels(a.get_yticklabels(), rotation=30)
plt.show()
```



Interpretation

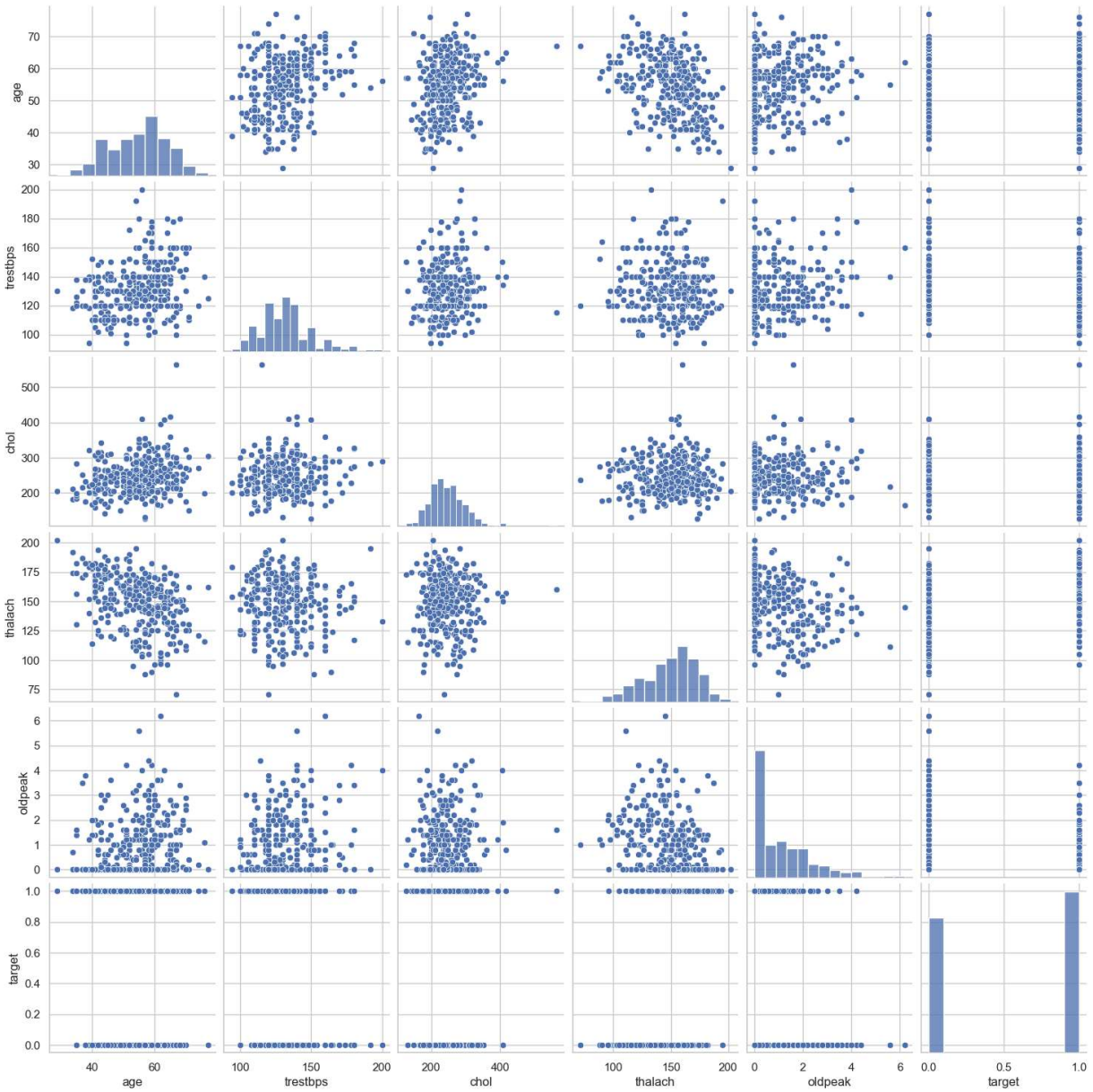
From the above correlation heat map, we can conclude that :-

- `target` and `cp` variable are mildly positively correlated (correlation coefficient = 0.43).
- `target` and `thalach` variable are also mildly positively correlated (correlation coefficient = 0.42).
- `target` and `slope` variable are weakly positively correlated (correlation coefficient = 0.35).
- `target` and `exang` variable are mildly negatively correlated (correlation coefficient = -0.44).
- `target` and `oldpeak` variable are also mildly negatively correlated (correlation coefficient = -0.43).

- `target` and `ca` variable are weakly negatively correlated (correlation coefficient = -0.39).
- `target` and `thal` variable are also waekly negatively correlated (correlation coefficient = -0.34).

Pair plot

In [40]:
```
num_var = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target' ]
sns.pairplot(df[num_var], kind='scatter', diag_kind='hist')
plt.show()
```



Comment

- I have defined a variable `num_var`. Here `age`, `trestbps`, `chol`, `thalach` and `oldpeak` are numerical variables and `target` is the categorical variable.

- So, I wll check relationships between these variables.

Analysis of age and other variables

check the number of unique values in `age` variable

In [41]:
```
df['age'].nunique()
```

Out[41]:
```
41
```

View statistical summary of `age` variable

In [42]:
```
df['age'].describe()
```

Out[42]:
```
count    303.000000
mean      54.366337
std        9.082101
min       29.000000
25%       47.500000
50%       55.000000
75%       61.000000
max       77.000000
Name: age, dtype: float64
```

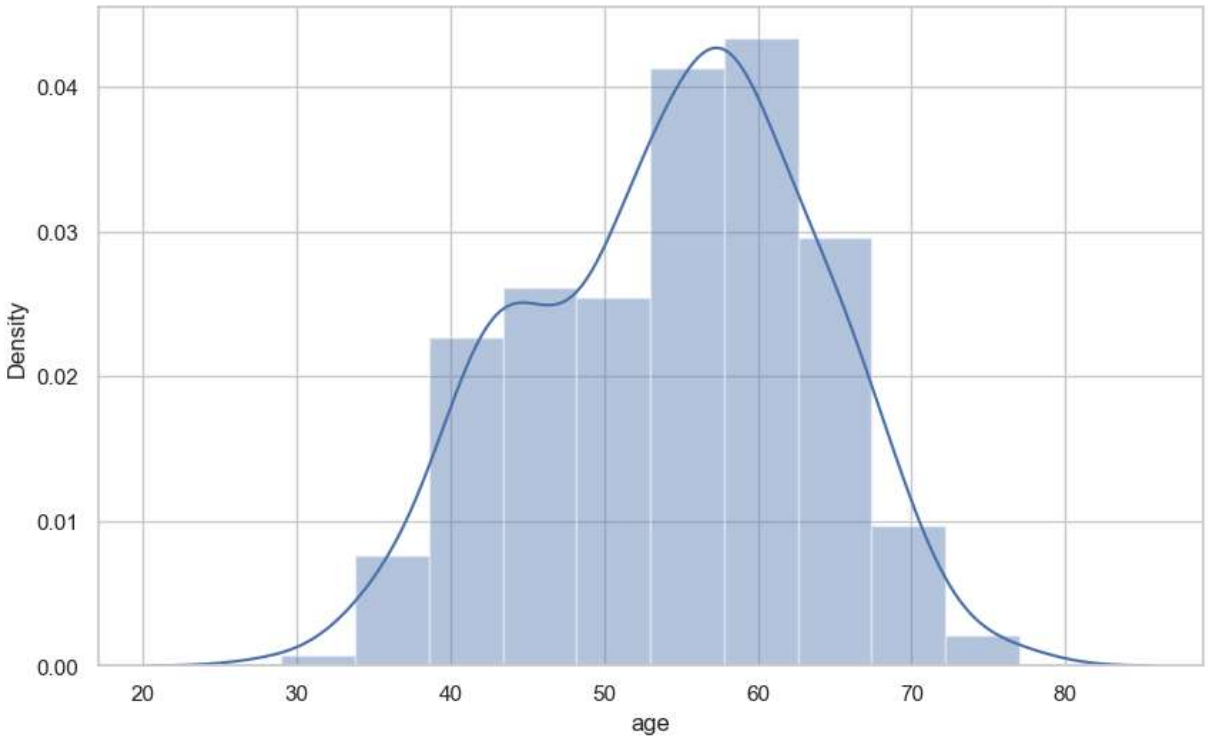
Interpretation

- The mean value of the `age` variable is 54.37 years.
- The minimum and maximum values of `age` are 29 and 77 years.

Plot the distribution of `age` variable

Now, I will plot the distribution of `age` variable to view the statistical properties.

In [43]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['age']
ax = sns.distplot(x, bins=10)
plt.show()
```



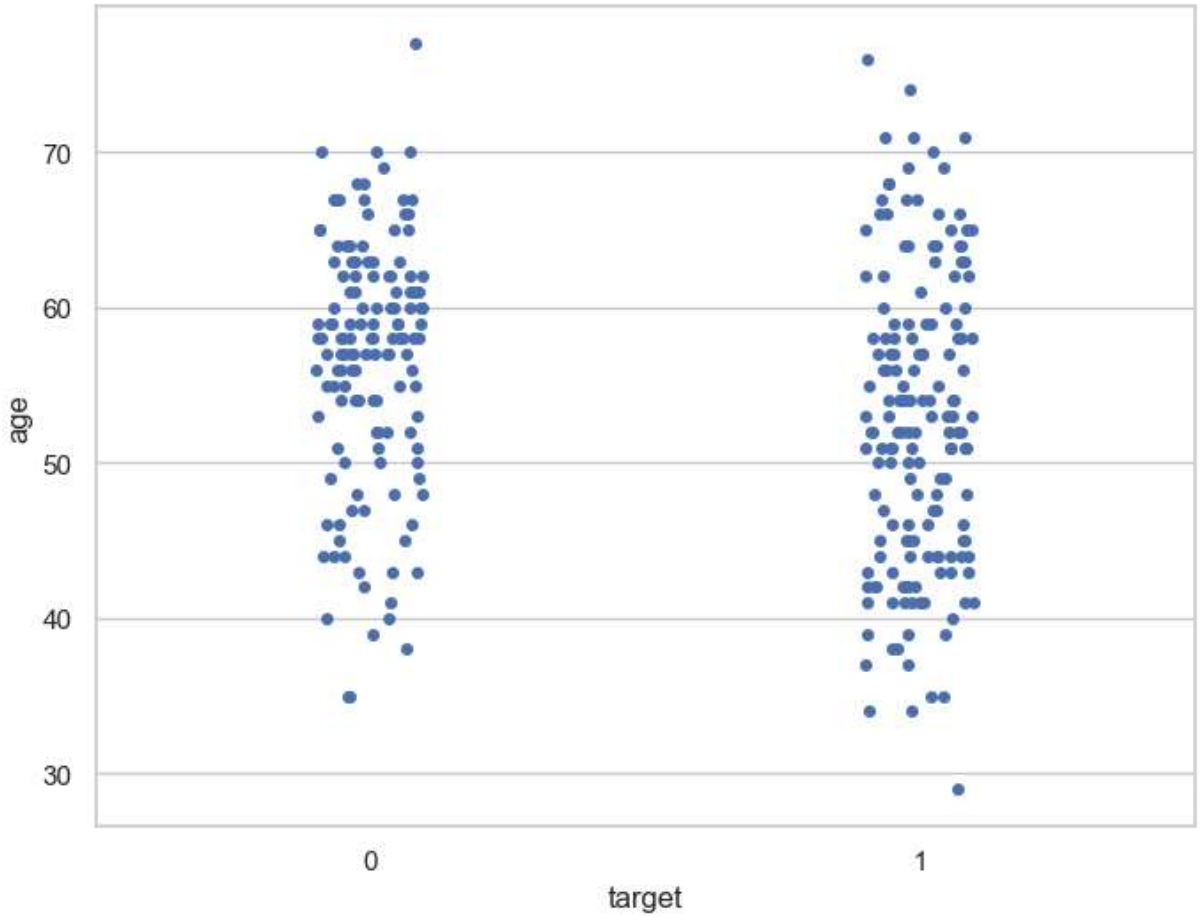
Interpretation

- The `age` variable distribution is approximately normal.

Analyze `age` and `target` variable

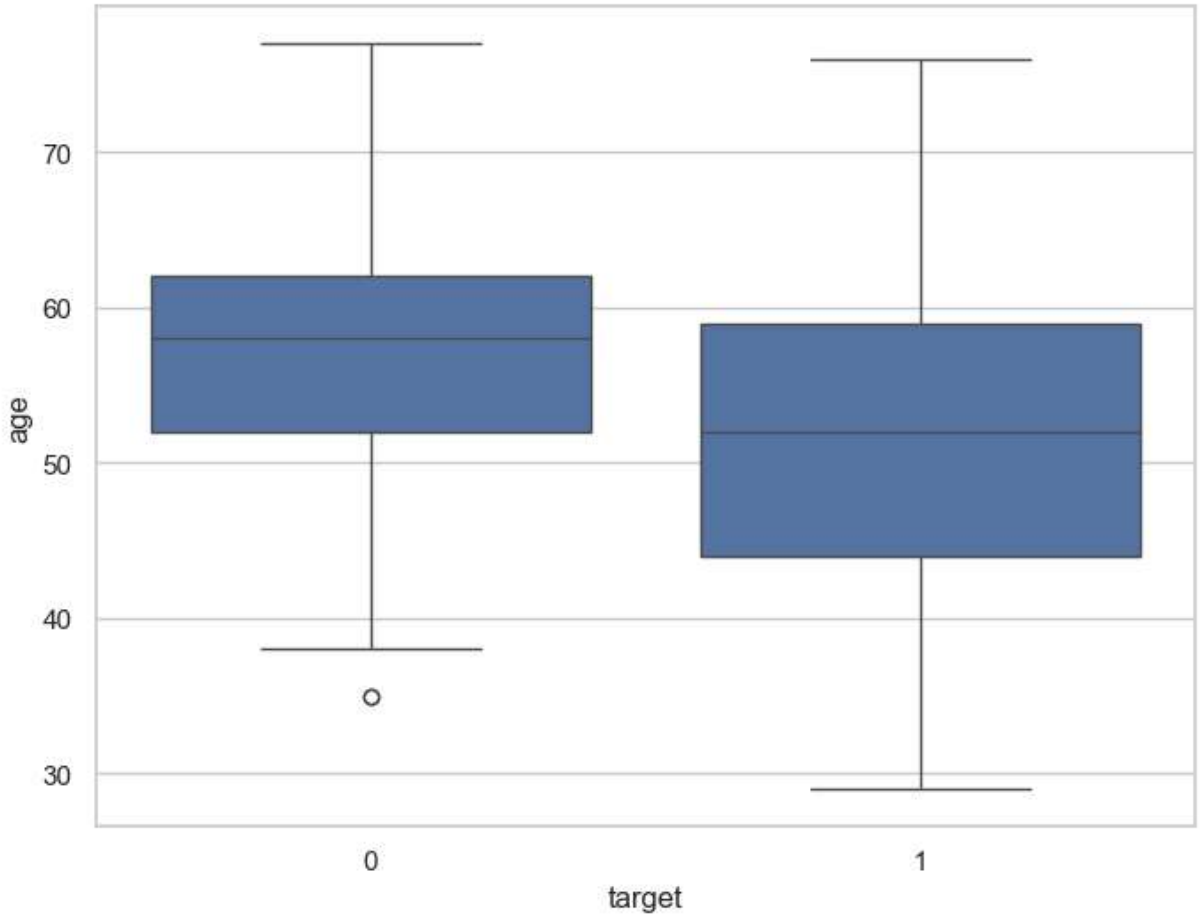
Visualize frequency distribution of `age` variable wrt `target`

In [44]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.stripplot(x="target", y="age", data=df)
plt.show()
```



Visualize distribution of `age` variable wrt `target` with boxplot

In [45]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x="target", y="age", data=df)
plt.show()
```



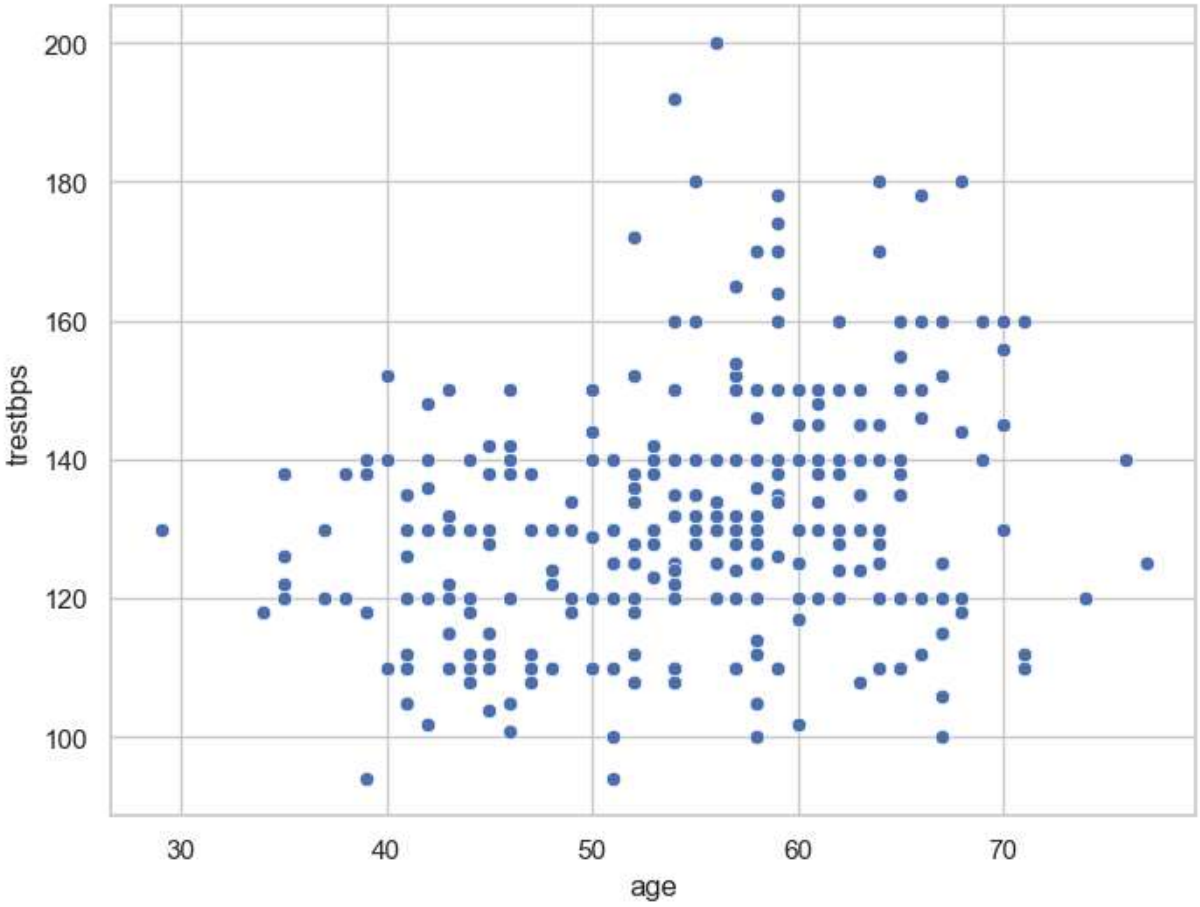
Interpretation

- The above boxplot tells two different things :
  - The mean age of the people who have heart disease is less than the mean age of the people who do not have heart disease.
  - The dispersion or spread of age of the people who have heart disease is greater than the dispersion or spread of age of the people who do not have heart disease.

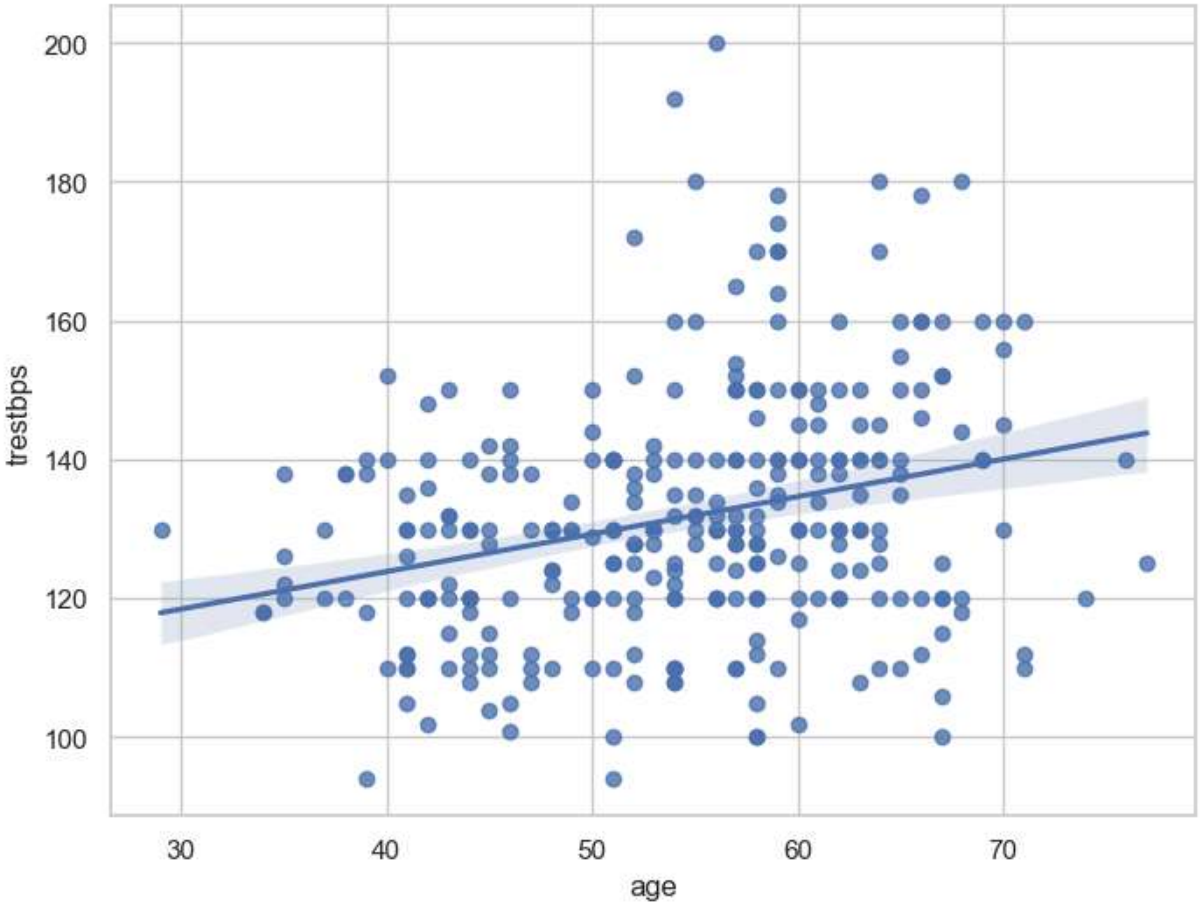
Analyze `age` and `trestbps` variable

I will plot a scatterplot to visualize the relationship between `age` and `trestbps` variable.

In [46]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.scatterplot(x="age", y="trestbps", data=df)
plt.show()
```

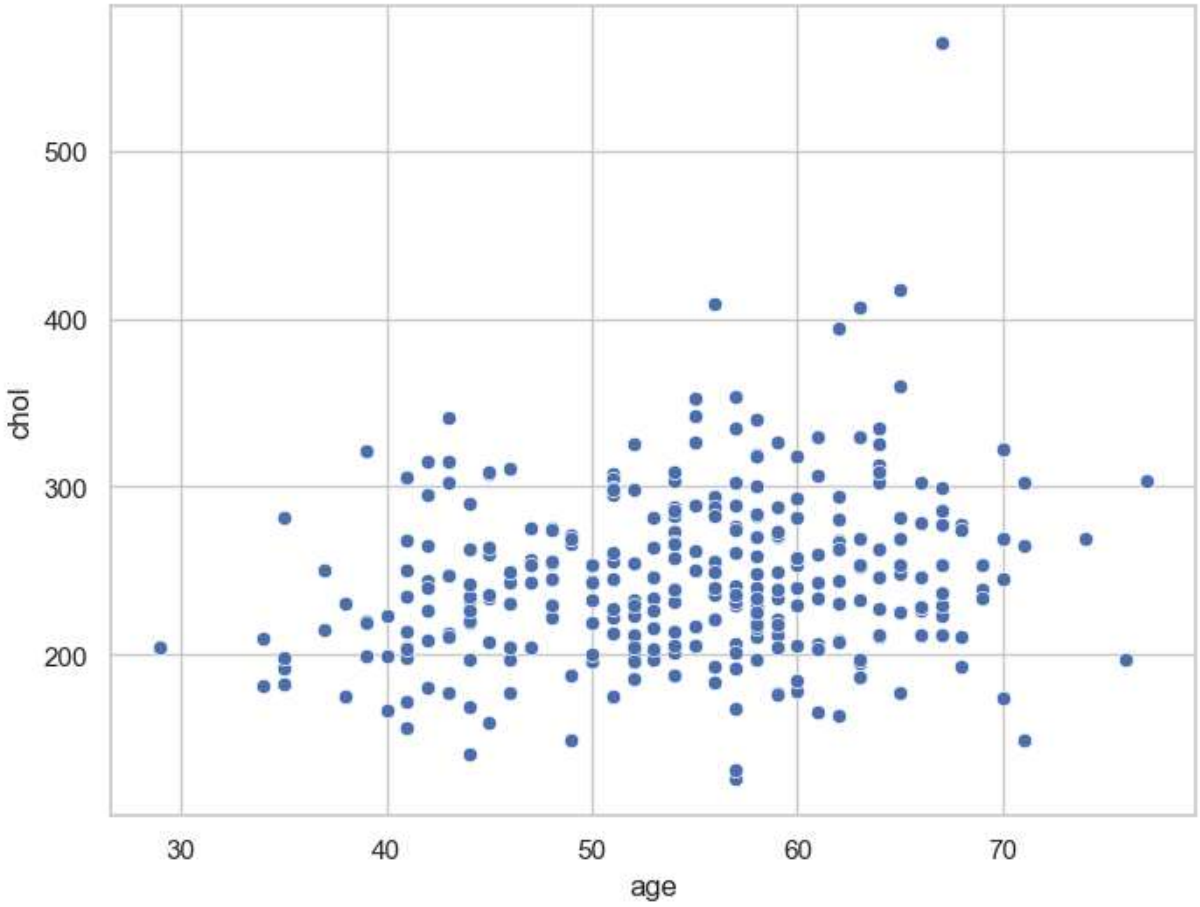


In [47]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="age", y="trestbps", data=df)
plt.show()
```

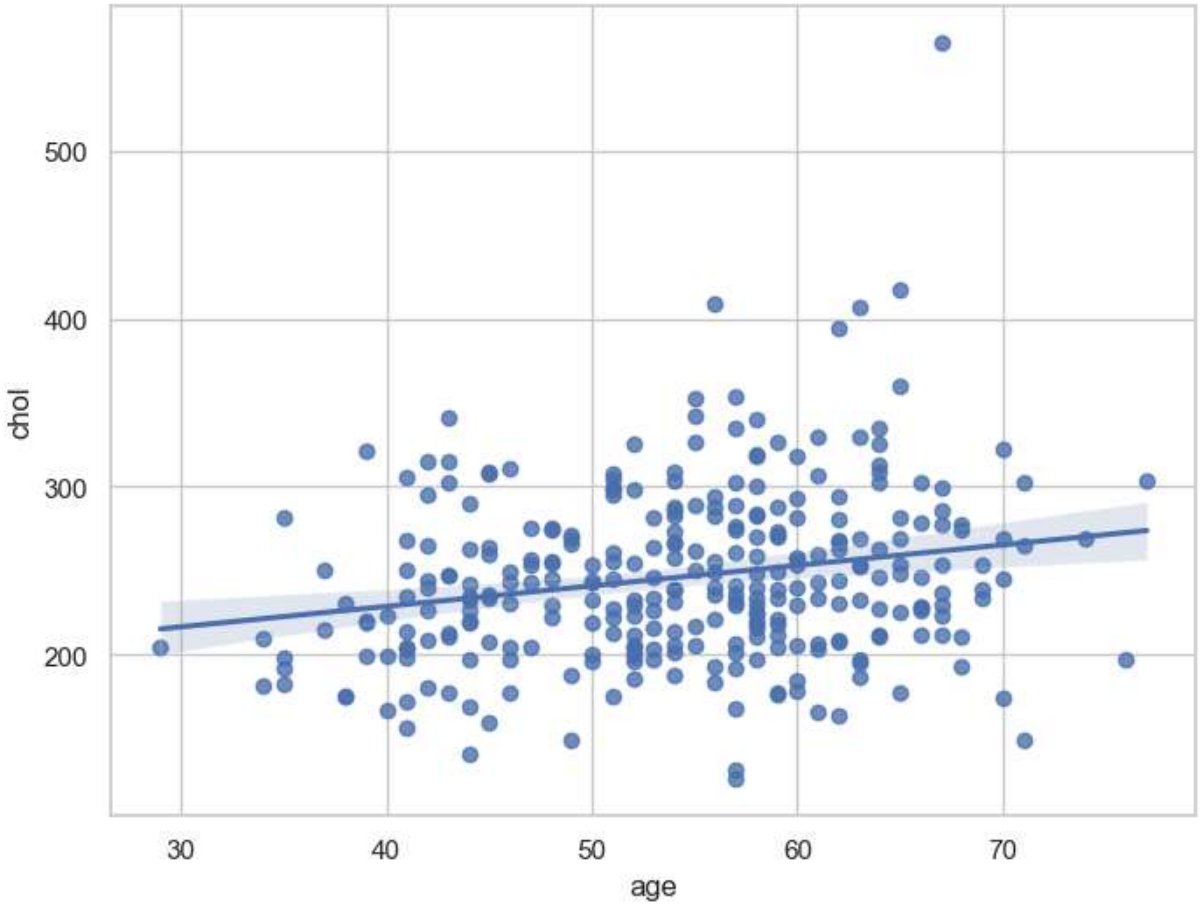


Analyze `age` and `chol` variable

In [48]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.scatterplot(x="age", y="chol", data=df)
plt.show()
```

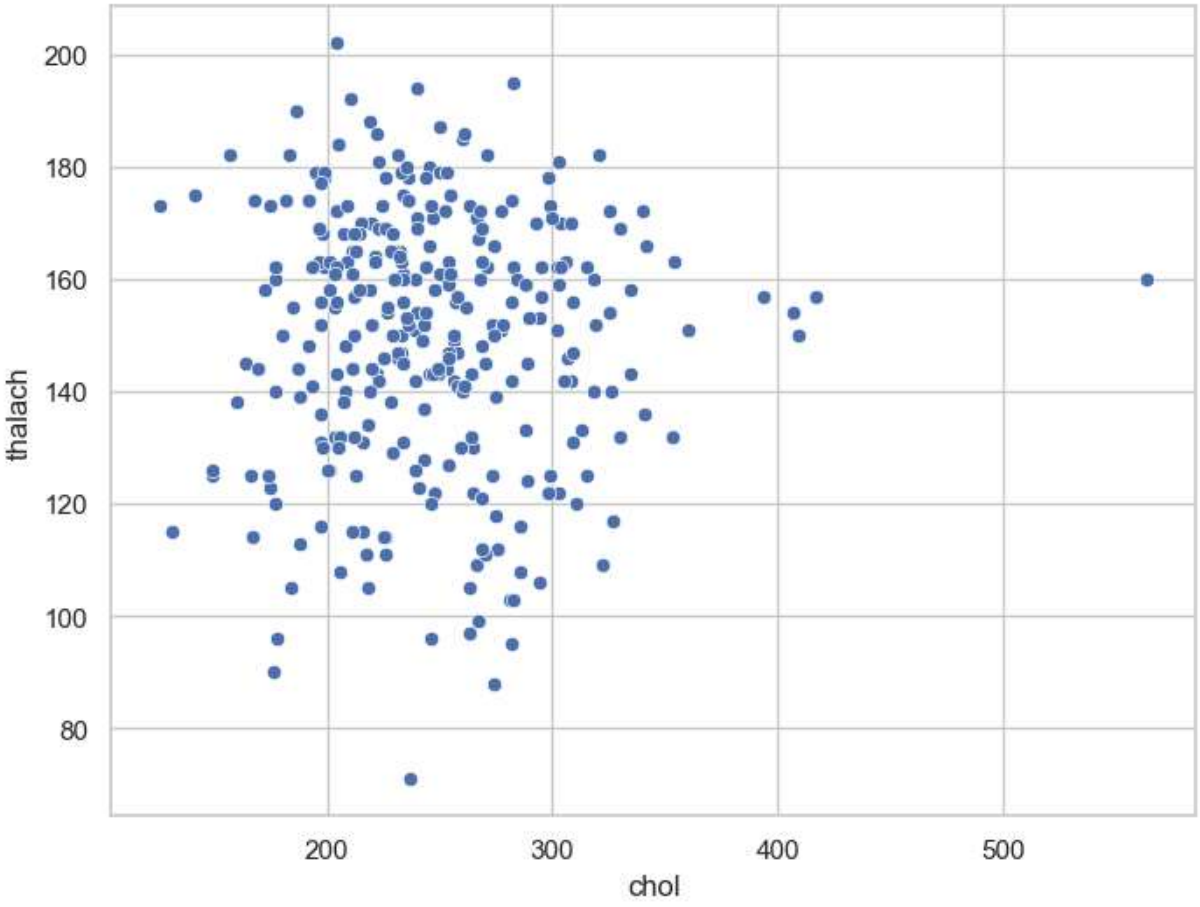


In [49]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="age", y="chol", data=df)
plt.show()
```



Analyze `chol` and `thalach` variable

In [50]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.scatterplot(x="chol", y = "thalach", data=df)
plt.show()
```



Dealing with missing values

Back to Table of Contents

- In Pandas missing data is represented by two values:
- **None**: None is a Python singleton object that is often used for missing data in Python code.
- **NaN** : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation.
- There are different methods in place on how to detect missing values.

Pandas isnull() and notnull() functions

- Pandas offers two functions to test for missing data - `isnull()` and `notnull()`. These are simple functions that return a boolean value indicating whether the passed in argument value is in fact missing data.
- Below, I will list some useful commands to deal with missing values.

Useful commands to detect missing values

- **df.isnull()**

The above command checks whether each cell in a dataframe contains missing values or not. If the cell contains missing value, it returns True otherwise it returns False.

- **df.isnull().sum()**

The above command returns total number of missing values in each column in the dataframe.

- **df.isnull().sum().sum()**

It returns total number of missing values in the dataframe.

- **df.isnull().mean()**

It returns percentage of missing values in each column in the dataframe.

- **df.isnull().any()**

It checks which column has null values and which has not. The columns which has null values returns TRUE and FALSE otherwise.

- **df.isnull().any().any()**

It returns a boolean value indicating whether the dataframe has missing values or not. If dataframe contains missing values it returns TRUE and FALSE otherwise.

- **df.isnull().values.any()**

It checks whether a particular column has missing values or not. If the column contains missing values, then it returns TRUE otherwise FALSE.

- **df.isnull().values.sum()**

It returns the total number of missing values in the dataframe.

Interpretation

We can see that there are no missing values in the dataset.

Check with ASSERT statement

Back to Table of Contents

- We must confirm that our dataset has no missing values.
- We can write an **assert statement** to verify this.

- We can use an assert statement to programmatically check that no missing, unexpected 0 or negative values are present.
- This gives us confidence that our code is running properly.
- **Assert statement** will return nothing if the value being tested is true and will throw an AssertionError if the value is false.
- **Asserts**
  - assert 1 == 1 (return Nothing if the value is True)
  - assert 1 == 2 (return AssertionError if the value is False)

In [51]:
```
#assert that there are no missing values in the dataframe

assert pd.notnull(df).all().all()
```

In [52]:
```
#assert all values are greater than or equal to 0

assert (df >= 0).all().all()
```

Interpretation

- The above two commands do not throw any error. Hence, it is confirmed that there are no missing or negative values in the dataset.
- All the values are greater than or equal to zero.

Outlier detection

I will make boxplots to visualise outliers in the continuous numerical variables : -

`age` , `trestbps` , `chol` , `thalach` and `oldpeak` variables.

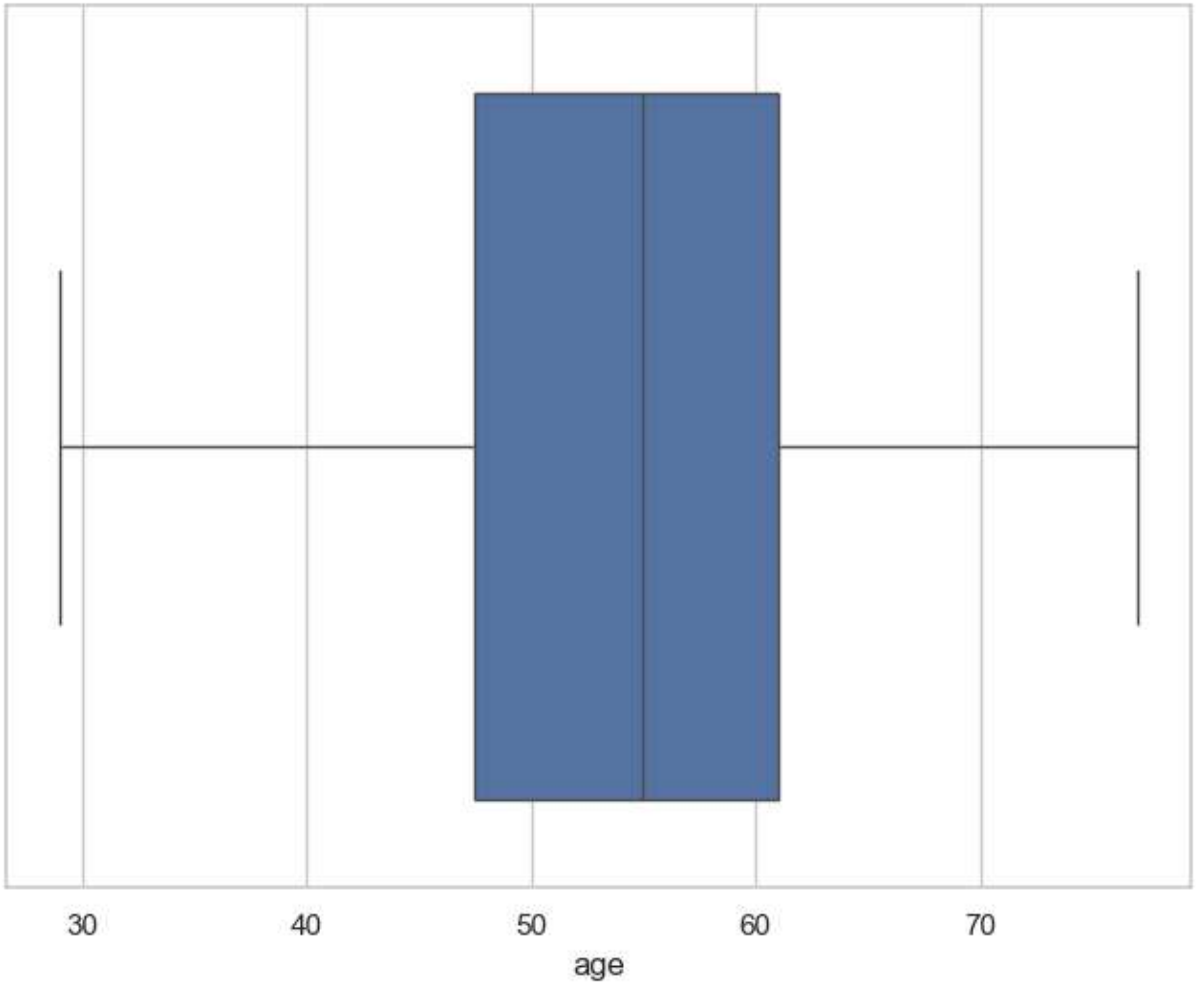
`age` variable

In [53]:
```
df['age'].describe()
```

Out[53]:
```
count    303.000000
mean      54.366337
std        9.082101
min       29.000000
25%       47.500000
50%       55.000000
75%       61.000000
max       77.000000
Name: age, dtype: float64
```

Box-plot of `age` variable

In [54]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["age"])
plt.show()
```



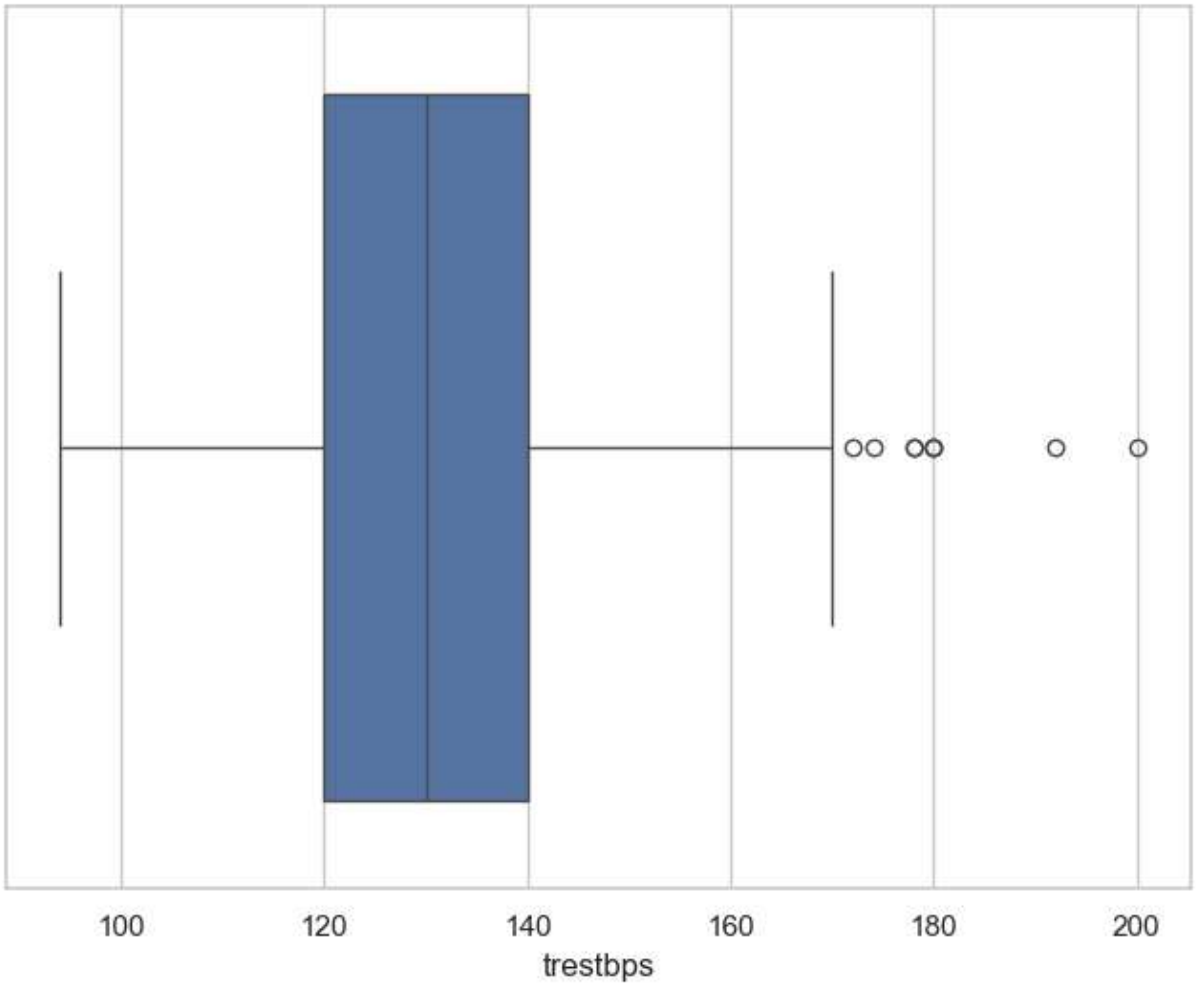
`trestbps` variable

In [55]:
```
df['trestbps'].describe()
```

Out[55]:
```
count    303.000000
mean     131.623762
std       17.538143
min       94.000000
25%      120.000000
50%      130.000000
75%      140.000000
max      200.000000
Name: trestbps, dtype: float64
```

Box-plot of `trestbps` variable

In [56]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["trestbps"])
plt.show()
```



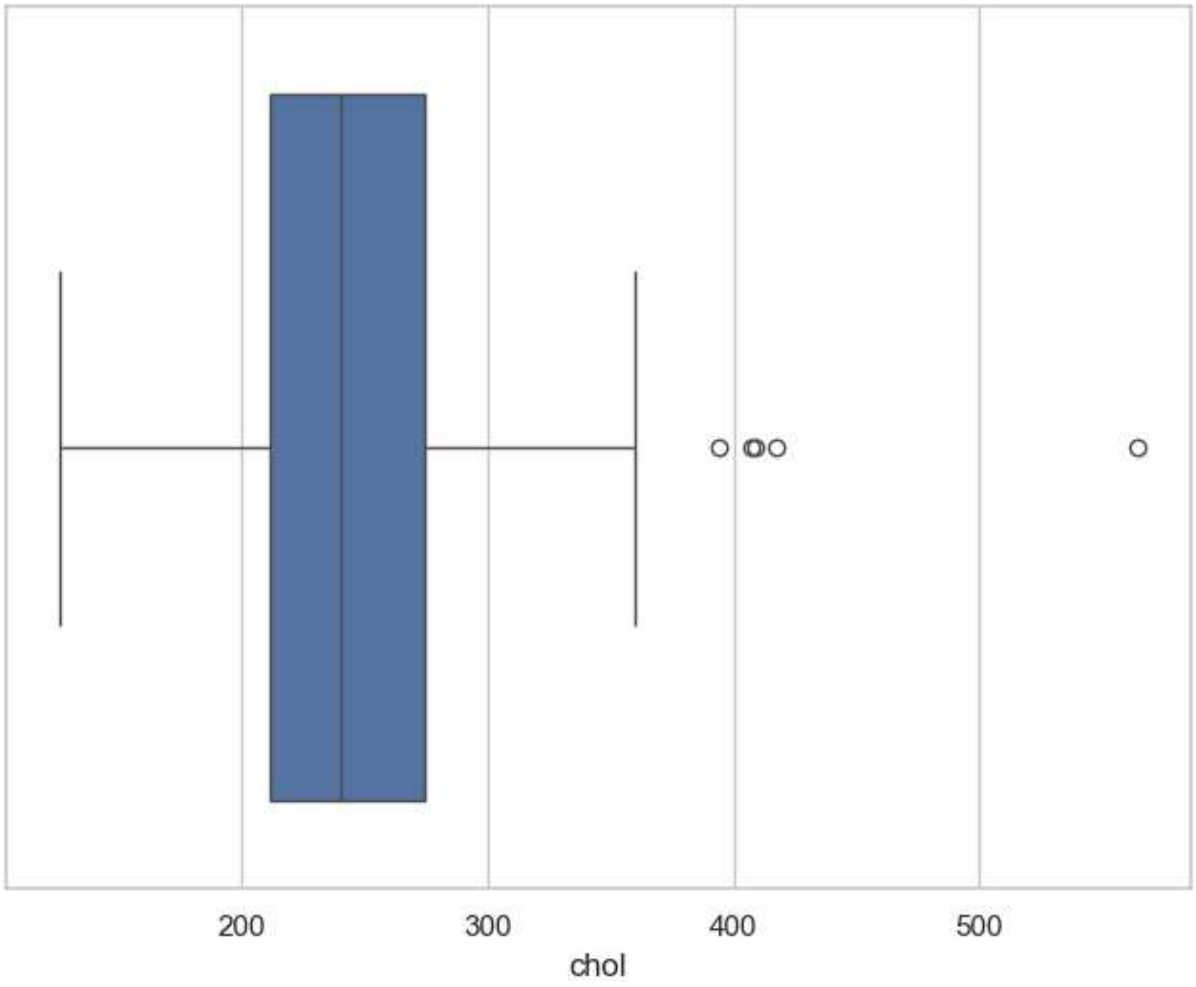
Chol variable

In [57]:
```
df['chol'].describe()
```

Out[57]:
```
count    303.000000
mean     246.264026
std       51.830751
min      126.000000
25%      211.000000
50%      240.000000
75%      274.500000
max      564.000000
Name: chol, dtype: float64
```

Box-plot of chol variable

In [58]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["chol"])
plt.show()
```



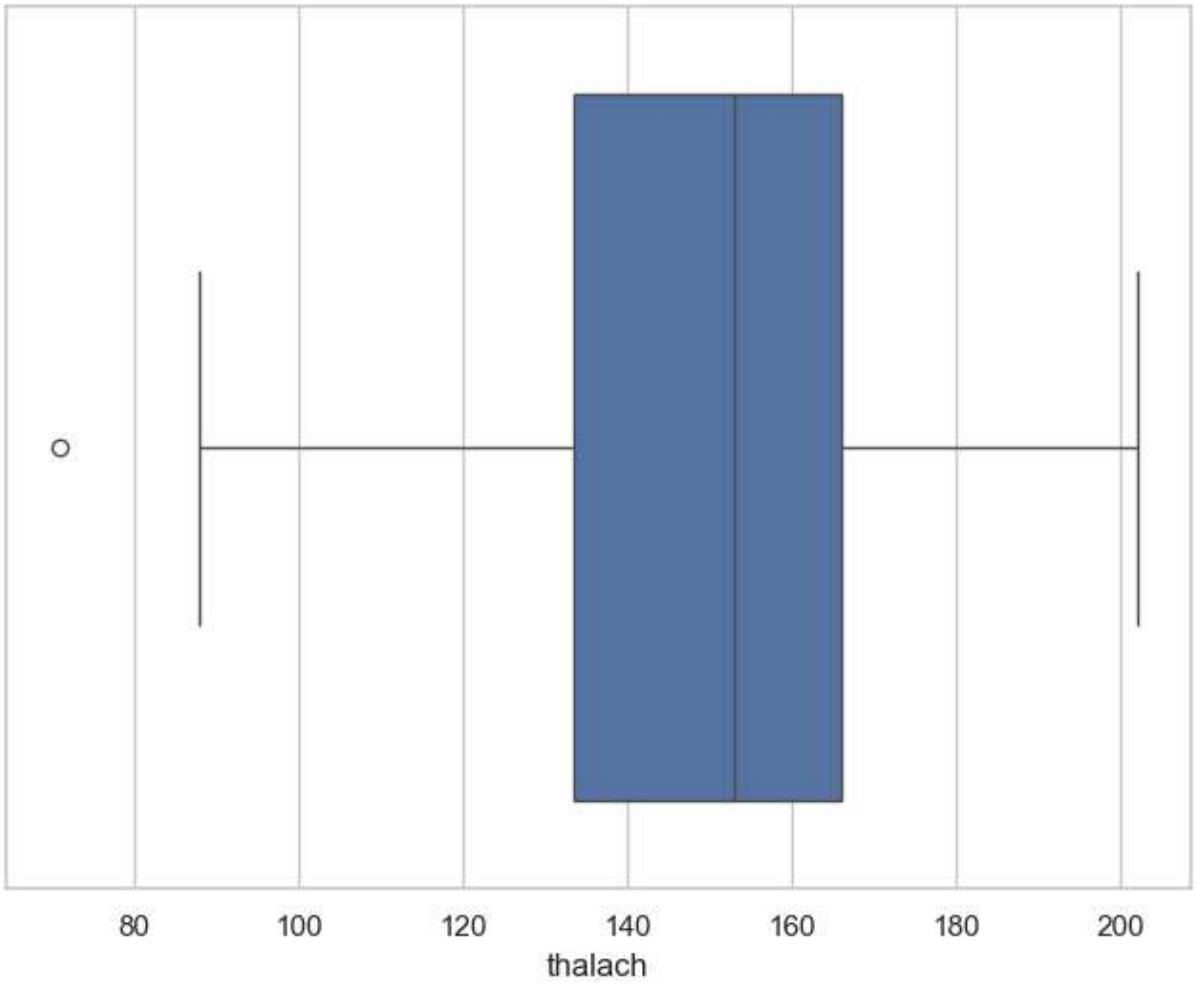
thalach variable

In [59]:
```
df['thalach'].describe()
```

Out[59]:
```
count    303.000000
mean     149.646865
std       22.905161
min       71.000000
25%      133.500000
50%      153.000000
75%      166.000000
max      202.000000
Name: thalach, dtype: float64
```

Box-plot of `thalach` variable

In [60]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["thalach"])
plt.show()
```



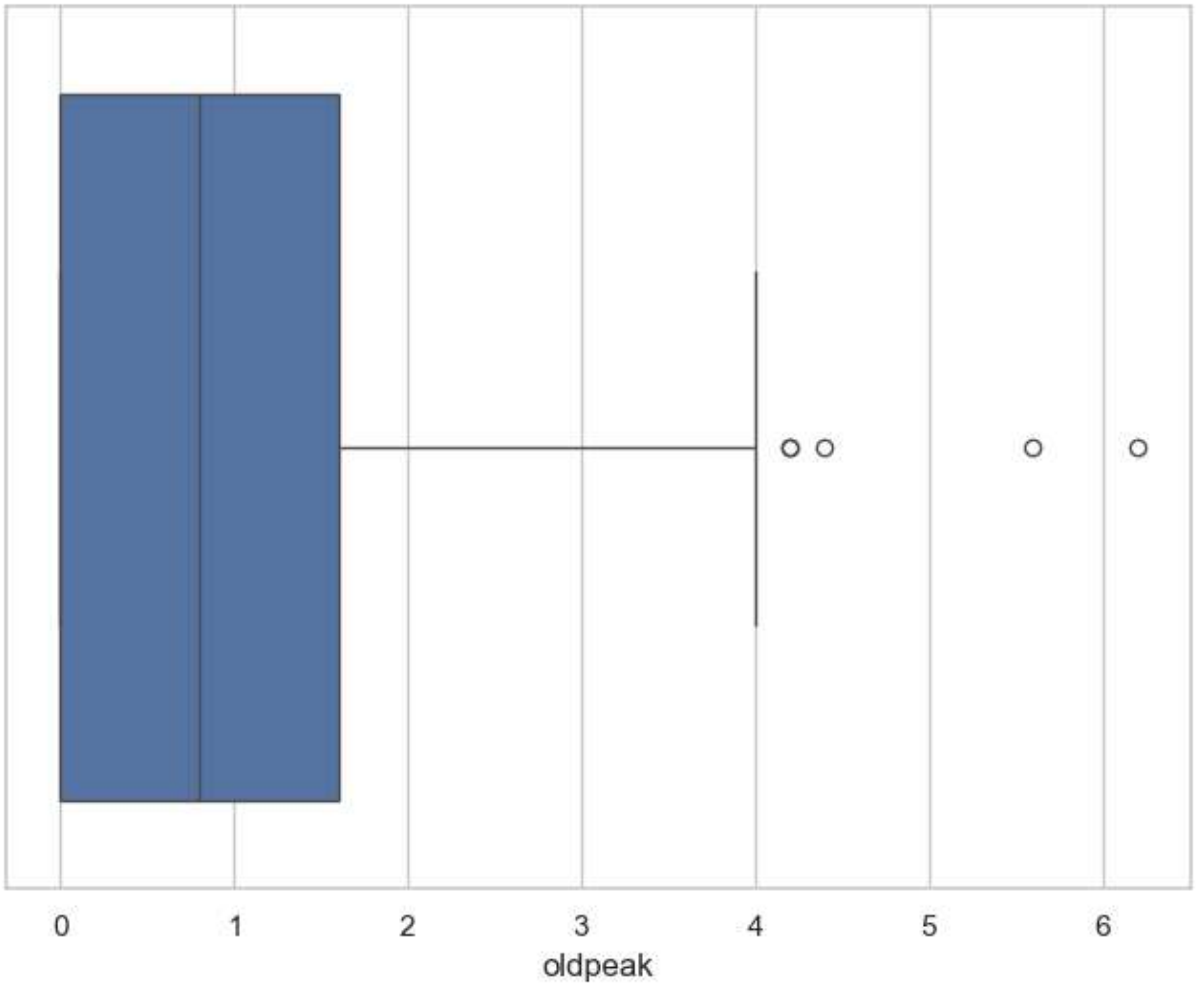
`oldpeak` variable

In [61]:
```
df['oldpeak'].describe()
```

Out[61]:
```
count    303.000000
mean       1.039604
std        1.161075
min        0.000000
25%        0.000000
50%        0.800000
75%        1.600000
max        6.200000
Name: oldpeak, dtype: float64
```

Box-plot of `oldpeak` variable

In [62]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["oldpeak"])
plt.show()
```



Findings

- The `age` variable does not contain any outlier.
- `trestbps` variable contains outliers to the right side.
- `chol` variable also contains outliers to the right side.
- `thalach` variable contains a single outlier to the left side.
- `oldpeak` variable contains outliers to the right side.
- Those variables containing outliers needs further investigation.

13. Conclusion

So, friends, our EDA journey has come to an end.

In this kernel, we have explored the heart disease dataset. In this kernel, we have implemented many of the strategies presented in the book **Think Stats - Exploratory Data Analysis in Python by Allen B Downey** . The feature variable of interest is `target`

variable. We have analyzed it alone and check its interaction with other variables. We have also discussed how to detect missing data and outliers.

I hope you like this kernel on EDA journey.

Thanks

In [ ]: