# Extensive Analysis + Visualization with Pytho

- Analysing Exploratory Data Analysis or EDA of the heart disease dataset.
- Exploratory Data Analysis or EDA is a critical first step in analyzing a new dataset.
- The primary objective of EDA is to analyze the data for distribution, outliers and anomalies in the dataset.
- It enable us to direct specific testing of the hypothesis.
- It includes analysing the data to find the distribution of data, its main characteristics, identifying patterns and visualizations.
- It also provides tools for hypothesis generation by visualizing and understanding the data through graphical representation.

# Objectives of EDA

The objectives of the EDA are as follows:-

- To get an overview of the distribution of the dataset.
- Check for missing numerical values, outliers or other anomalies in the dataset.
- Discover patterns and relationships between variables in the dataset.
- Check the underlying assumptions in the dataset.

# Types of EDA

- EDA is generally cross-classified in two ways. First, each method is either non-graphical or graphical.
- Second, each method is either univariate or multivariate (usually bivariate).
- The non-graphical methods provide insight into the characteristics and the distribution of the variable(s) of interest.
- So, non-graphical methods involve calculation of summary statistics while graphical methods include summarizing the data diagrammatically.
- There are four types of exploratory data analysis (EDA) based on the above cross-classification methods.
- Each of these types of EDA are described below:-

Univariate non-graphical EDA:

- The objective of the univariate non-graphical EDA is to understand the sample distribution and also to make some initial conclusions about population distributions. Outlier detection is also a part of this analysis.

Multivariate non-graphical EDA

- Multivariate non-graphical EDA techniques show the relationship between two or more variables in the form of either cross-tabulation or statistics.

Univariate graphical EDA

- In addition to finding the various sample statistics of univariate distribution (discussed above), we also look graphically at the distribution of the sample. The non-graphical methods are quantitative and objective. They do not give full picture of the data. Hence, we need graphical methods, which are more qualitative in nature and presents an overview of the data.

Multivariate graphical EDA

- There are several useful multivariate graphical EDA techniques, which are used to look at the distribution of multivariate data. These are as follows:-
  - Side-by-Side Boxplots
  - Scatterplots
  - Heat Maps and 3-D Surface Plots

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st

#sns.set(style='whitegrid')
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_csv('heart.csv')
df
```

Out[3]:

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

303 rows × 14 columns

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

Out[4]:

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

In [5]:
```
df.shape
```

Out[5]:
```
(303, 14)
```

In [6]:
```
#Data Type Summery
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 [7]:
```
#Check datatype of each coloumn
df.dtypes
```

Out[7]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

# Important points about dataset

- sex is a character variable. Its data type should be object. But it is encoded as (1 = male; 0 = female). So, its data type is given as int64.
- Same is the case with several other variables - fbs, exang and target.
- fbs (fasting blood sugar) should be a character variable as it contains only 0 and 1 as values (1 = true; 0 = false). As it contains only 0 and 1 as values, so its data type is given as int64.
- exang (exercise induced angina) should also be a character variable as it contains only 0 and 1 as values (1 = yes; 0 = no). It also contains only 0 and 1 as values, so its data type is given as int64.
- target should also be a character variable. But, it also contains 0 and 1 as values. So, its data type is given as int64.

# describe()

- df.describe() helps us to view the statistical properties of numerical variables. It excludes character variables.
- If we want to view the statistical properties of character variables, we should run the following command -
  - df.describe(include=['object'])
- If we want to view the statistical properties of all the variables, we should run the following command -
  - df.describe(include='all')

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

Out[8]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 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 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 |

In [9]:
```
df.describe(include='all')
```

Out[9]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 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 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 |

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

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

# Univariate analysis

# Analysis of target feature variable

- Our feature variable of interest is target.
- It refers to the presence of heart disease in the patient.
- It is integer valued as it contains two integers 0 and 1 - (0 stands for absence of heart disease and 1 for presence of heart disease).

# Check the number of unique values in target variable

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

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

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

Out[12]:
```
array([1, 0], dtype=int64)
```

So, the unique values are 1 and 0. (1 stands for presence of heart disease and 0 for absence of hear disease).

# Frequency distribution of target variable

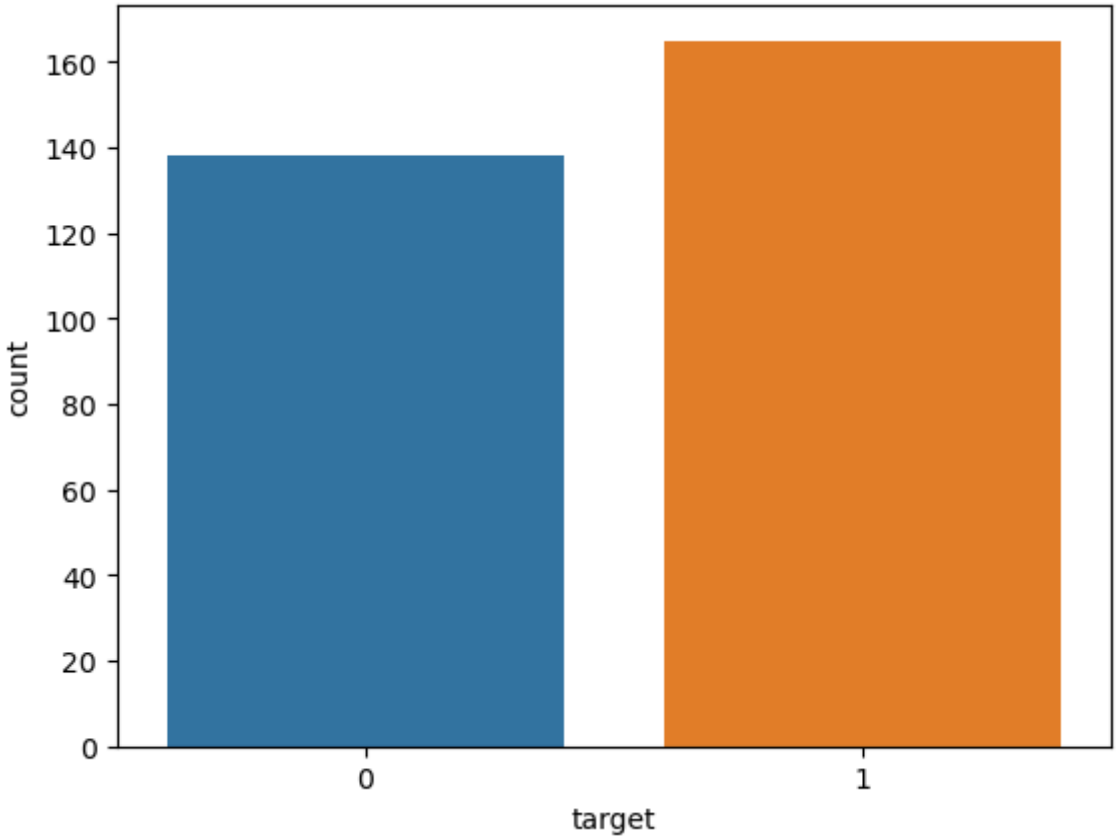
In [13]:
```
df['target'].value_counts()
```

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

- 1 stands for presence of heart disease. So, there are 165 patients suffering from heart disease.
- Similarly, 0 stands for absence of heart disease. So, there are 138 patients who do not have any heart disease.

## Visualize frequency distribution of target variable

In [14]:
```
f,ax = plt.subplots()
ax = sns.countplot(x='target',data=df)
```




## Frequency distribution of target variable wrt sex

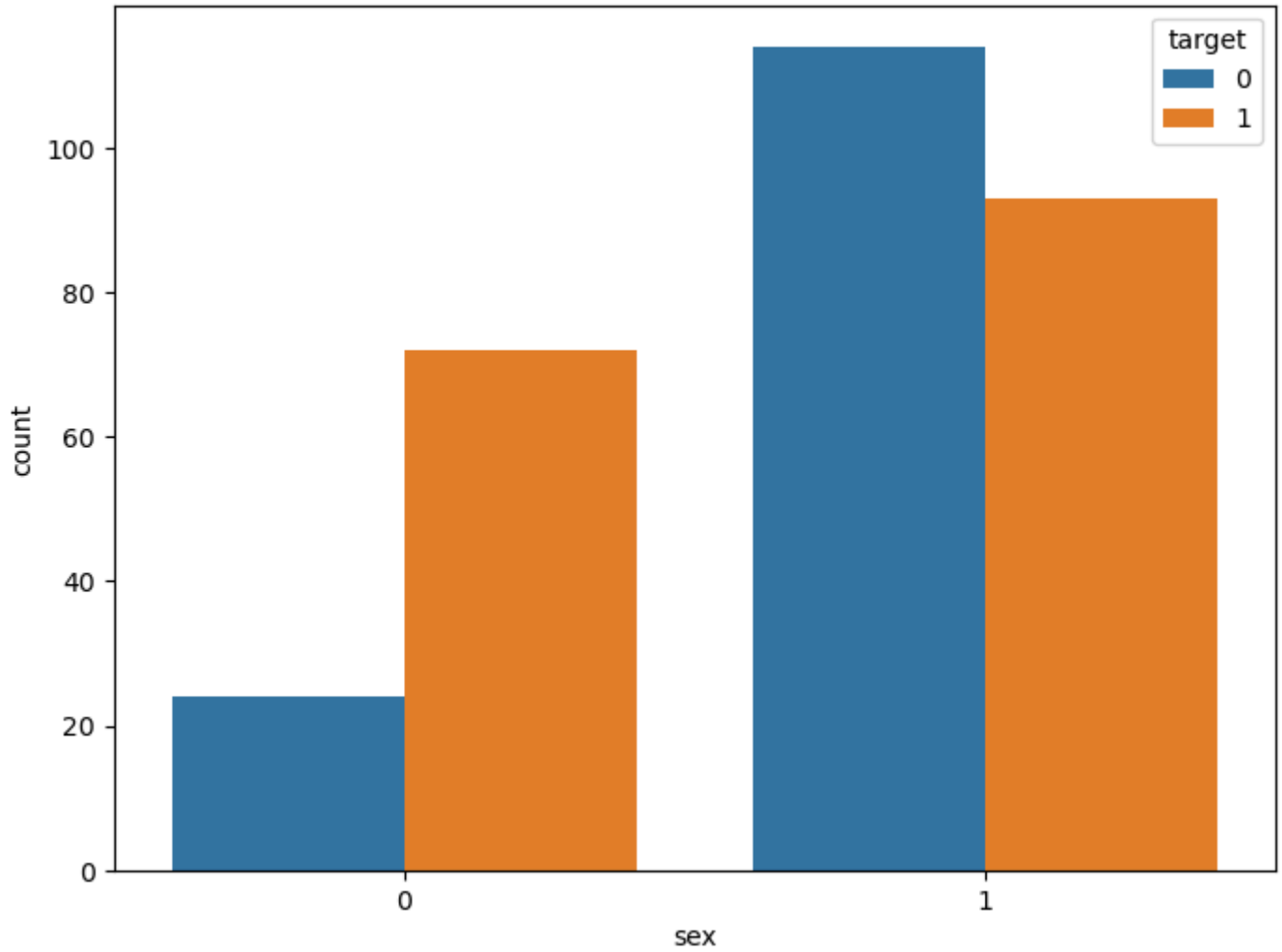
In [15]:
```
df.groupby('sex')['target'].value_counts()
```

Out[15]:
```
sex  target
0    1          72
     0          24
1    0         114
     1          93
Name: count, dtype: int64
```

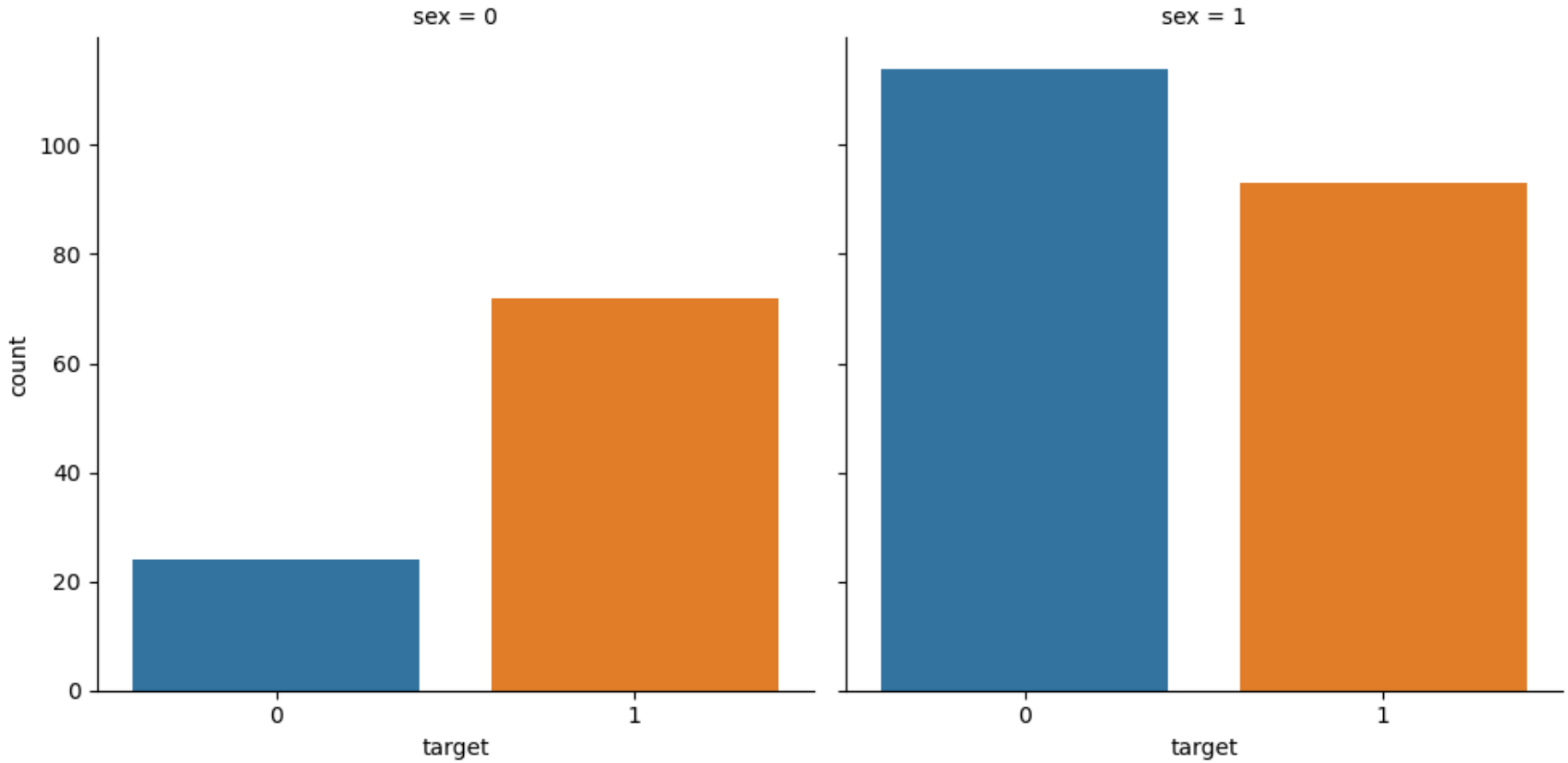
- sex variable contains two integer values 1 and 0 : (1 = male; 0 = female).
- target variable also contains two integer values 1 and 0 : (1 = Presence of heart disease; 0 = Absence of heart disease)
- So, out of 96 females - 72 have heart disease and 24 do not have heart disease.
- Similarly, out of 207 males - 93 have heart disease and 114 do not have heart disease.
- We can visualize this information below.

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

In [16]:
```
f, ax = plt.subplots(figsize=(8, 6))
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=5,aspect=1)
```



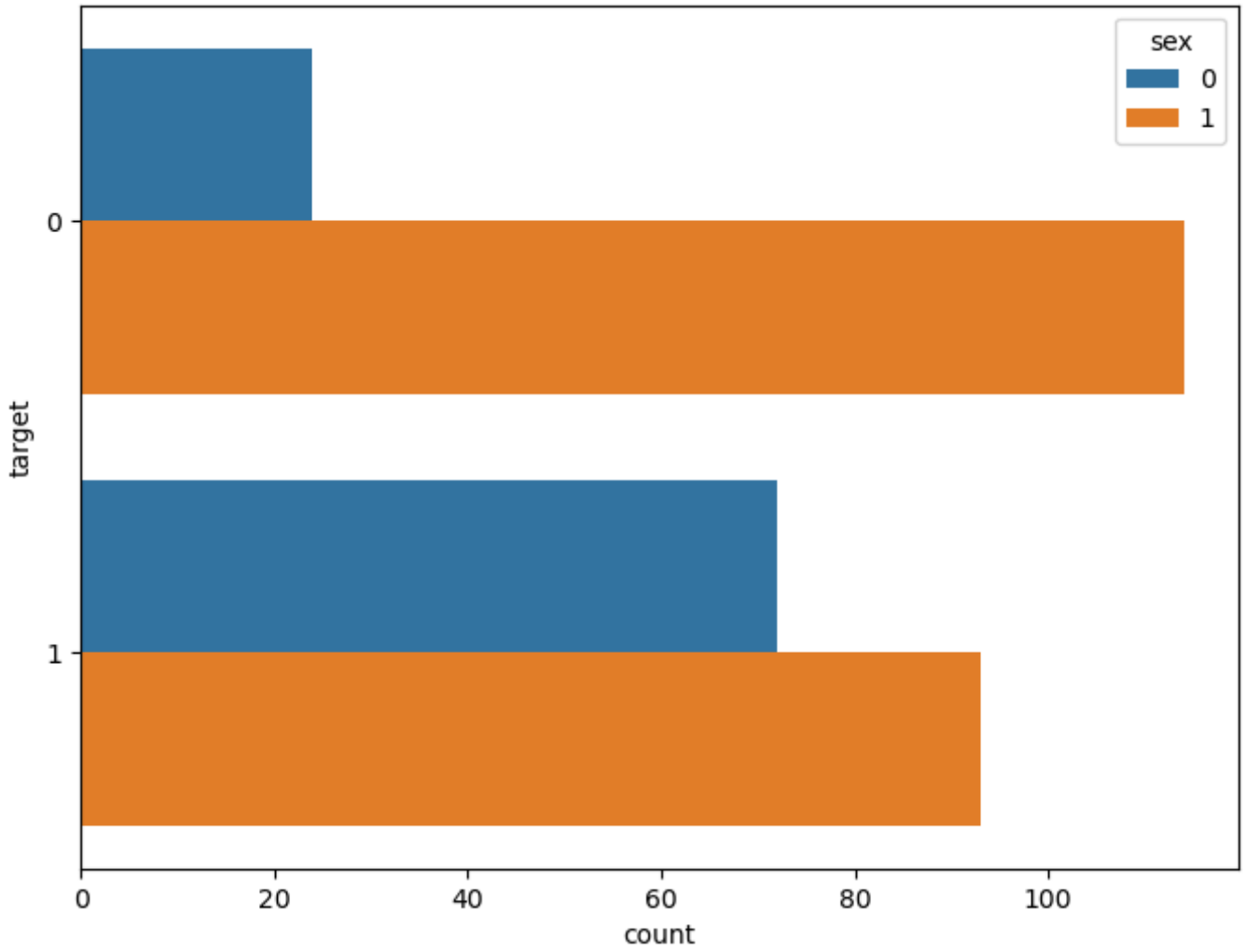

The above plot segregate the values of target variable and plot on two different columns labelled as (sex = 0, sex = 1)

# countplot()

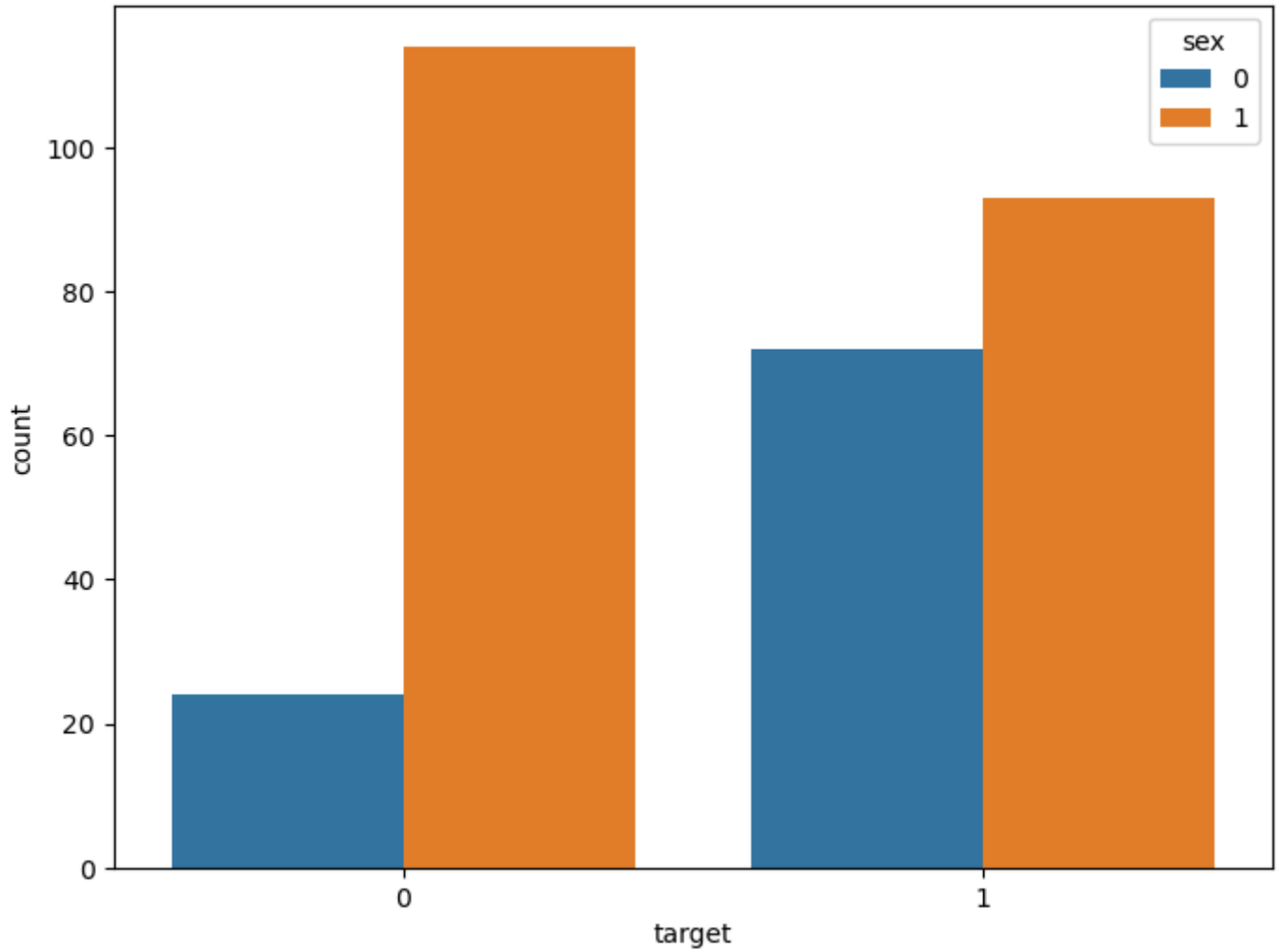
seaborn.countplot() method is used to Show the counts of observations in each categorical bin using bars.

# Horizantal plot

In [18]:
```
#Syntax : seaborn.countplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, orient=None, color=None, palette=Non
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(y='target',hue='sex',data=df) # Horuzantal plot based on given axes
```




In [19]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='target',hue='sex',data=df) # vertical plot based on given axes
```



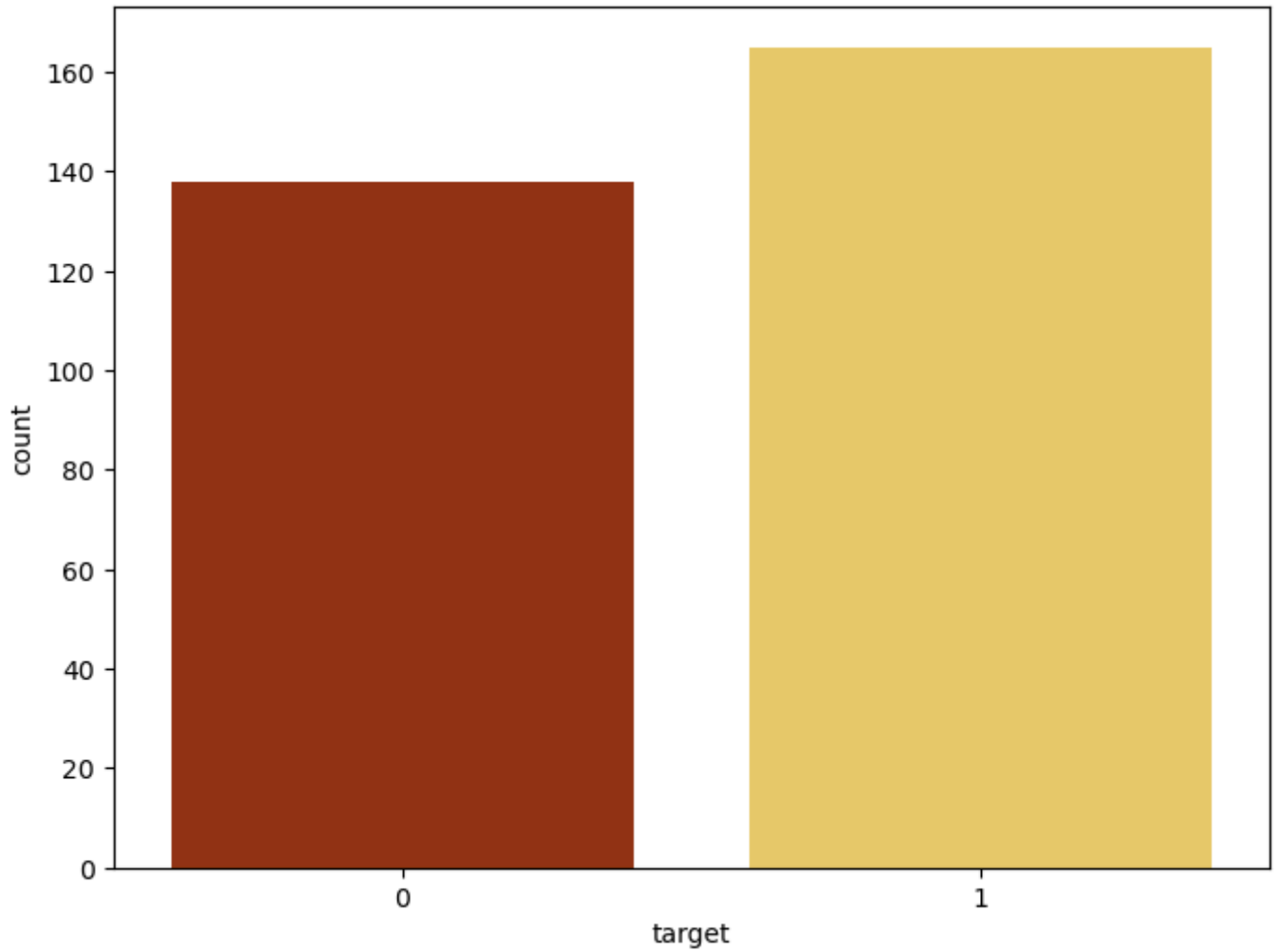

# Different colour paletter

Possible values of palette are:

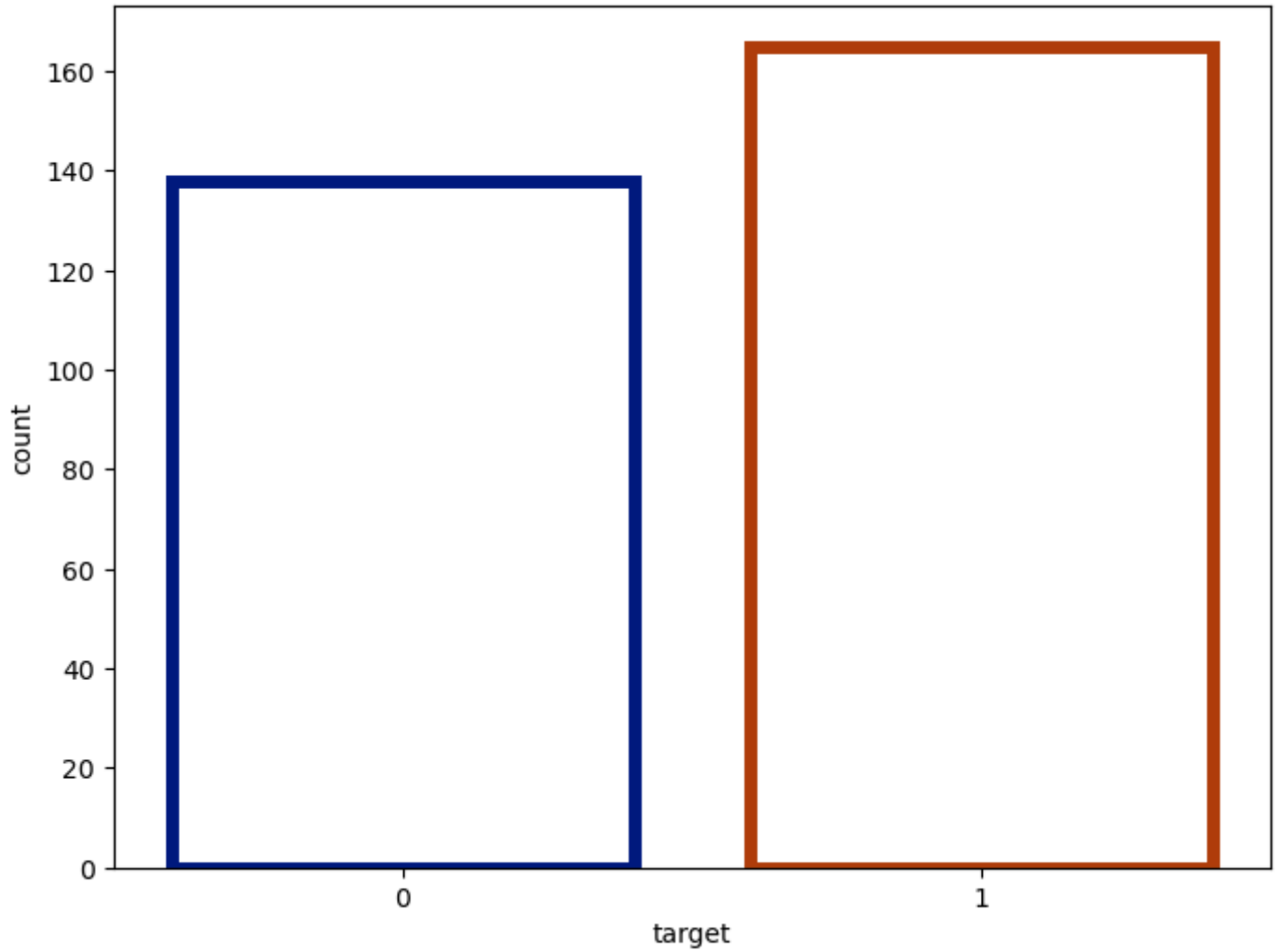
Accent, Accent_r, Blues, Blues_r, BrBG, BrBG_r, BuGn, BuGn_r, BuPu, BuPu_r, CMRmap, CMRmap_r, Dark2, Dark2_r, GnBu, GnBu_r, Greens, Greens_r, Greys, Greys_r, OrRd, OrRd_r, Oranges, Oranges_r, PRGn, PRGn_r, Paired, Paired_r, Pastel1, Pastel1_r, Pastel2, Pastel2_r, PiYG, PiYG_r, PuBu, PuBuGn, PuBuGn_r, PuBu_r, PuOr, PuOr_r, PuRd, PuRd_r, Purples, Purples_r, RdBu, RdBu_r, RdGy, RdGy_r, RdPu, RdPu_r, RdYlBu, RdYlBu_r, RdYlGn, RdYlGn_r,

Reds, Reds_r, Set1, Set1_r, Set2, Set2_r, Set3, Set3_r, Spectral, Spectral_r, Wistia, Wistia_r, YlGn, YlGnBu, YlGnBu_r, YlGn_r, YlOrBr, YlOrBr_r, YlOrRd, YlOrRd_r, afmhot, afmhot_r, autumn, autumn_r, binary, binary_r, bone, bone_r, brg, brg_r, bwr, bwr_r, cividis, cividis_r, cool, cool_r, coolwarm, coolwarm_r, copper, copper_r, cubehelix, cubehelix_r, flag, flag_r, gist_earth, gist_earth_r, gist_gray, gist_gray_r, gist_heat, gist_heat_r, gist_ncar, gist_ncar_r, gist_rainbow, gist_rainbow_r, gist_stern,

In [20]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='target',data=df,palette='afmhot')
```

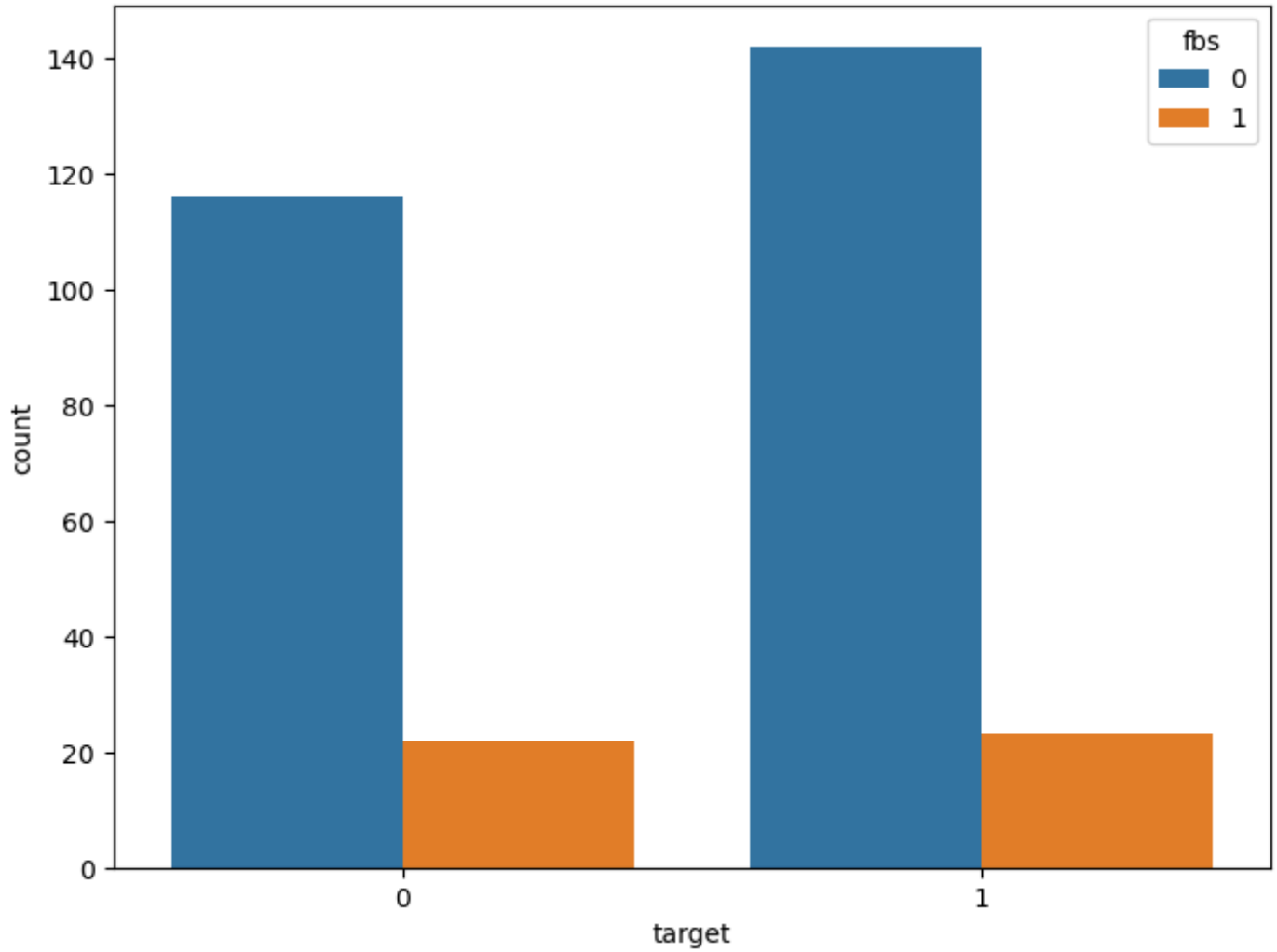


In [21]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='target',data=df,facecolor=(0,0,0,0),linewidth=5,edgecolor=sns.color_palette('dark',3))
#sns.color_palette ("deep", "muted", "bright", "dark", "colorblind")
```

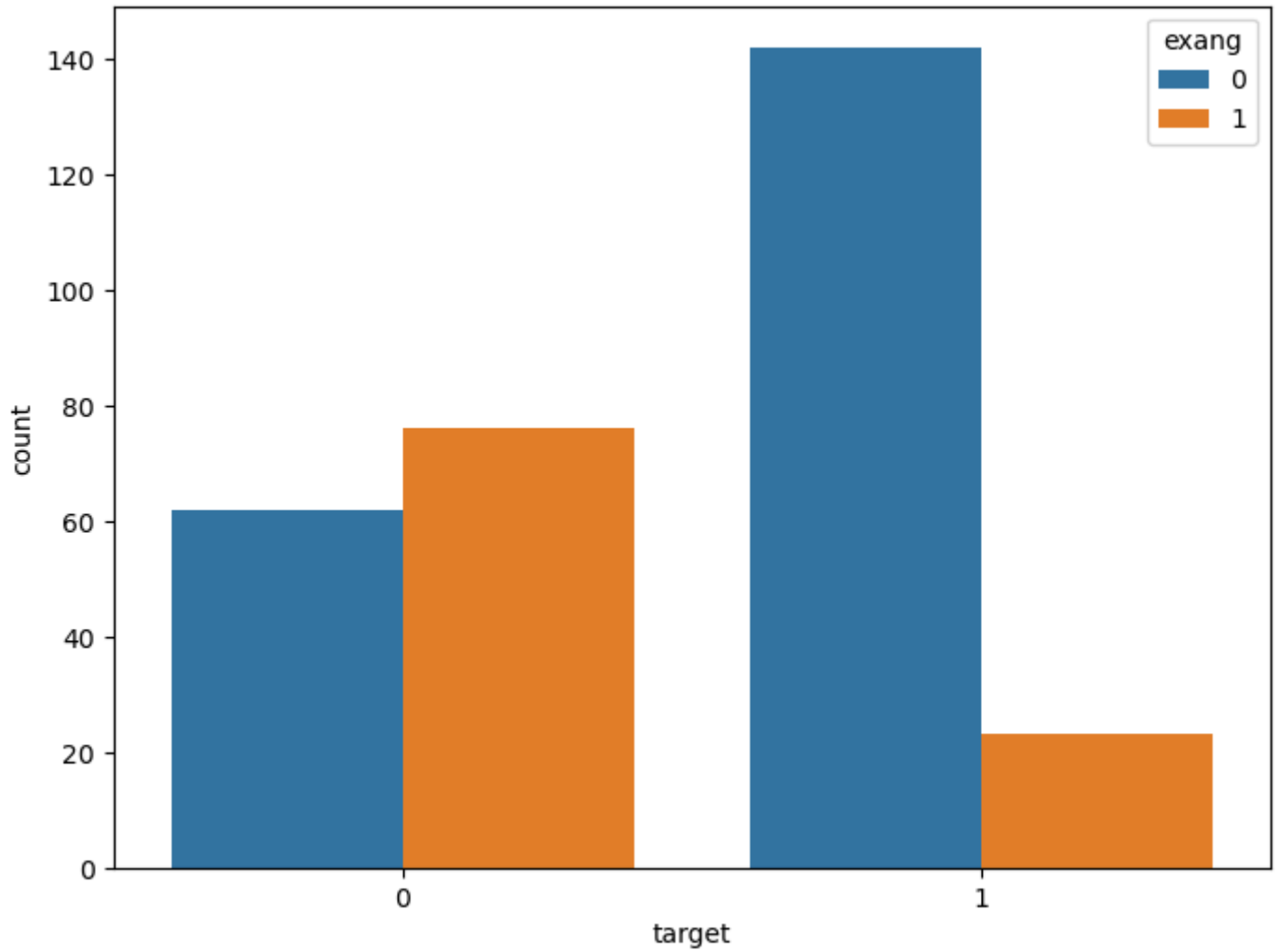



## visualize the target values distribution wrt fbs (fasting blood sugar) and exang (exercise induced angina)

In [22]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='target',hue='fbs',data=df)
```




In [23]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='target',hue='exang',data=df)
```




# Findings of Univariate Analysis

Findings of univariate analysis are as follows:-

- Our feature variable of interest is `target`.
- It refers to the presence of heart disease in the patient.

- It is integer valued as it contains two integers 0 and 1 - (0 stands for absence of heart disease and 1 for presence of heart disease).
- 1 stands for presence of heart disease. So, there are 165 patients suffering from heart disease.
- Similarly, 0 stands for absence of heart disease. So, there are 138 patients who do not have any heart disease.
- There are 165 patients suffering from heart disease, and
- There are 138 patients who do not have any heart disease.
- Out of 96 females - 72 have heart disease and 24 do not have heart disease.
- Similarly, out of 207 males - 93 have heart disease and 114 do not have heart disease.

# Bivariate Analysis

# Estimate correlation coefficients

- Our dataset is very small. So, compute the standard correlation coefficient (also called Pearson's r) between every pair of attributes.
- Compute it using the df.corr() method as follows:

In [24]:
```
correlation = df.corr()
```

The target variable is target. So, we should check how each attribute correlates with the target variable.

In [25]:
```
correlation['target'].sort_values(ascending=False)
```

Out[25]:
```
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
```

Interpretation of correlation coefficient

- The correlation coefficient ranges from -1 to +1.
- When it is close to +1, this signifies that there is a strong positive correlation. So, we can see that there is no variable which has strong positive correlation with target variable.
- When it is clsoe to -1, it means that there is a strong negative correlation. So, we can see that there is no variable which has strong negative correlation with target variable.
- When it is close to 0, it means that there is no correlation. So, there is no correlation between target and fbs.
- We can see that the cp and thalach variables are mildly positively correlated with target variable. So, I will analyze the interaction between these features and target variable.

# Analysis of target and cp variable

Find unique values in cp variable

In [26]:
```
df['cp'].nunique()
```

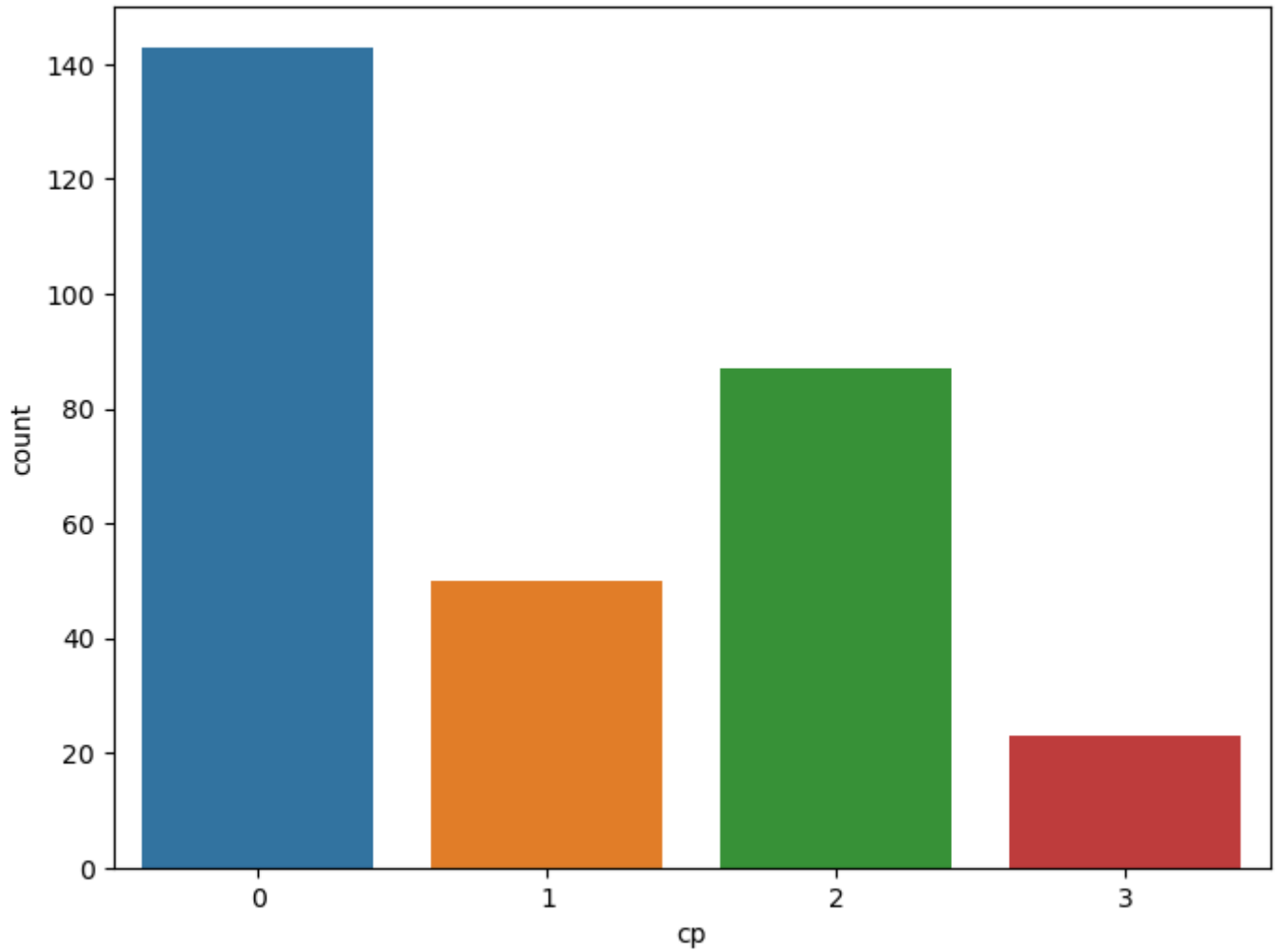
Out[26]:
```
4
```

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

Out[27]:
```
cp
0    143
2     87
1     50
3     23
Name: count, dtype: int64
```

# Visualize the frequency distribution of cp variable

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




## Frequency distribution of target variable wrt cp

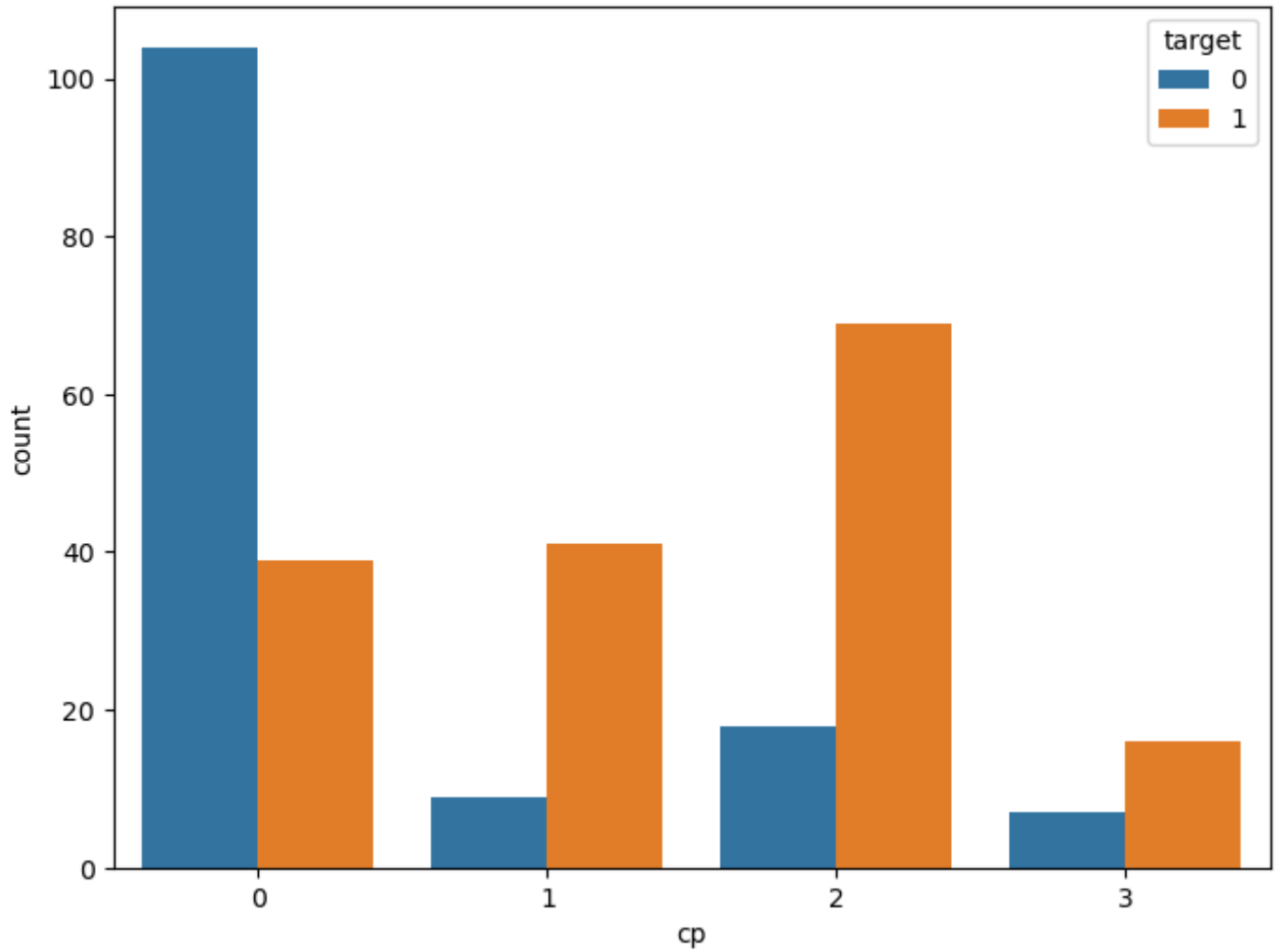
In [29]:
```
df.groupby('cp')['target'].value_counts()
```

Out[29]:
```
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
```

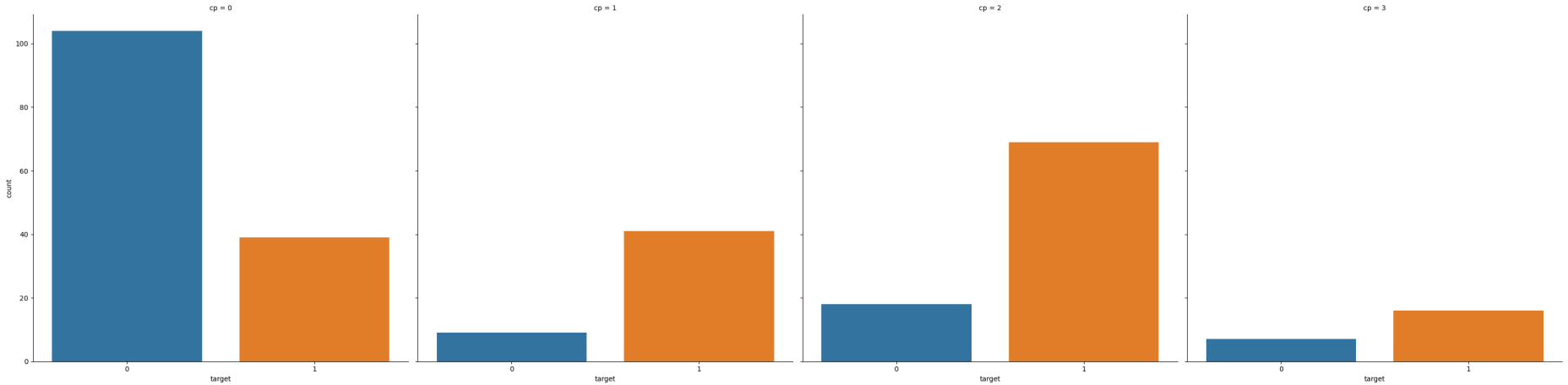
- cp variable contains four integer values 0, 1, 2 and 3.
- target variable contains two integer values 1 and 0 : (1 = Presence of heart disease; 0 = Absence of heart disease)
- So, the above analysis gives target variable values categorized into presence and absence of heart disease and groupby cp variable values.

We can visualize the value counts of the cp variable wrt target

In [30]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='cp',hue='target',data=df)
```




In [31]:
```
ax = sns.catplot(x='target',col='cp',data=df,kind='count',height=8,aspect=1)
```




# Analysis of target and thalach variable

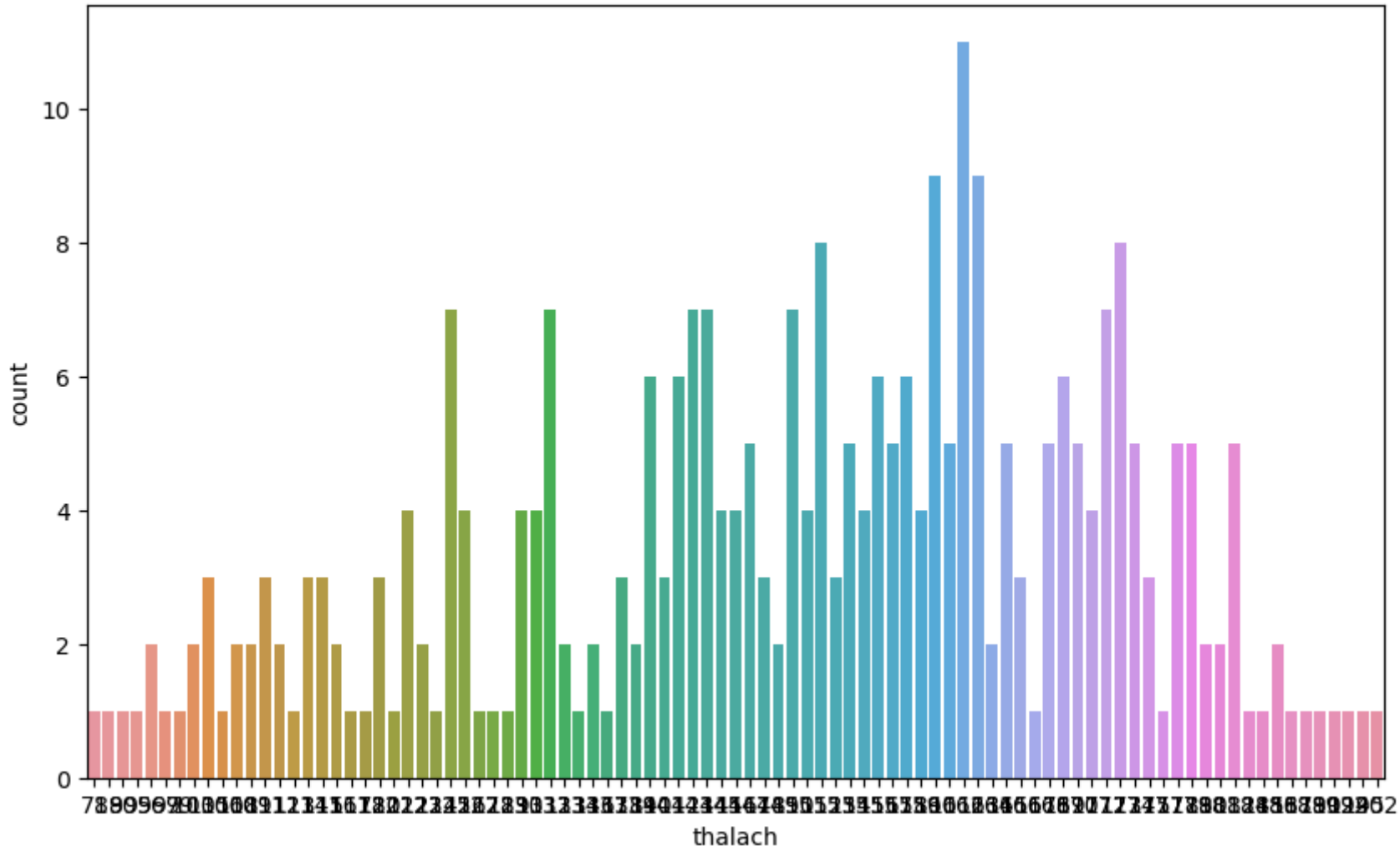
thalach stands for maximum heart rate achived

In [32]:
```
df['thalach'].nunique()
```

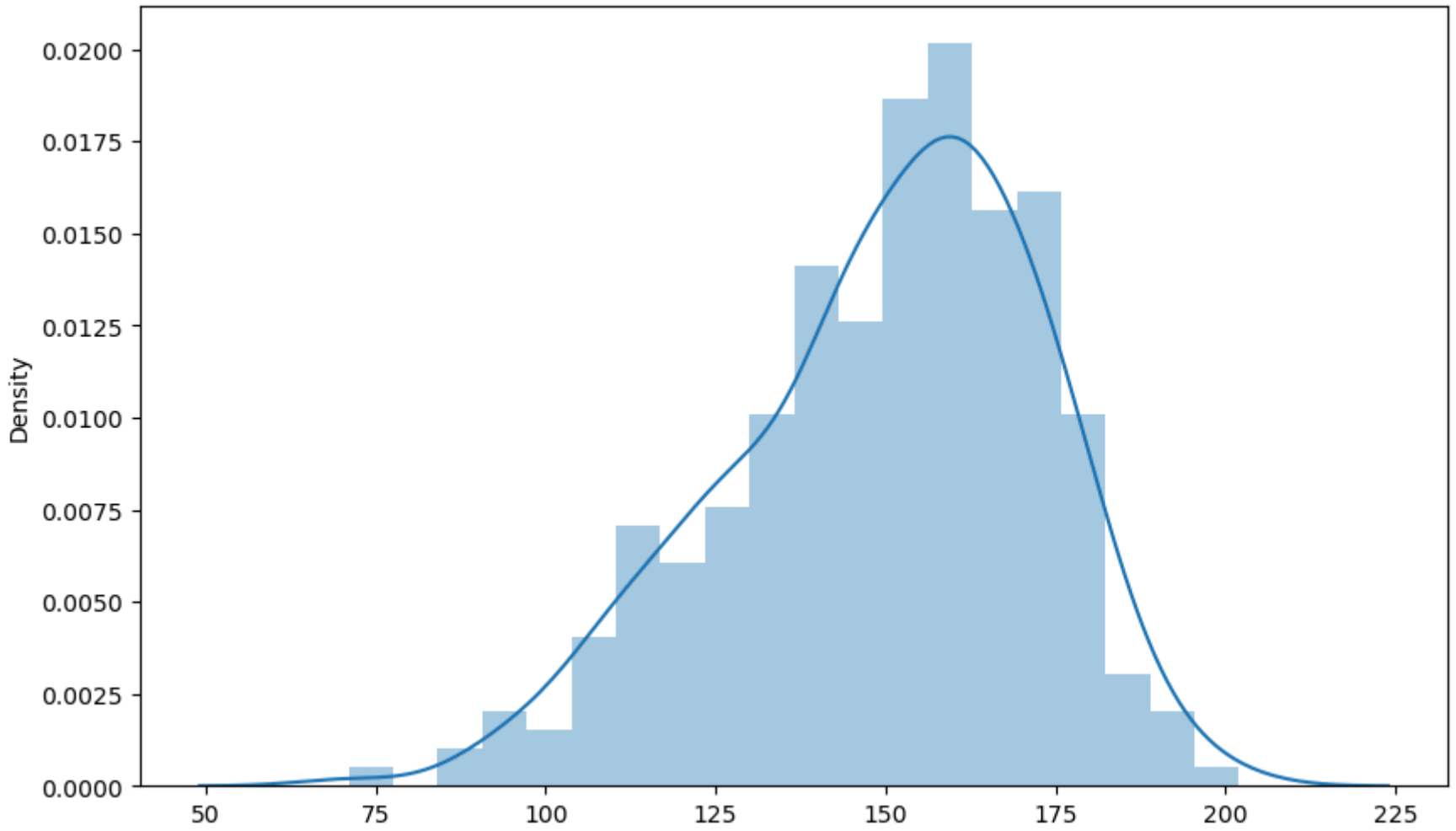
Out[32]:
```
91
```

# Visualize the frequency distribution of thalach variable

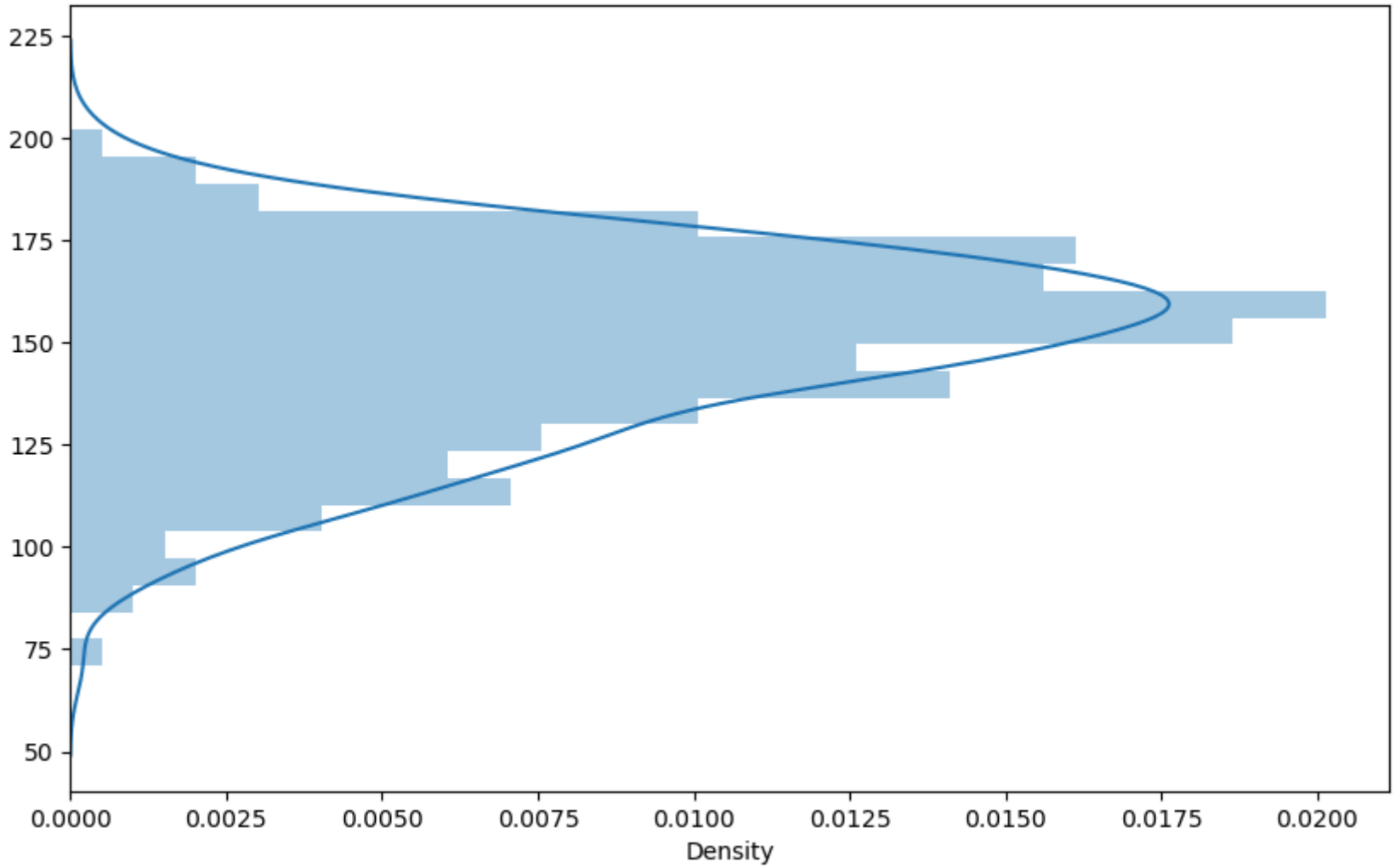
In [33]:
```
f,ax = plt.subplots(figsize=(10,6))
ax = sns.countplot(x='thalach',data=df)
```




In [34]:
```
f,ax = plt.subplots(figsize=(10,6))
ax = sns.distplot(x=df['thalach'],bins=20)
```




In [35]:
```
f,ax = plt.subplots(figsize=(10,6))
ax = sns.distplot(x=df['thalach'],bins=20,vertical=True)
```




# Seaborn Kernel Density Estimation (KDE) Plot

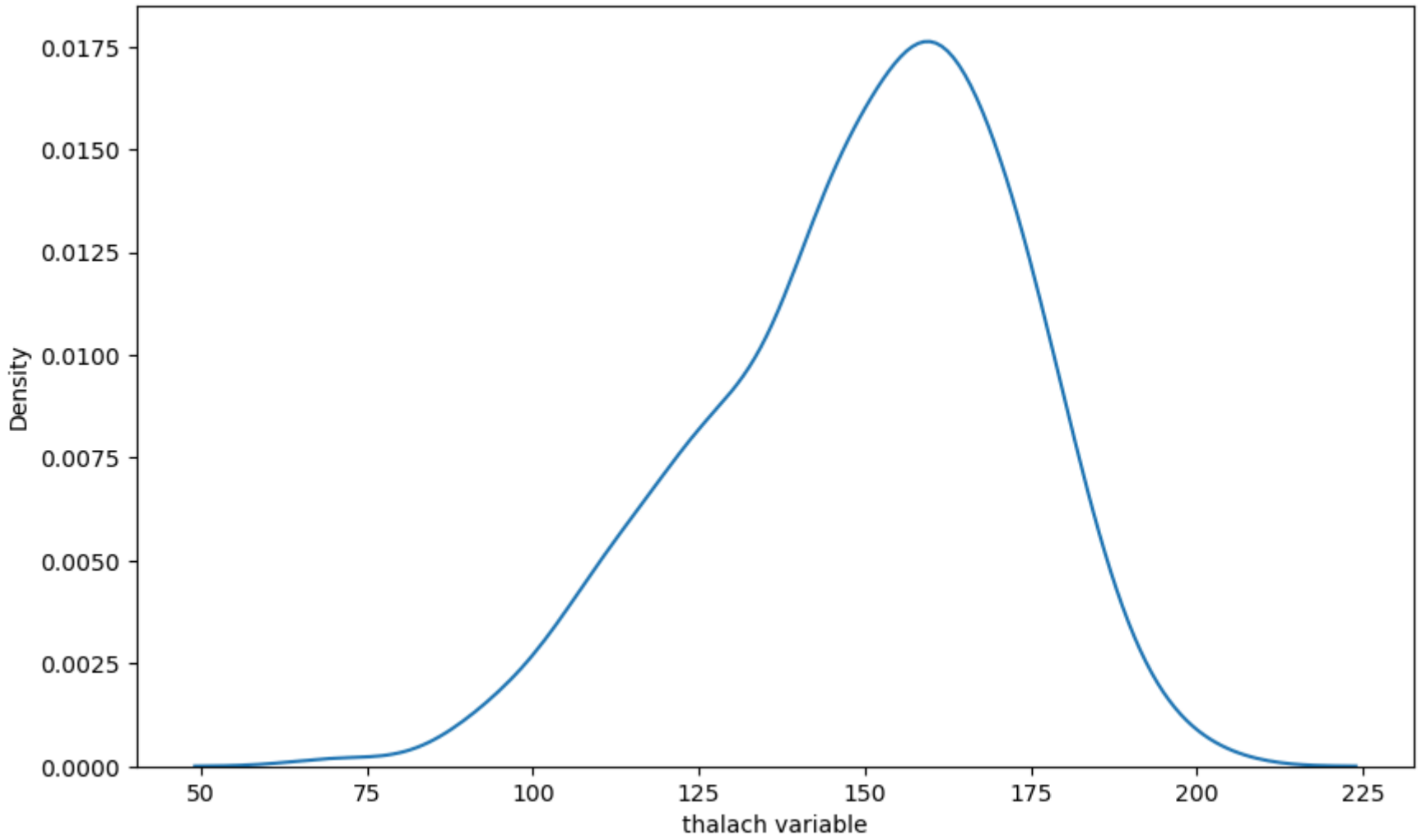
- The kernel density estimate (KDE) plot is a useful tool for plotting the shape of a distribution.
- The KDE plot plots the density of observations on one axis with height along the other axis.

In [36]:
```
df.head()
```

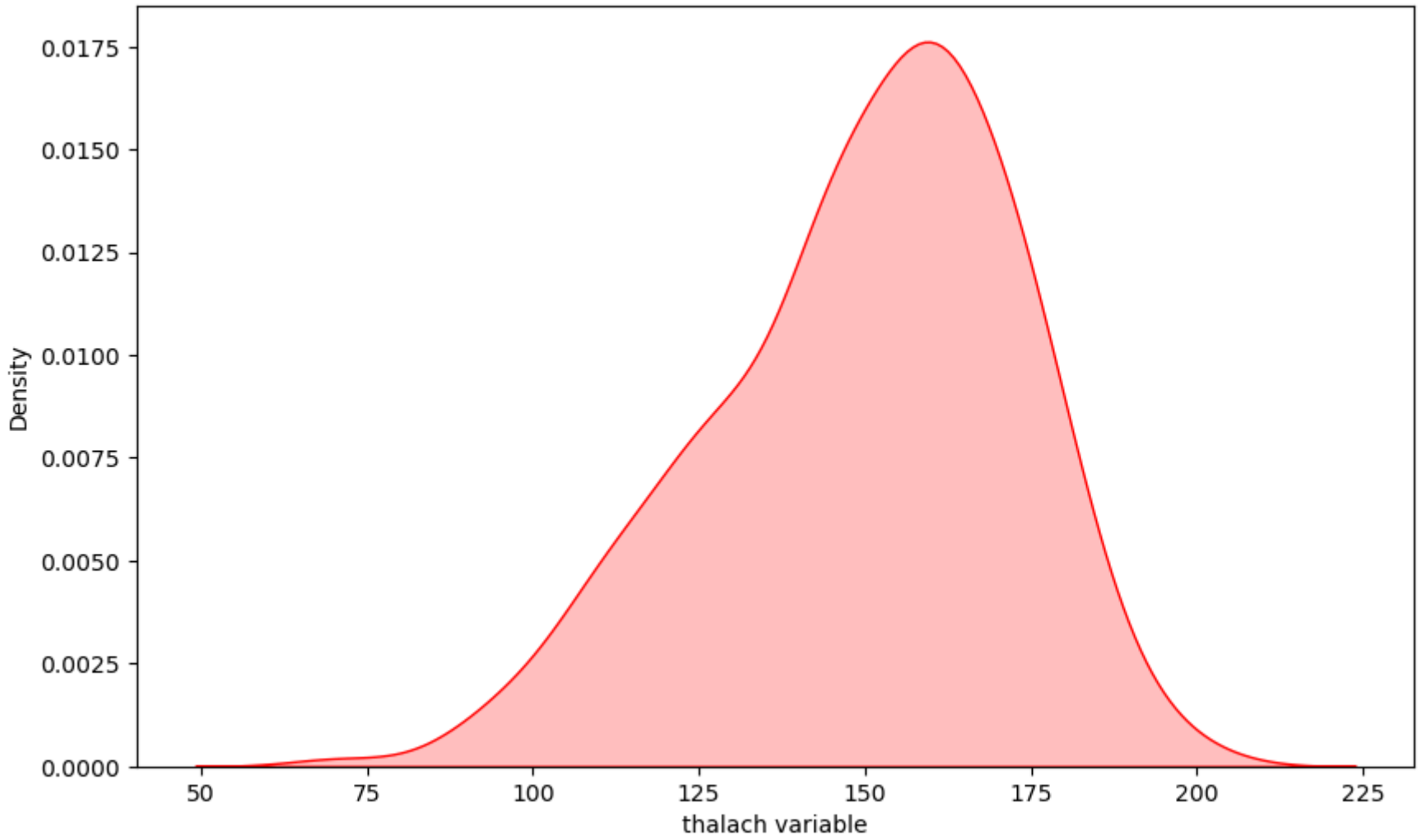
Out[36]:

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

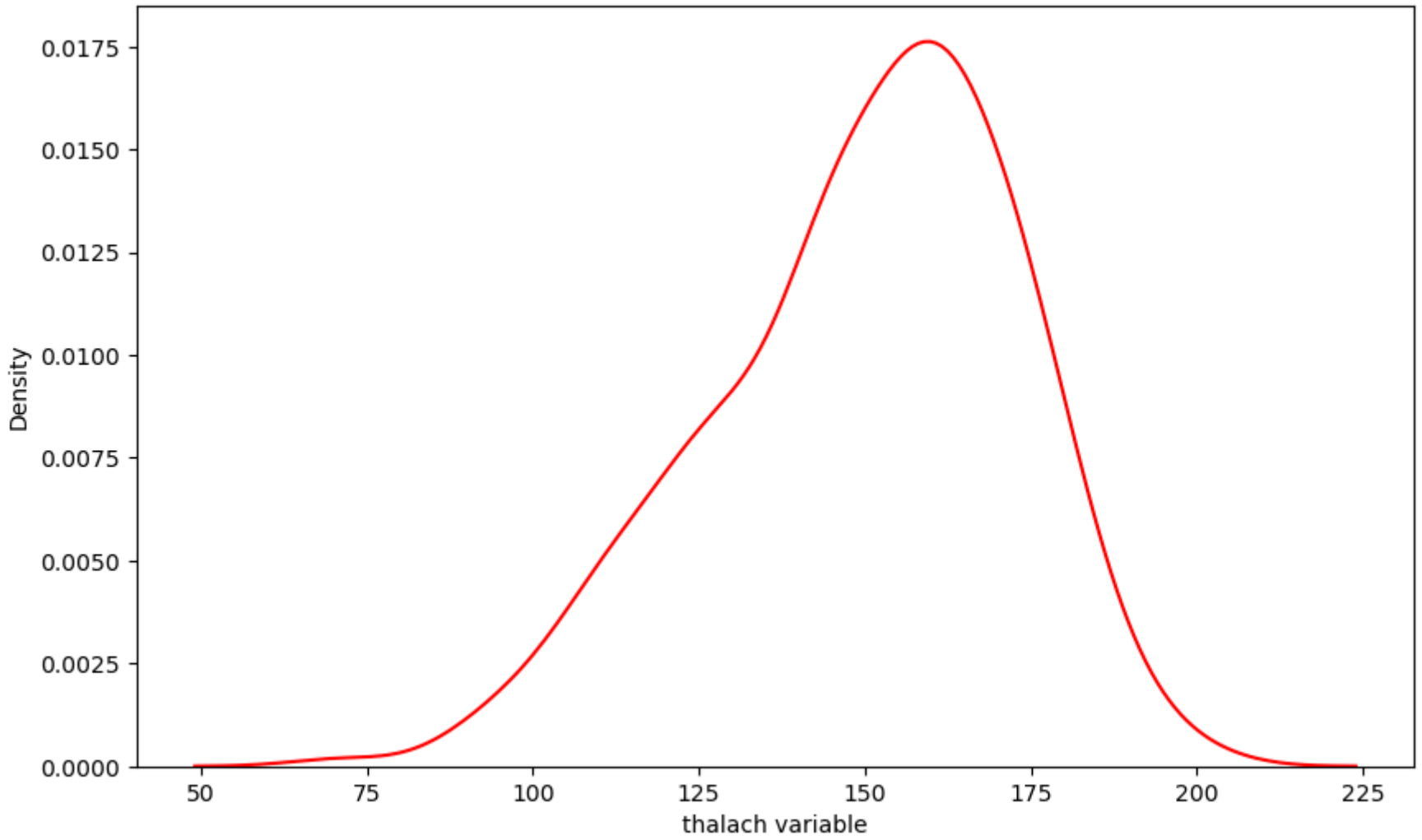
In [37]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x,name='thalach variable')
ax = sns.kdeplot(x)
```




In [38]:
```
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') # make shade = True you get  shade inside the graph
```




In [39]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x,name='thalach variable')
ax = sns.kdeplot(x, color='r')  # By default shade is False
```

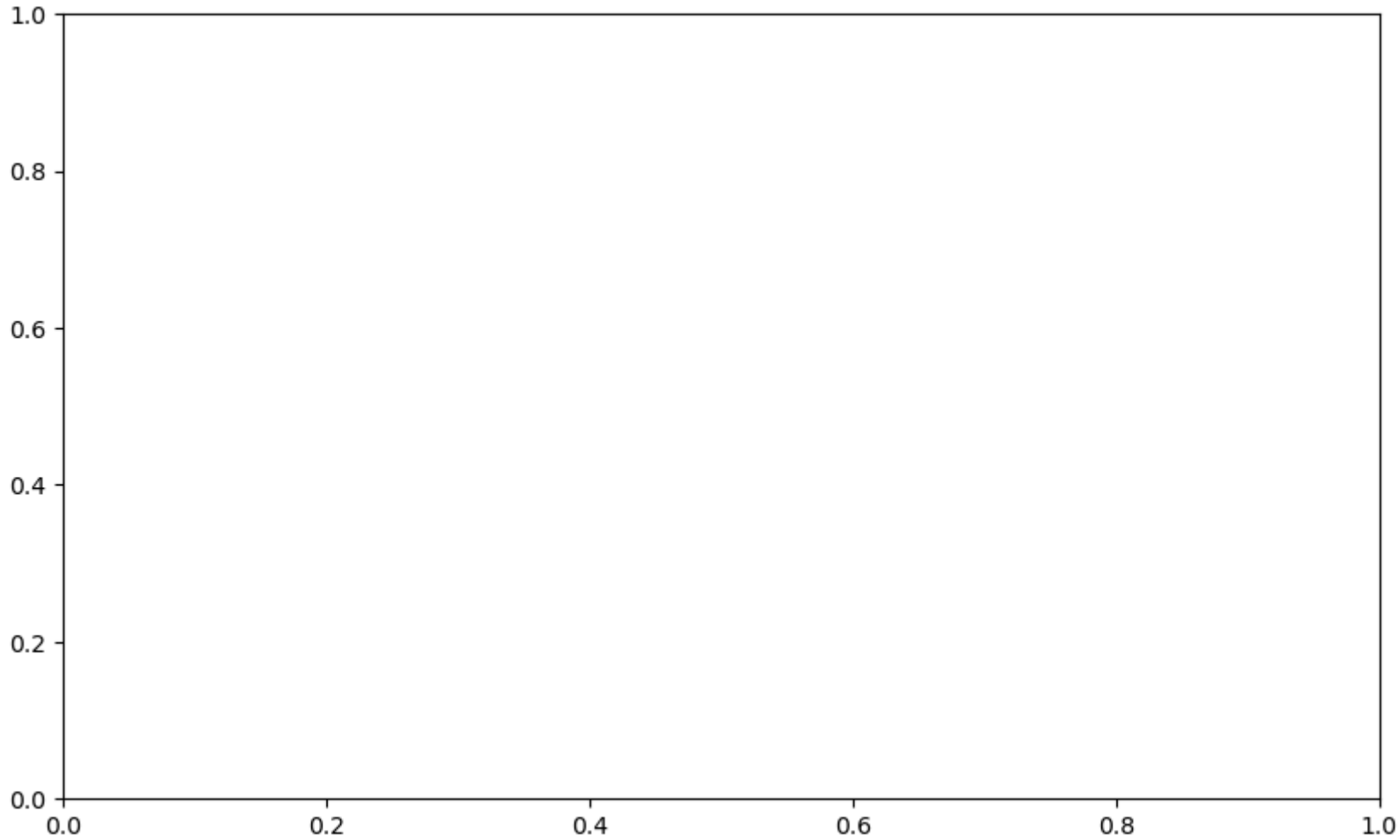


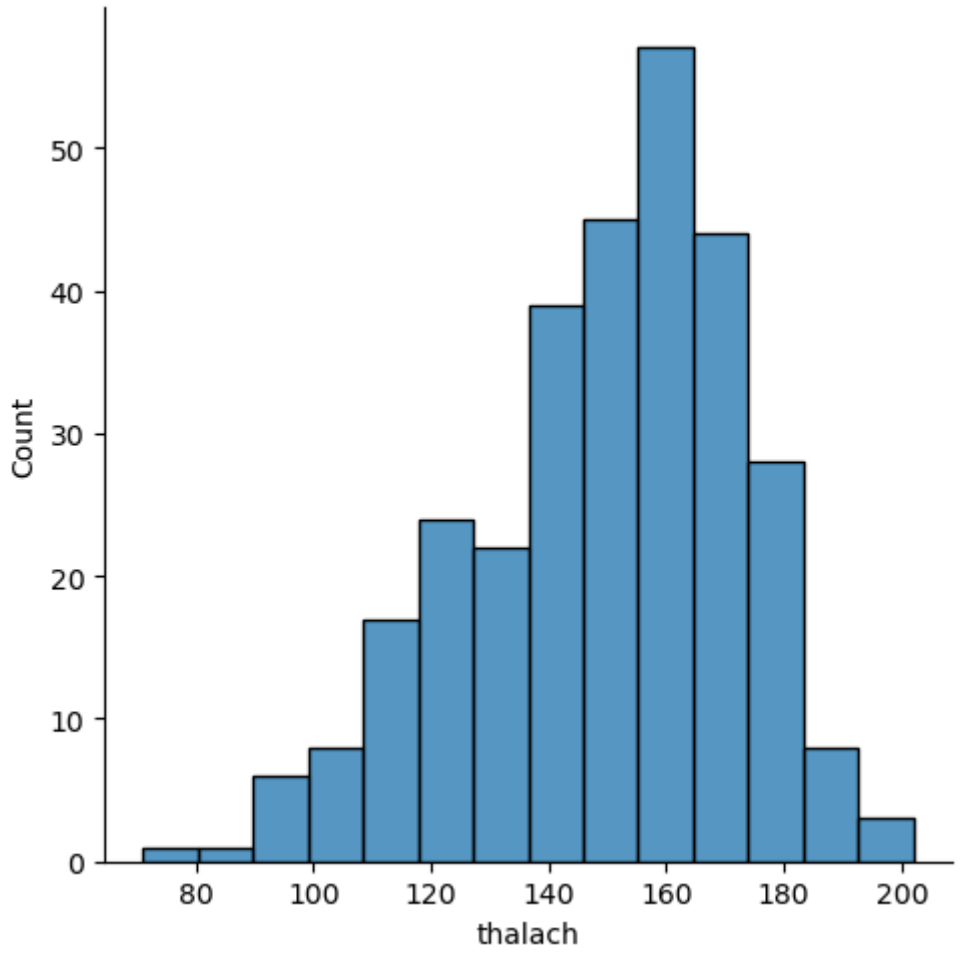

# Histogram

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

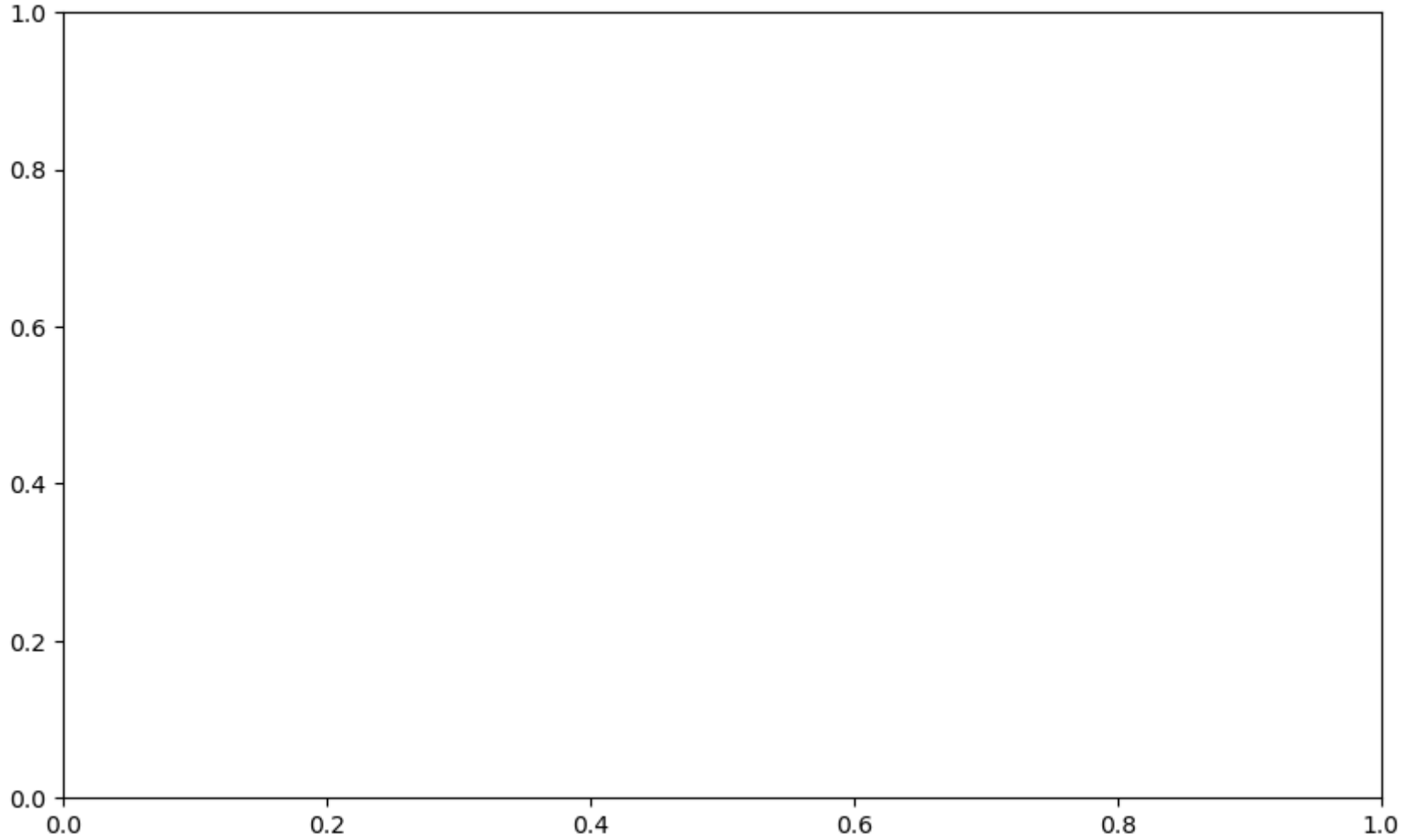
In [40]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['thalach']
```

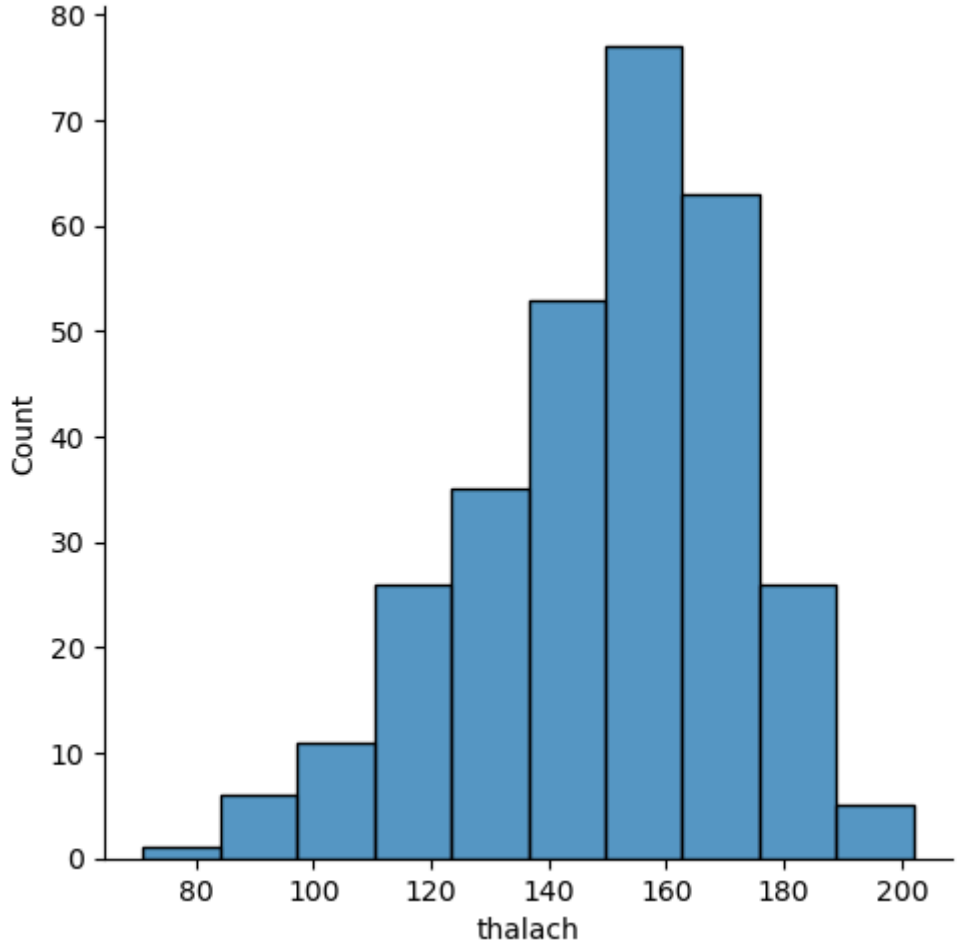
```
ax =  sns.displot(x)
plt.show()
```



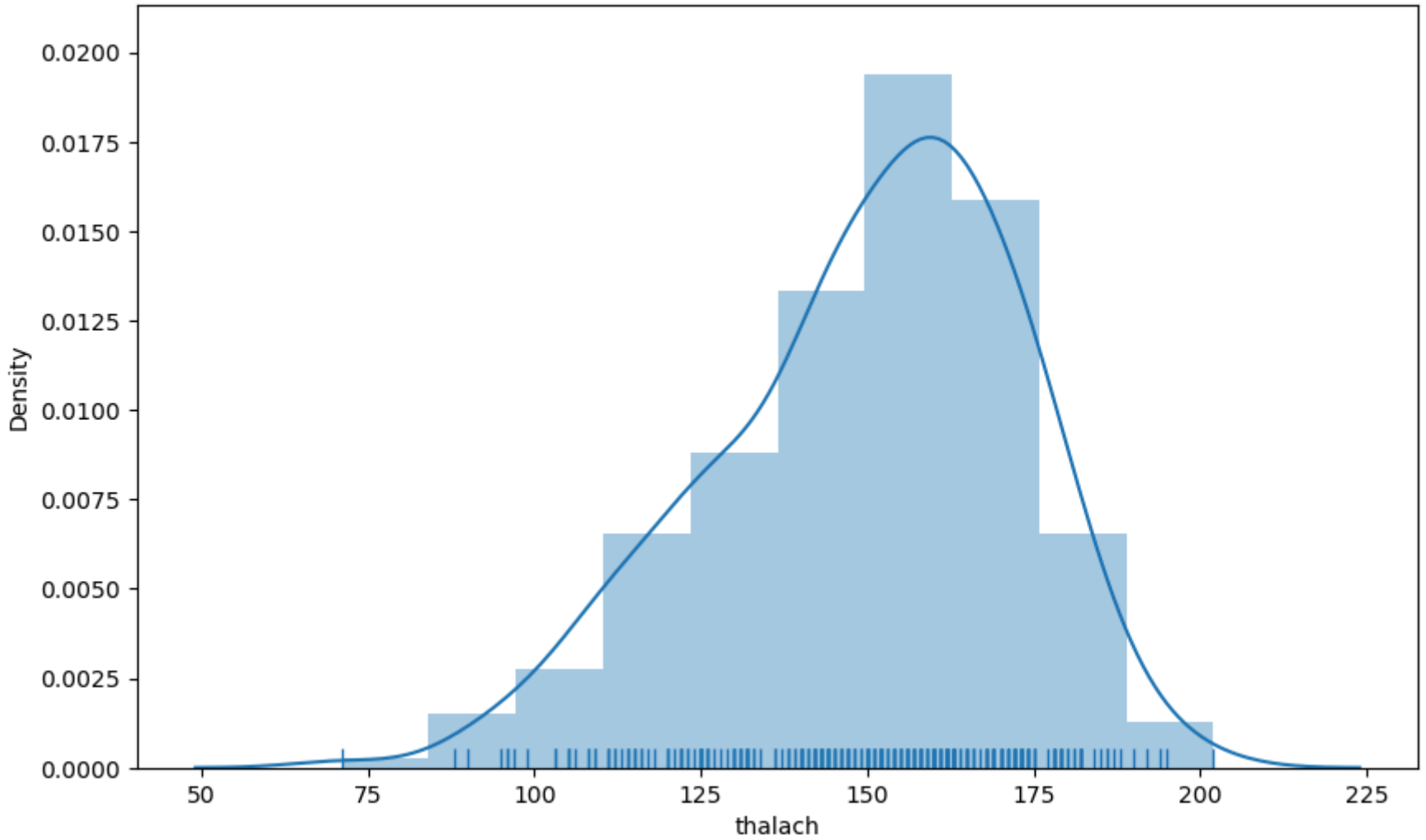



In [41]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['thalach']
ax =  sns.displot(x,bins=10)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



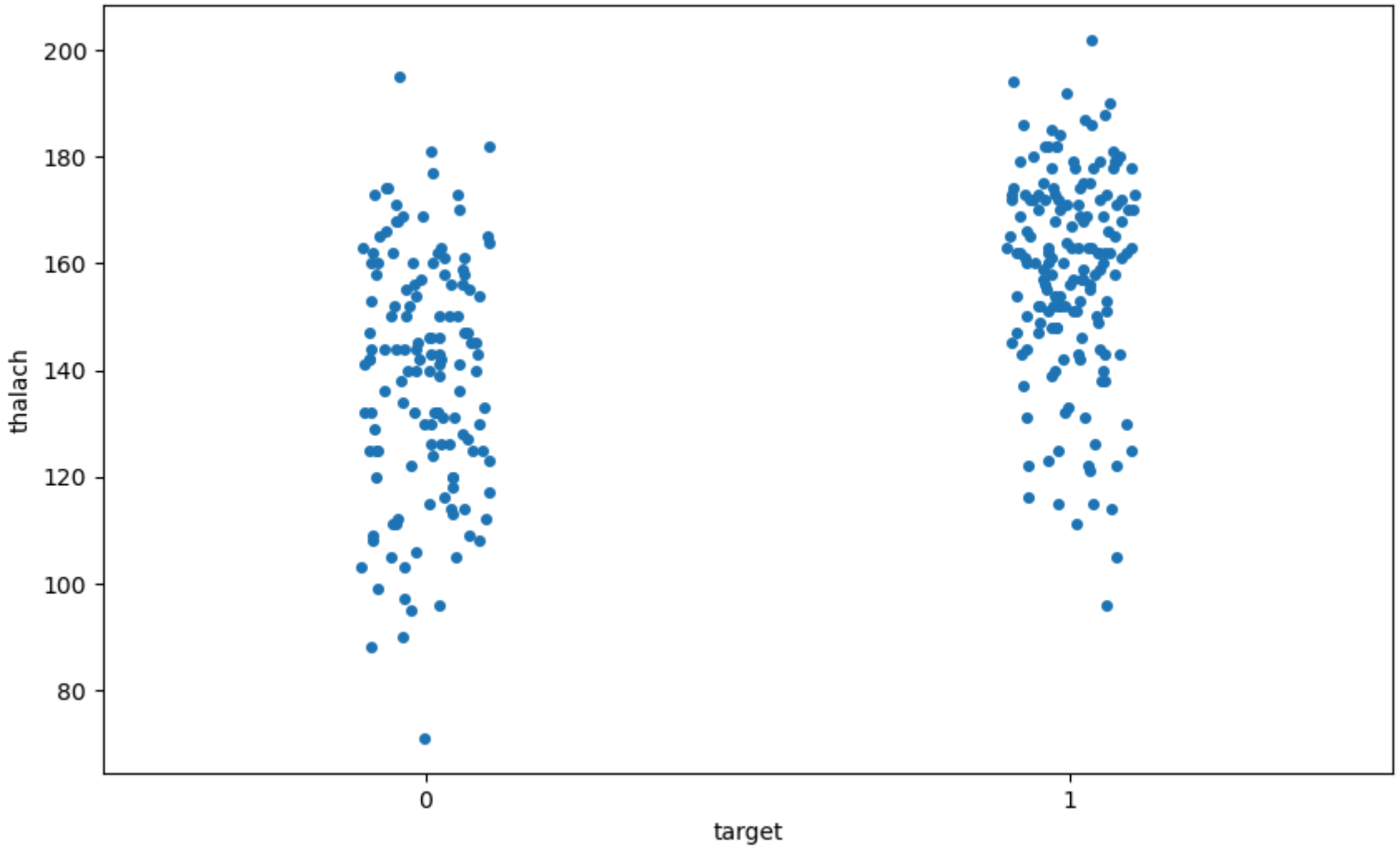

In [42]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['thalach']
ax =  sns.distplot(x,kde=True,rug=True,bins=10)
```




## Visuvalize Frequency distribution of thalach variable wrt target

In [43]:
```
f,ax = plt.subplots(figsize=(10,6))
sns.stripplot(x='target',y='thalach',data=df)
```

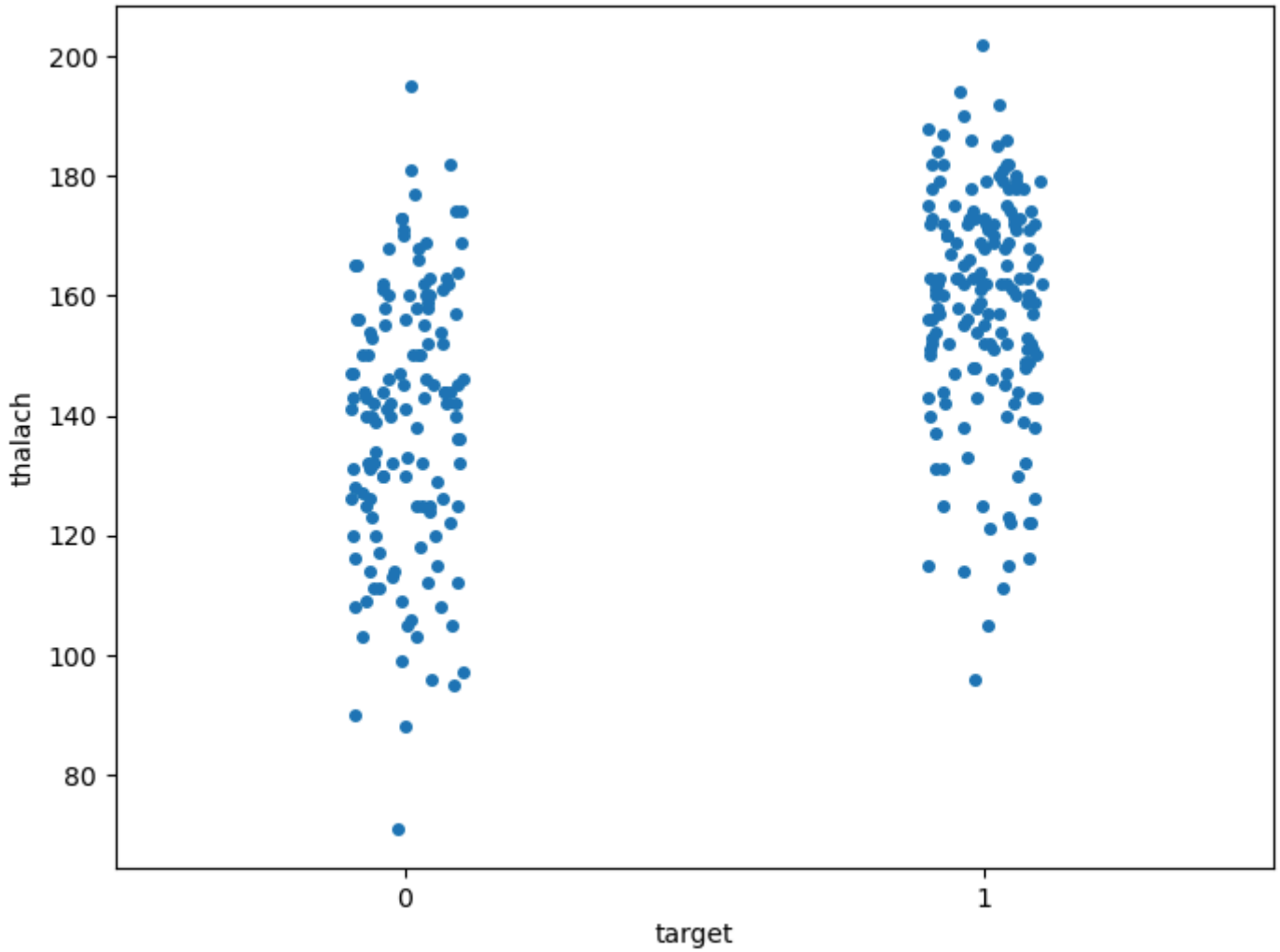
Out[43]:
```
<Axes: xlabel='target', ylabel='thalach'>
```




We can add jitter to bring out the distribution of values

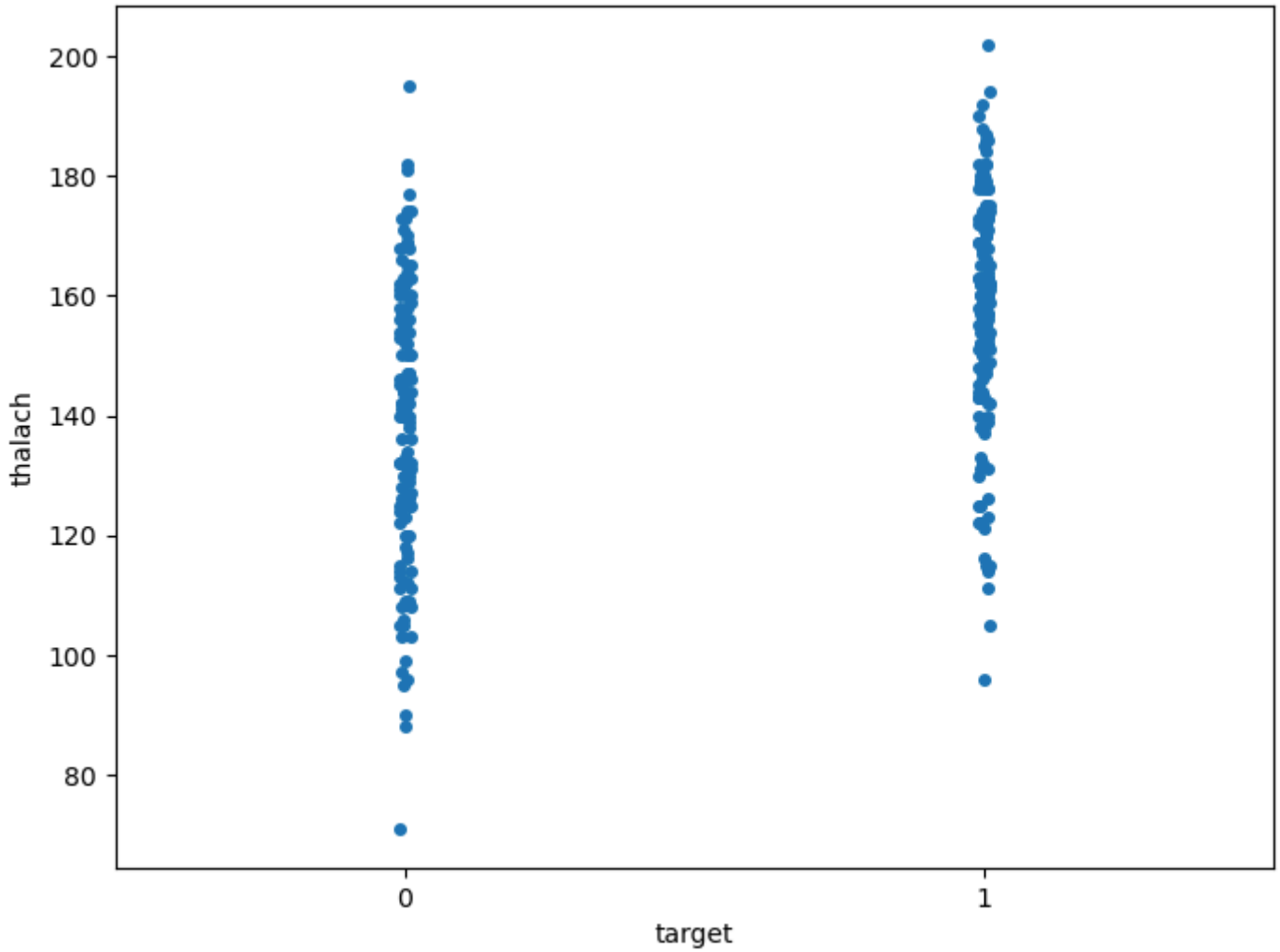
In [45]:
```
f, ax = plt.subplots(figsize=(8,6))
sns.stripplot(x='target',y='thalach',data=df)#with out jitter parameter
```

Out[45]:
```
<Axes: xlabel='target', ylabel='thalach'>
```




In [46]:
```
f, ax = plt.subplots(figsize=(8,6))
sns.stripplot(x='target',y='thalach',data=df,jitter=0.01)#with out jitter parameter
```

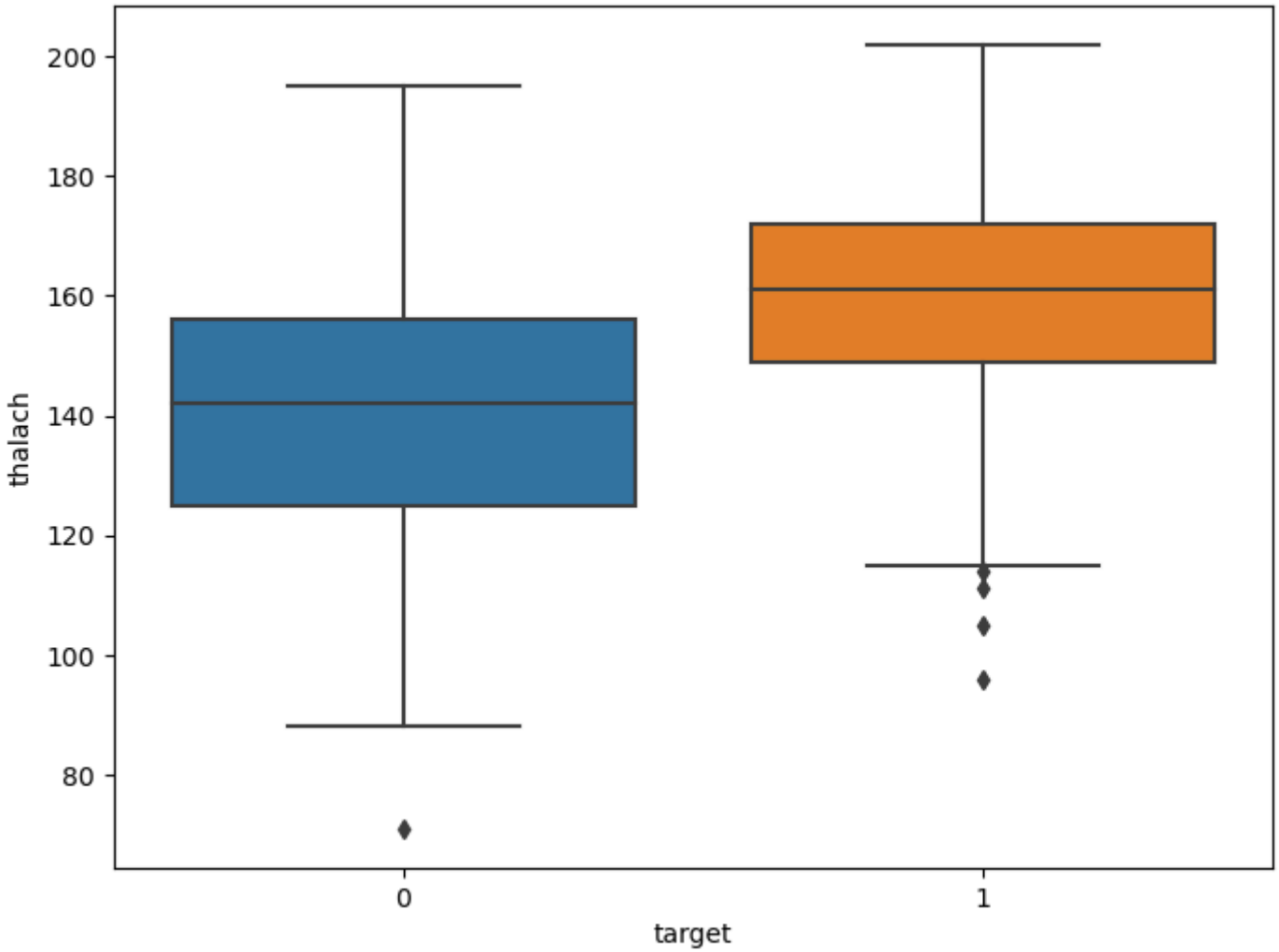
Out[46]:
```
<Axes: xlabel='target', ylabel='thalach'>
```




# BoxPlot

In [47]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x='target',y='thalach',data=df)
```

Out[47]:
```
<Axes: xlabel='target', ylabel='thalach'>
```




# Findings of Bivariate Analysis

- 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).

# Multivariate Analysis

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

# Discover Patterns and relationships

- An important step is EDA is to discover patterns and relationships between variable in the dataset
- Use heat map and pair plot to discover the patterns and relationships in the dataset

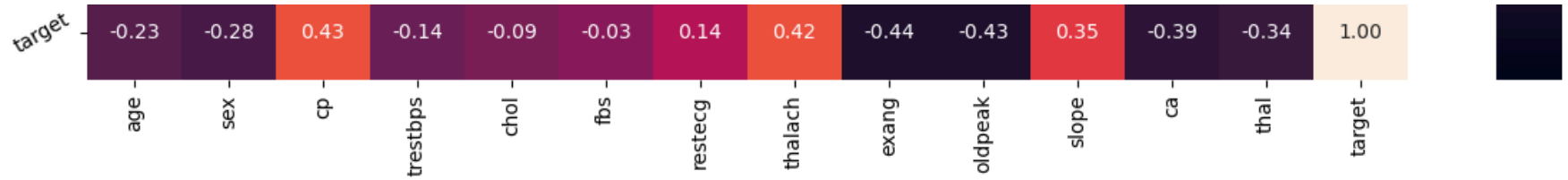
# Heat Map

- A heatmap is a graphical representation of data where values in a matrix are represented as colors.
- It's a way to visualize complex data sets, especially when dealing with large amounts of data or data with multiple variables.

In [48]:
```
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)
```

Out[48]:
```
[Text(0, 0.5, 'age'),
 Text(0, 1.5, 'sex'),
 Text(0, 2.5, 'cp'),
 Text(0, 3.5, 'trestbps'),
 Text(0, 4.5, 'chol'),
 Text(0, 5.5, 'fbs'),
 Text(0, 6.5, 'restecg'),
 Text(0, 7.5, 'thalach'),
 Text(0, 8.5, 'exang'),
 Text(0, 9.5, 'oldpeak'),
 Text(0, 10.5, 'slope'),
 Text(0, 11.5, 'ca'),
 Text(0, 12.5, 'thal'),
 Text(0, 13.5, 'target')]
```


correlation Heatmap of Heart Disease Dataset
age 1.00 -0.10 -0.07 0.28 0.21 0.12 -0.12 -0.40 0.10 0.21 -0.17 0.28 0.07 -0.23
sex -0.10 1.00 -0.05 -0.06 -0.20 0.05 -0.06 -0.04 0.14 0.10 -0.03 0.12 0.21 -0.28
cp -0.07 -0.05 1.00 0.05 -0.08 0.09 0.04 0.30 -0.39 -0.15 0.12 -0.18 -0.16 0.43
trestbps 0.28 -0.06 0.05 1.00 0.12 0.18 -0.11 -0.05 0.07 0.19 -0.12 0.10 0.06 -0.14
chol 0.21 -0.20 -0.08 0.12 1.00 0.01 -0.15 -0.01 0.07 0.05 -0.00 0.07 0.10 -0.09
fbs 0.12 0.05 0.09 0.18 0.01 1.00 -0.08 -0.01 0.03 0.01 -0.06 0.14 -0.03 -0.03
restecg -0.12 -0.06 0.04 -0.11 -0.15 -0.08 1.00 0.04 -0.07 -0.06 0.09 -0.07 -0.01 0.14
thalach -0.40 -0.04 0.30 -0.05 -0.01 -0.01 0.04 1.00 -0.38 -0.34 0.39 -0.21 -0.10 0.42
exang 0.10 0.14 -0.39 0.07 0.07 0.03 -0.07 -0.38 1.00 0.29 -0.26 0.12 0.21 -0.44
oldpeak 0.21 0.10 -0.15 0.19 0.05 0.01 -0.06 -0.34 0.29 1.00 -0.58 0.22 0.21 -0.43
slope -0.17 -0.03 0.12 -0.12 -0.00 -0.06 0.09 0.39 -0.26 -0.58 1.00 -0.08 -0.10 0.35
ca 0.28 0.12 -0.18 0.10 0.07 0.14 -0.07 -0.21 0.12 0.22 -0.08 1.00 0.15 -0.39
thal 0.07 0.21 -0.16 0.06 0.10 -0.03 -0.01 -0.10 0.21 0.21 -0.10 0.15 1.00 -0.34
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4




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

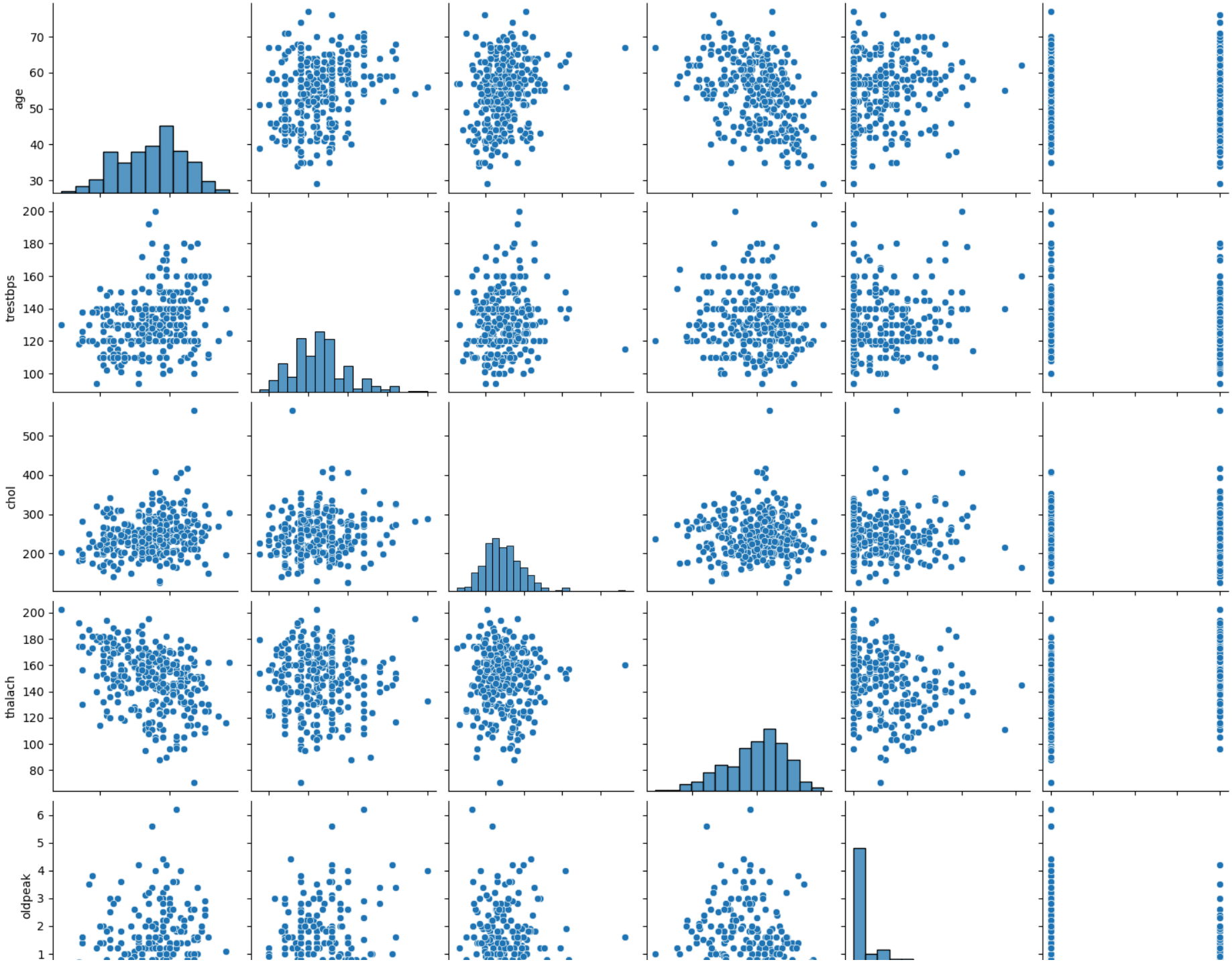
- target and cp variable are mildly positively correlated (correlation coefficient = 0.43).
- target and thalach variable are also mildly positively correlated (correlation coefficient = 0.42).
- target and slope variable are weakly positively correlated (correlation coefficient = 0.35).
- target and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

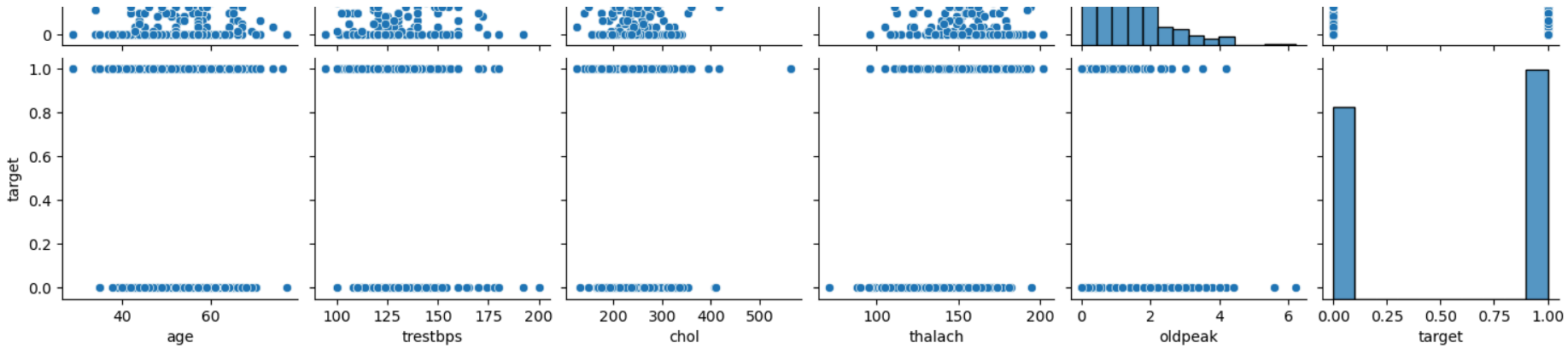
# PairPlot

- A PairPlot, short for Pairwise Plot, is a type of visualization commonly used in exploratory data analysis (EDA) to examine the relationships between multiple variables in a dataset.
- It's particularly useful when dealing with datasets containing numerous variables, as it allows for quick visual inspection of potential correlations and patterns.

In [49]:
```
num_var = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target' ]
sns.pairplot(df[num_var],kind='scatter',diag_kind='hist')
# kind : Takes values from{‘scatter’, ‘kde’, ‘hist’, ‘reg’}
#diag_kind : Takes values from{‘auto’, ‘hist’, ‘kde’, None}
```

Out[49]:
```
<seaborn.axisgrid.PairGrid at 0x1f5341cfd50>
```


age
trestbps
chol
thalach
oldpeak




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

# Analysis of age and other variables

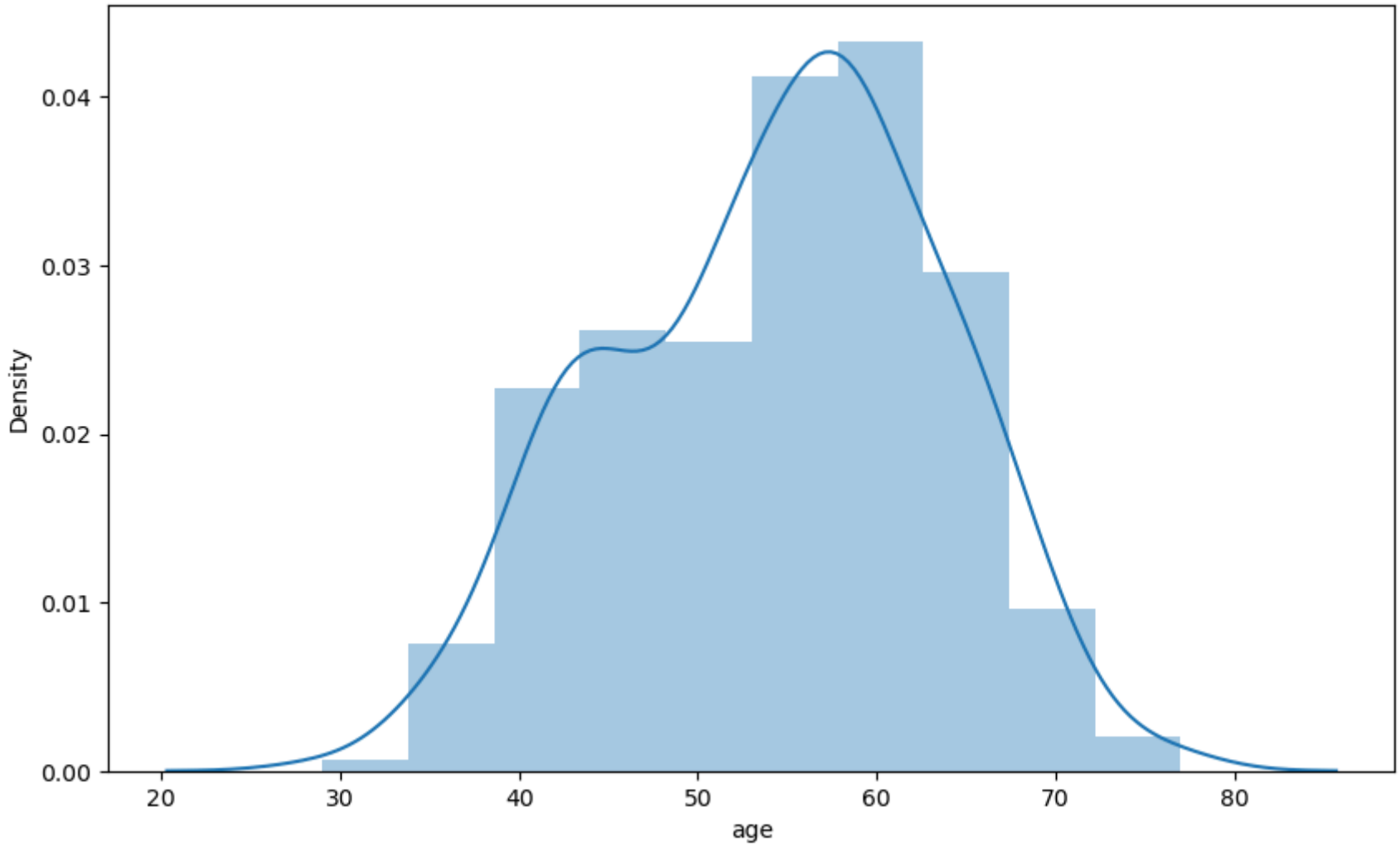
In [50]:
```
df['age'].nunique()
```

Out[50]:
```
41
```

In [51]:
```
df['age'].describe()
```

Out[51]:
```
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
```

In [52]:
```
f,ax = plt.subplots(figsize=(10,6))
x = df['age']
ax = sns.distplot(x,bins=10)
```



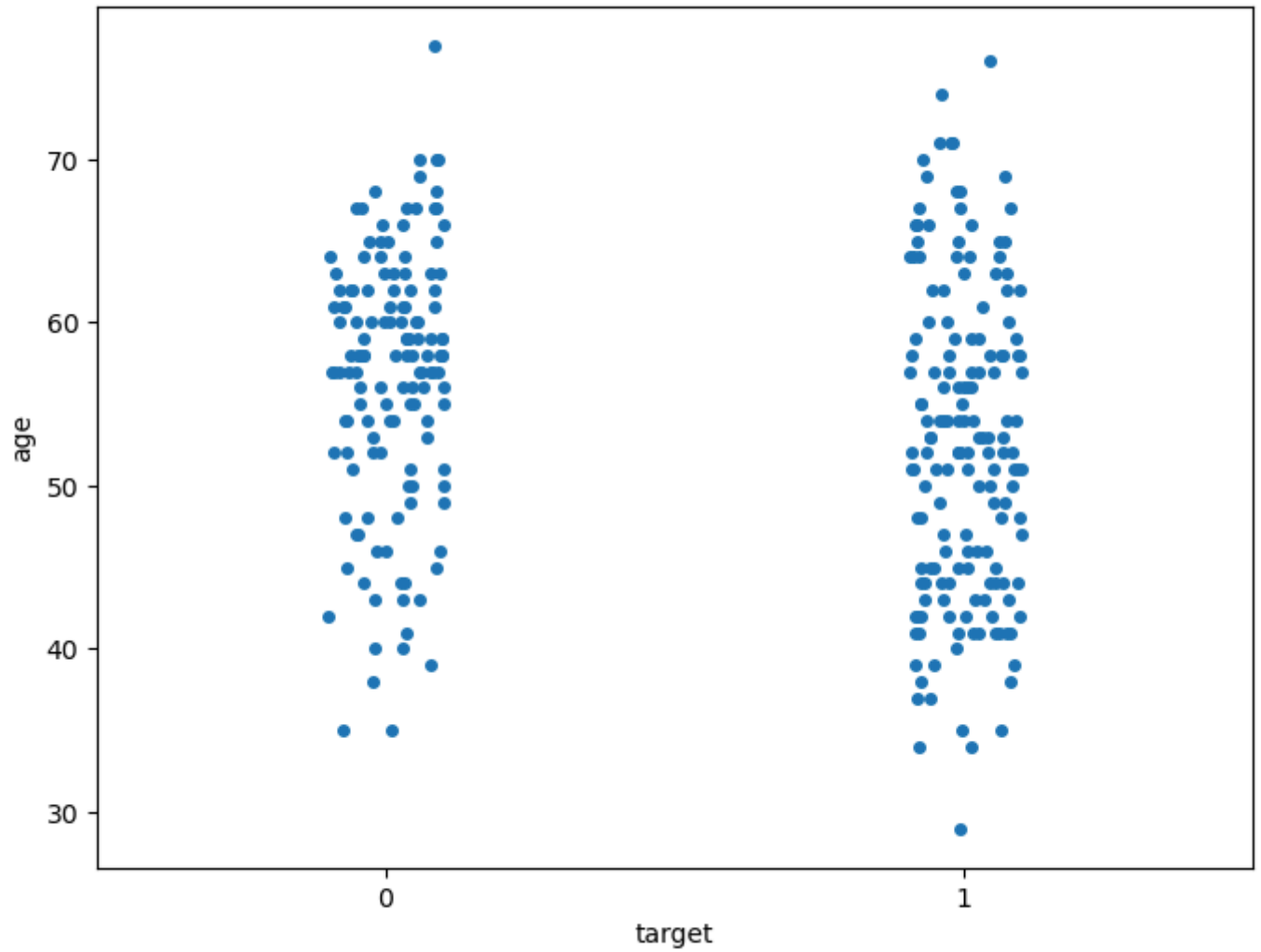

In [53]:
```
The age variable distribution is approximately normal
```

```
  Cell In[53], line 1
    The age variable distribution is approximately normal
        ^
SyntaxError: invalid syntax
```

# Analyze age and target variable

In [54]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.stripplot(x='target',y = 'age',data=df)
```

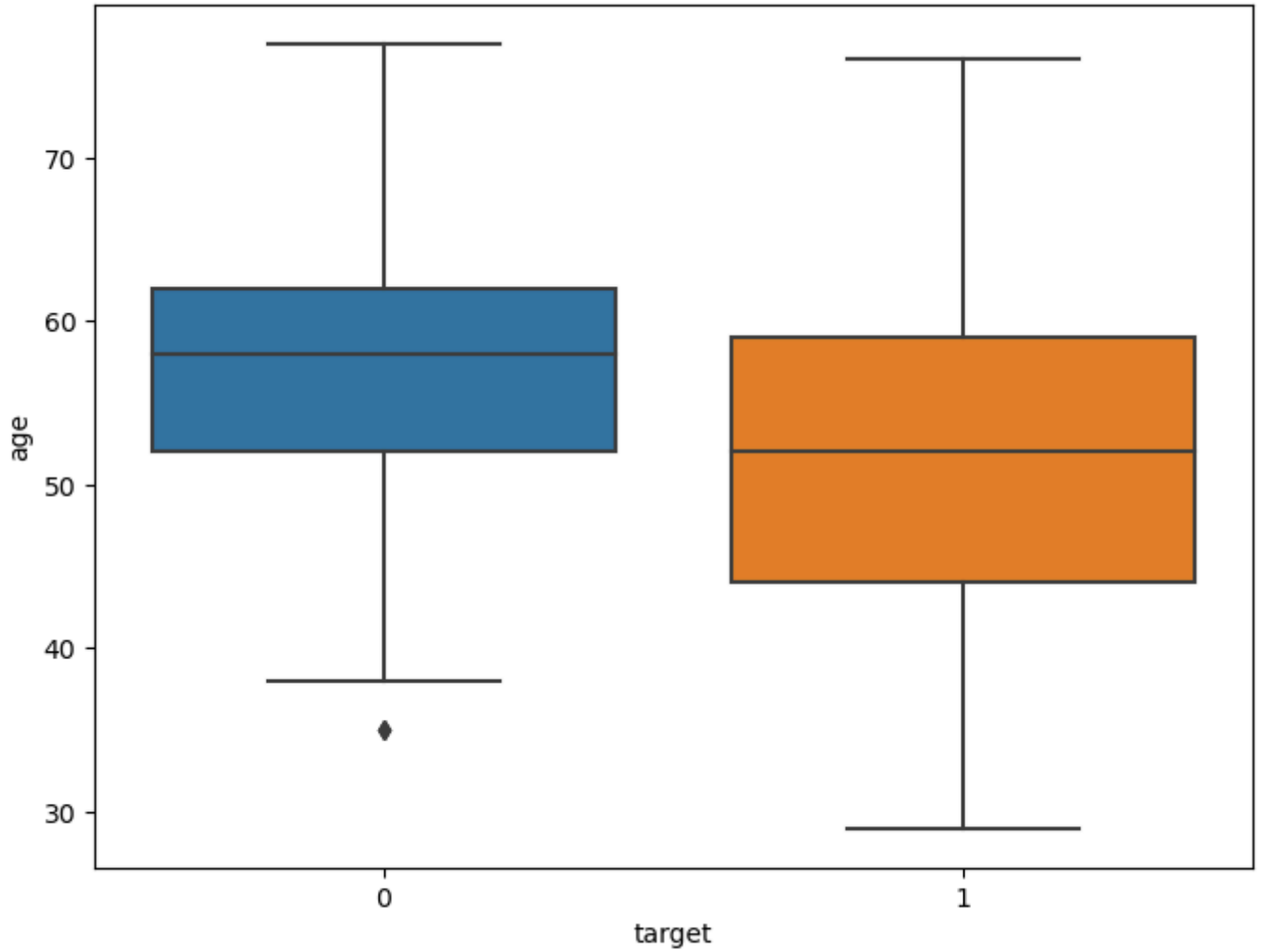
Out[54]:
```
<Axes: xlabel='target', ylabel='age'>
```




## Visualize distribution of age variable wrt target with boxplot

In [55]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x='target',y='age',data=df)
```

Out[55]:
```
<Axes: xlabel='target', ylabel='age'>
```



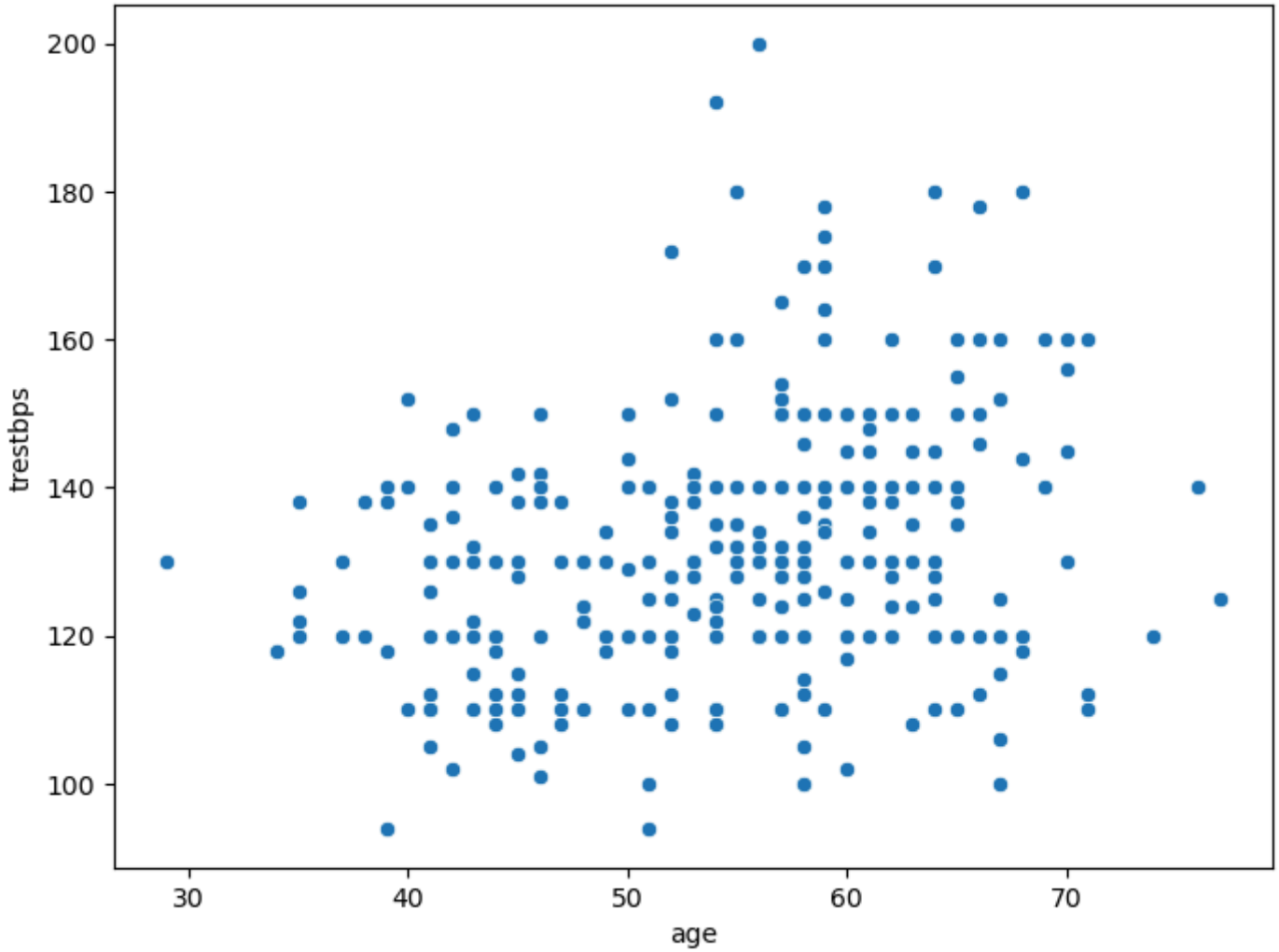

# The above boxplot tells two different things

- 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

In [56]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.scatterplot(x='age',y='trestbps',data=df)
```

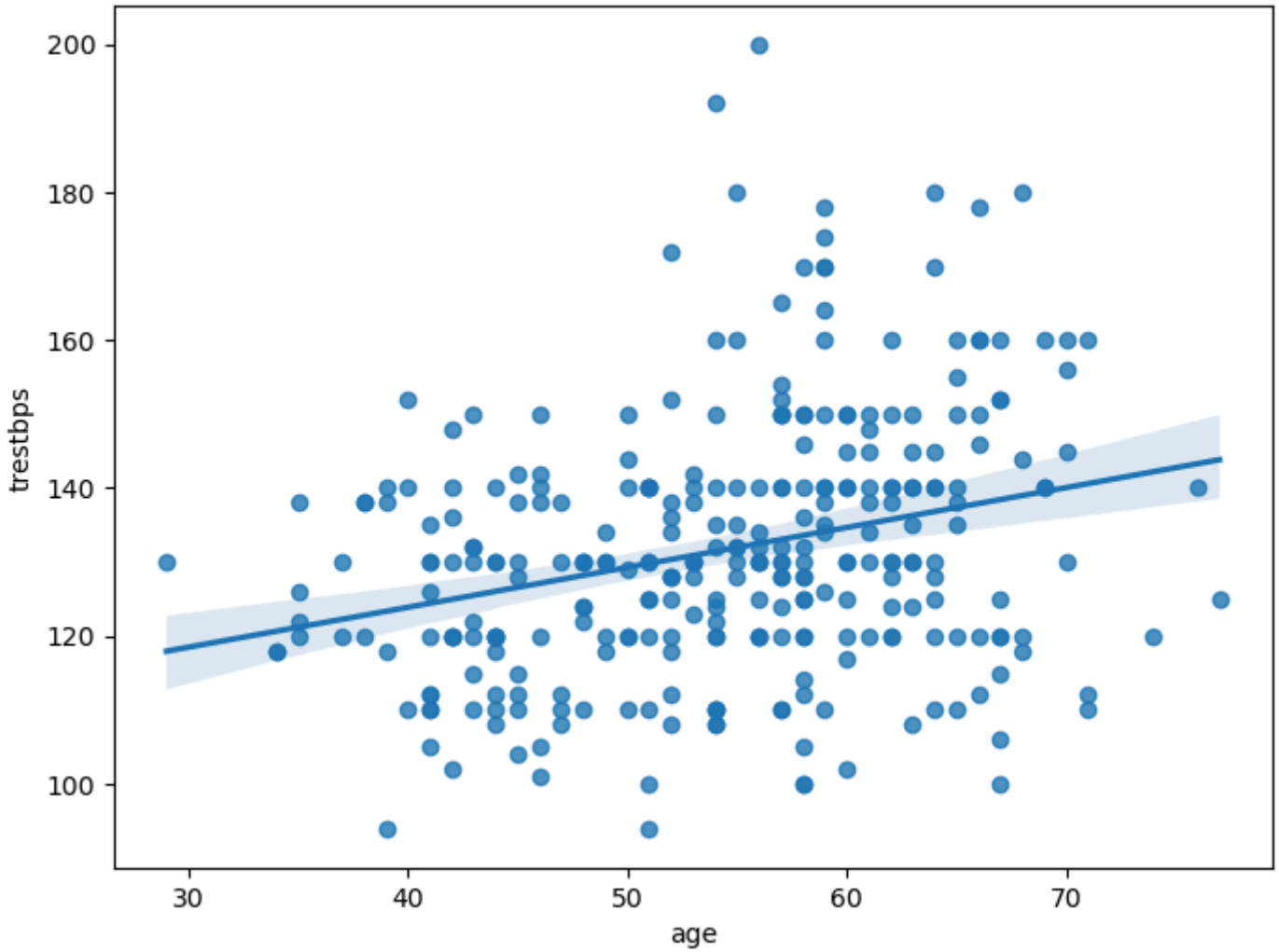


The above scatter plot shows that there is no correlation between age and trestbps variable

# seaborn.regplot()

This method is used to plot data and a linear regression model fit.

In [57]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.regplot(x='age',y='trestbps',data=df)
```




The above line shows that linear regression model is not good fit to the data.

# Analyze age and chol variable

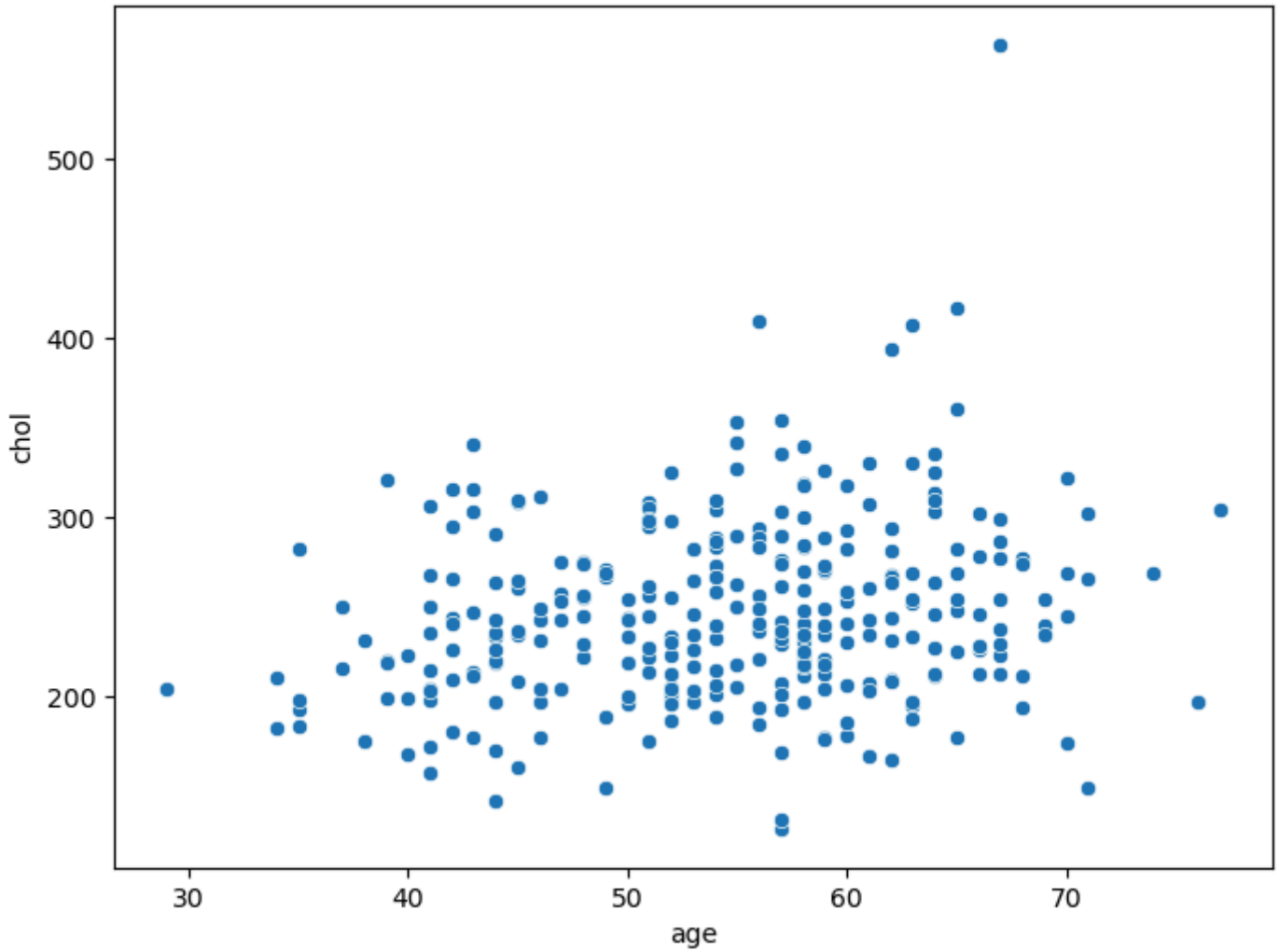
In [58]:
```
df.head()
```

Out[58]:

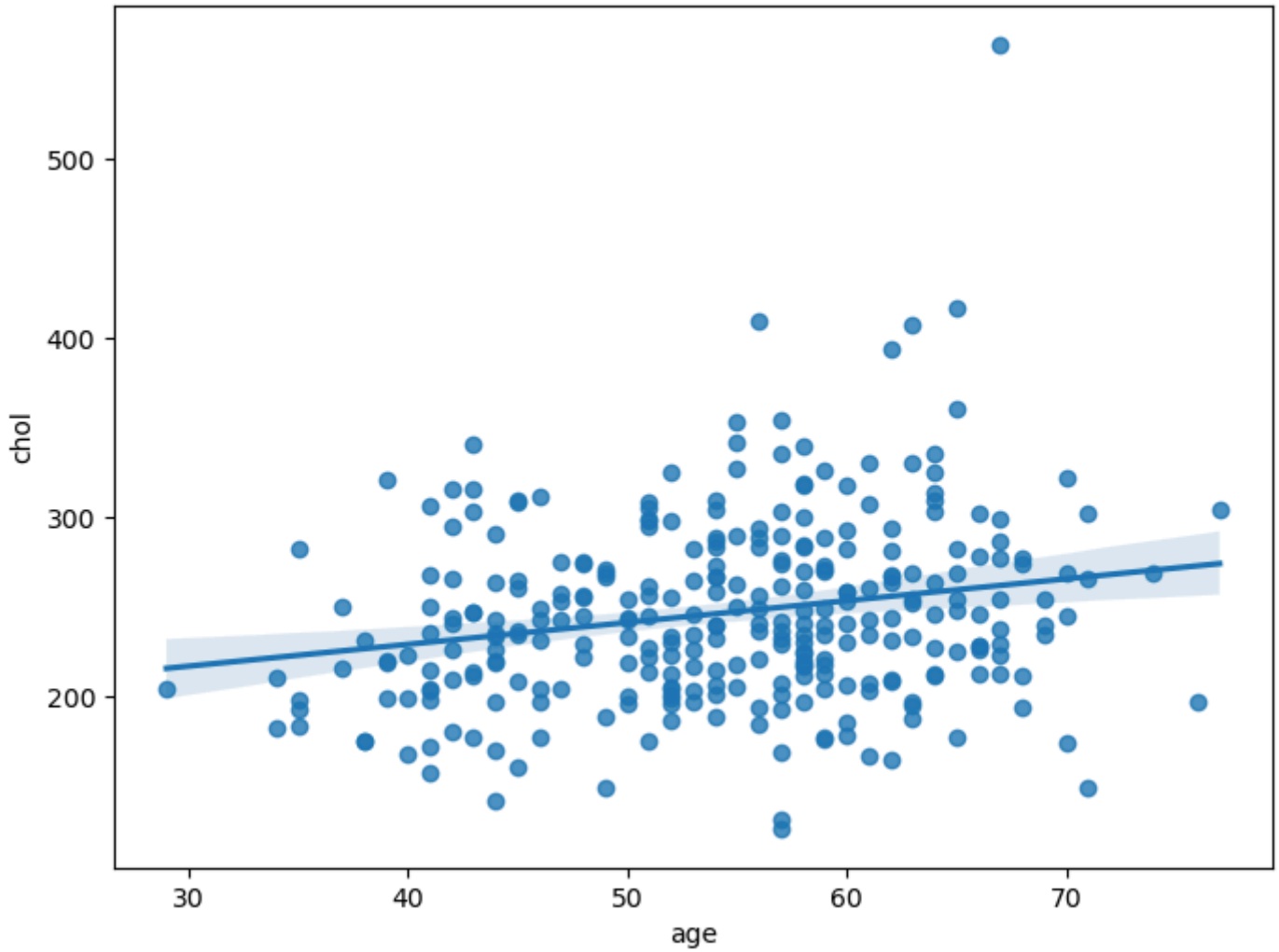
| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 |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

In [59]:

```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.scatterplot(x='age',y='chol',data=df)
```



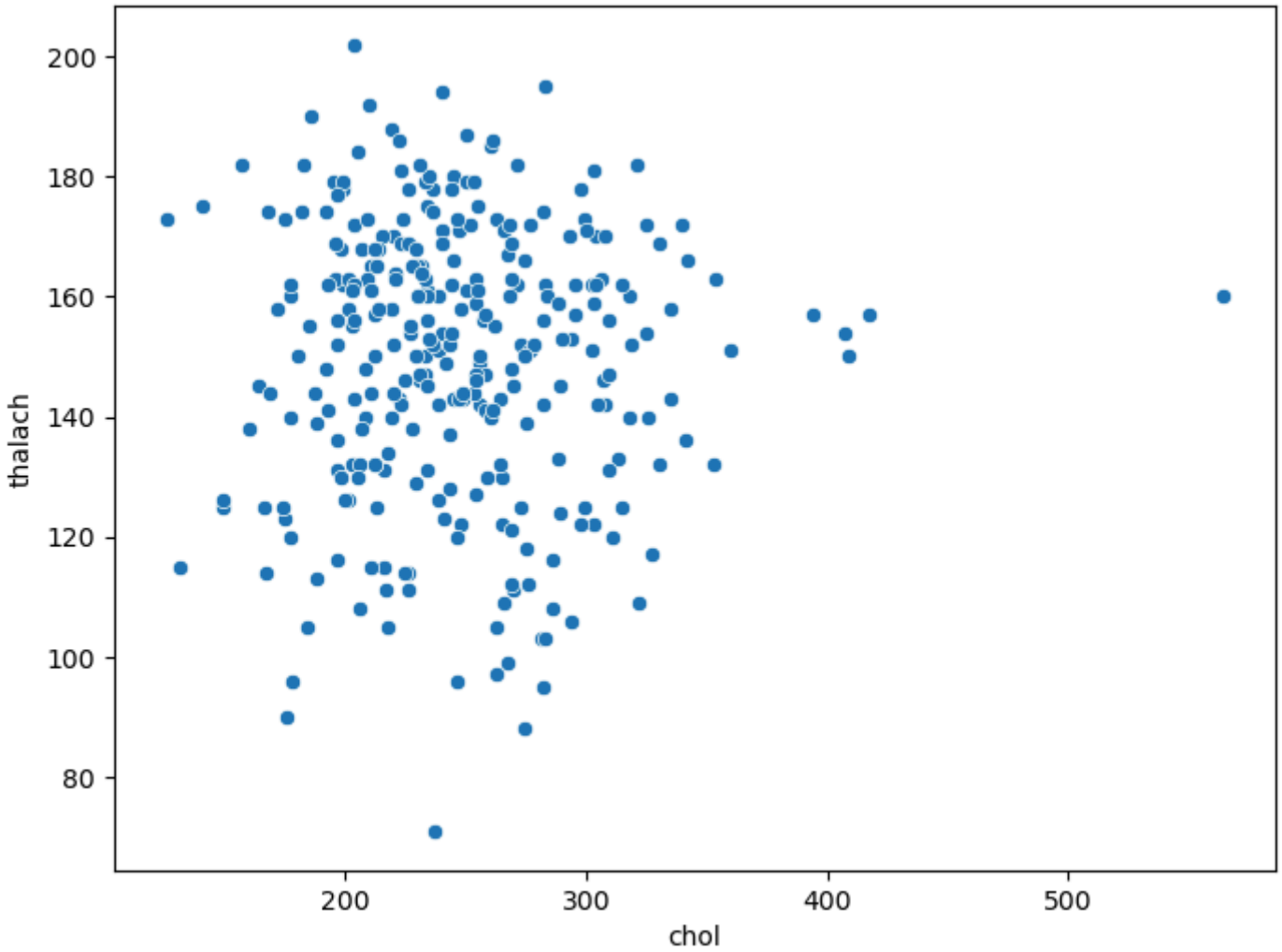

In [60]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.regplot(x='age',y='chol',data=df)
```



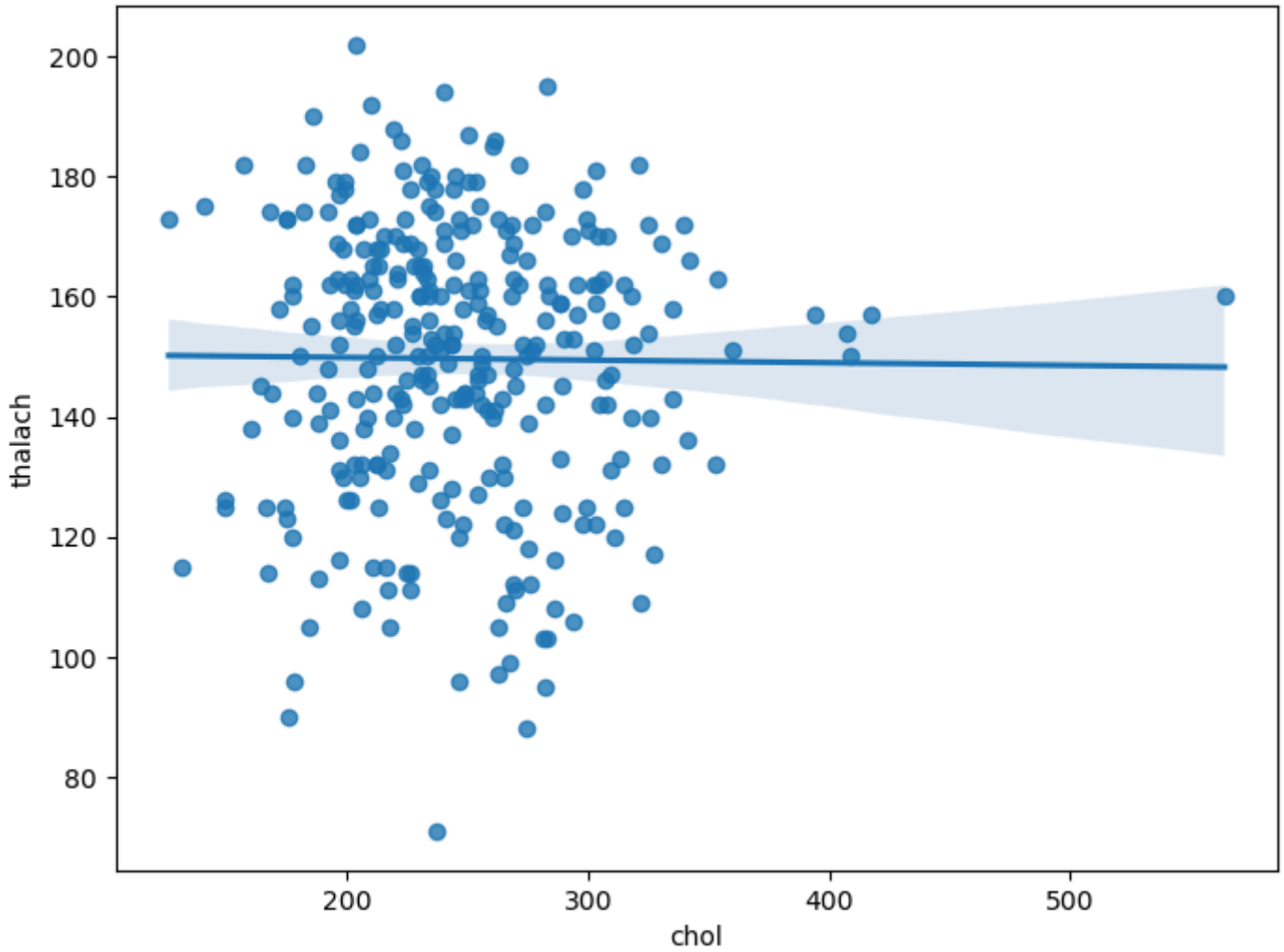

The above plot confirms that there is a slighly positive correlation between age and chol variables.

# Analyze `chol` and `thalach` variable

In [61]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.scatterplot(x='chol',y='thalach',data=df)
```




In [62]:
```
f,ax = plt.subplots(figsize=(8,6))
ax = sns.regplot(x='chol',y='thalach',data=df)
```




The above plot shows that there is no correlation between chol and thalach variable.

# Dealing with missing Values

- 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 acronmy for not anumber), is aspecial floating-point value recognized by all systems that use the standard IEEE floating-point representation

# Methods to detecting missing values

Pandas isnull() and notnull() functions

- Pandas offeres 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.

# Useful commands to detect missing values

**df.isnull()** 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()** 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.

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

# Check with ASSERT statement

- 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 [66]:
```
#assert that there are no missing values in the dataframe
assert pd.notnull(df).all().all()
```

In [69]:
```
#assert all values are greater than or equal to 0
assert (df >=0).all().all()
```

- 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.

# Oulier detection

In [70]:
```
df.head()
```

Out[70]:

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

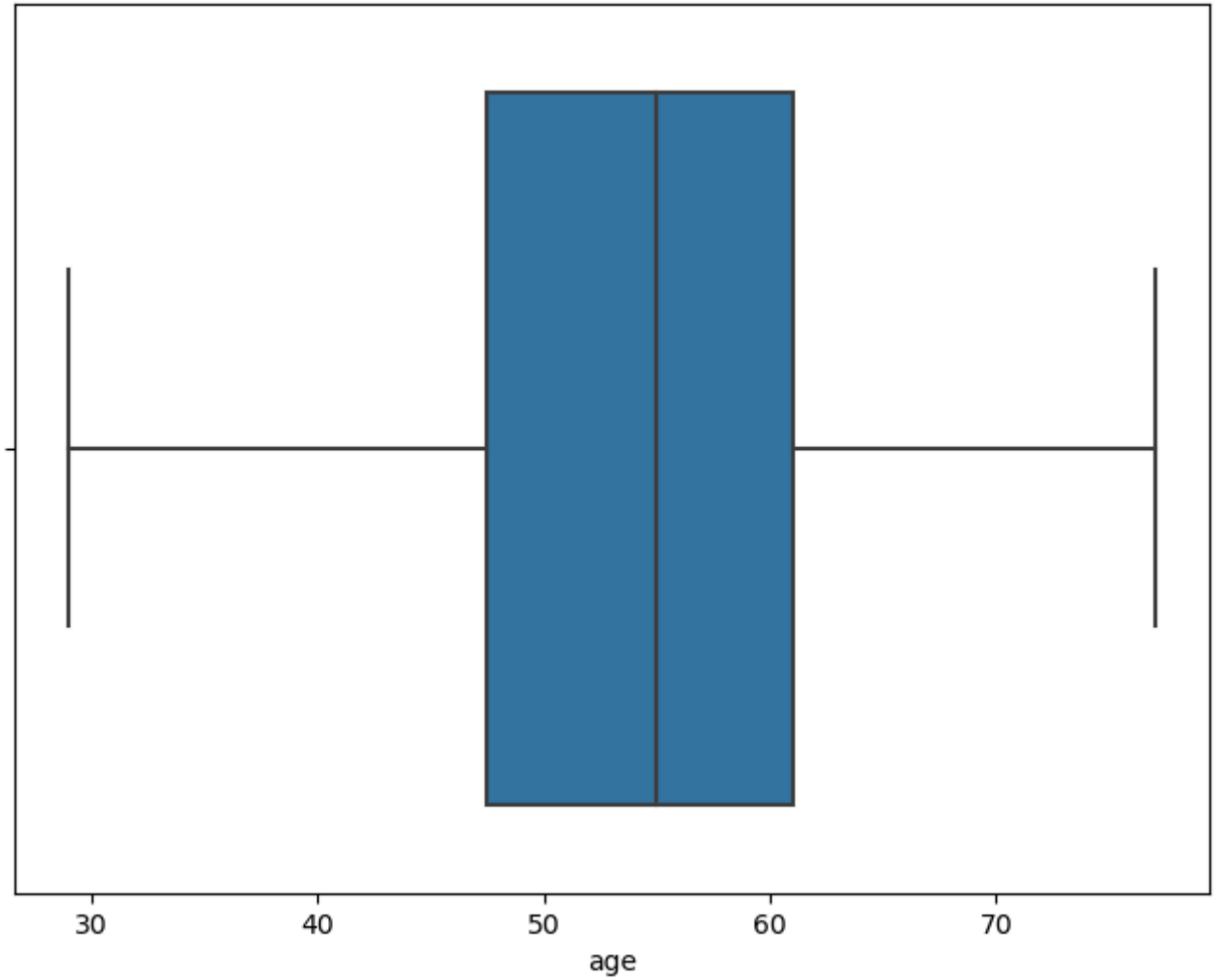
# age variable

In [71]:
```
df['age'].describe()
```

Out[71]:
```
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
```

# BoxPlot of age Variable

In [73]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=df['age'])
plt.show()
```

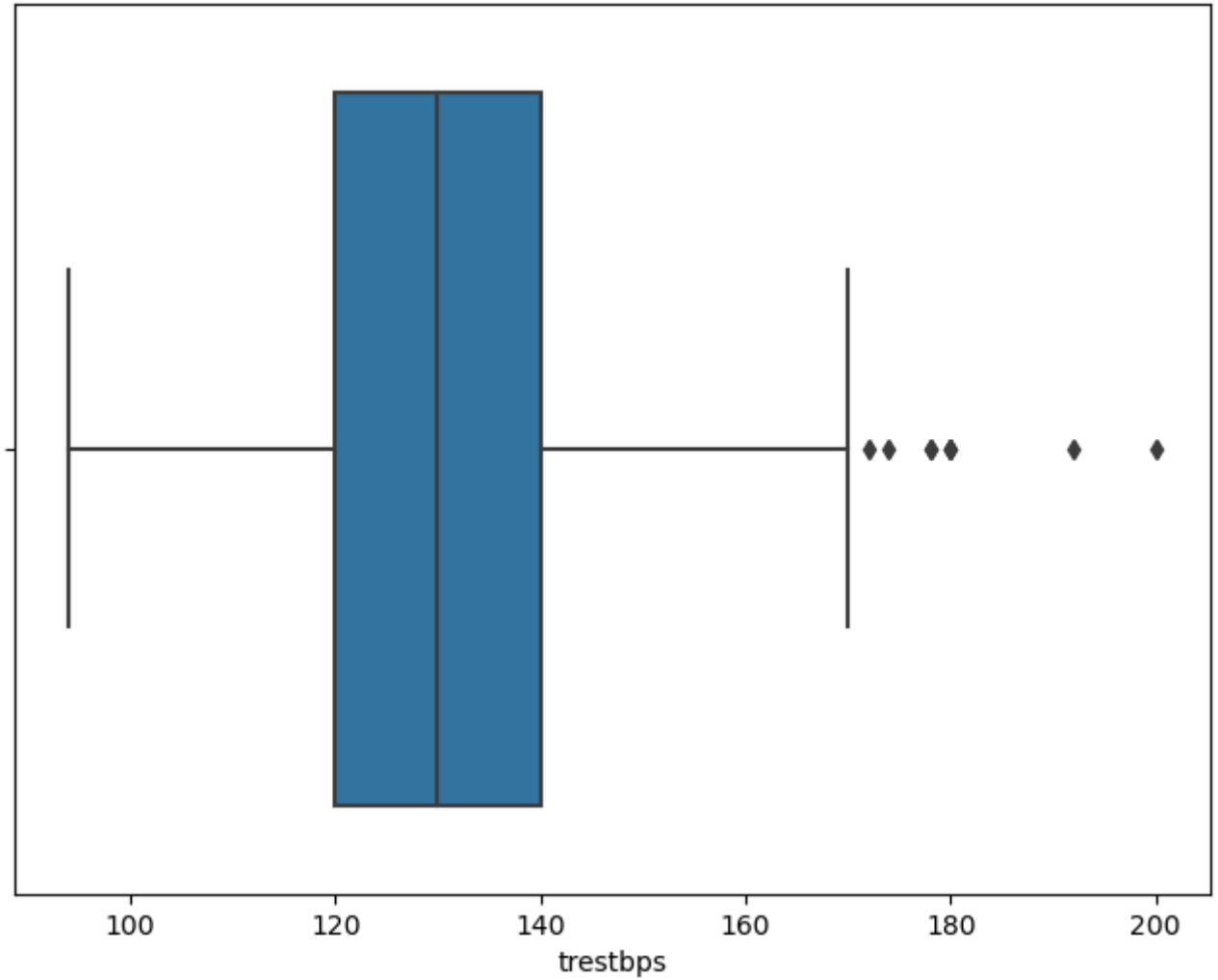



## trestbps Variable

In [74]:
```
df['trestbps'].describe()
```

Out[74]:
```
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
```

In [75]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=df['trestbps'])
plt.show()
```

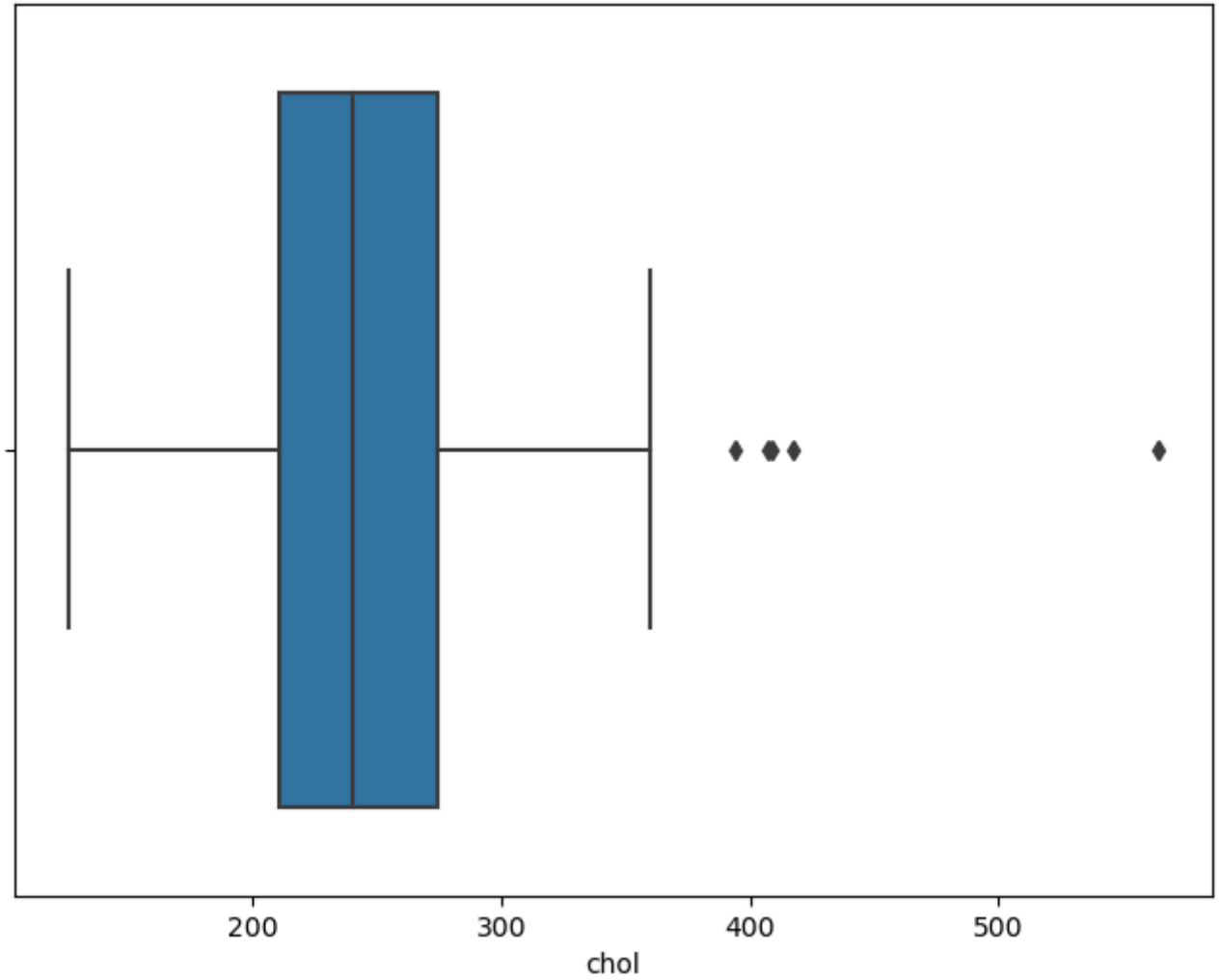



## Chol Variable

In [76]:
```
df['chol'].describe()
```

Out[76]:
```
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
```

In [77]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=df['chol'])
plt.show()
```

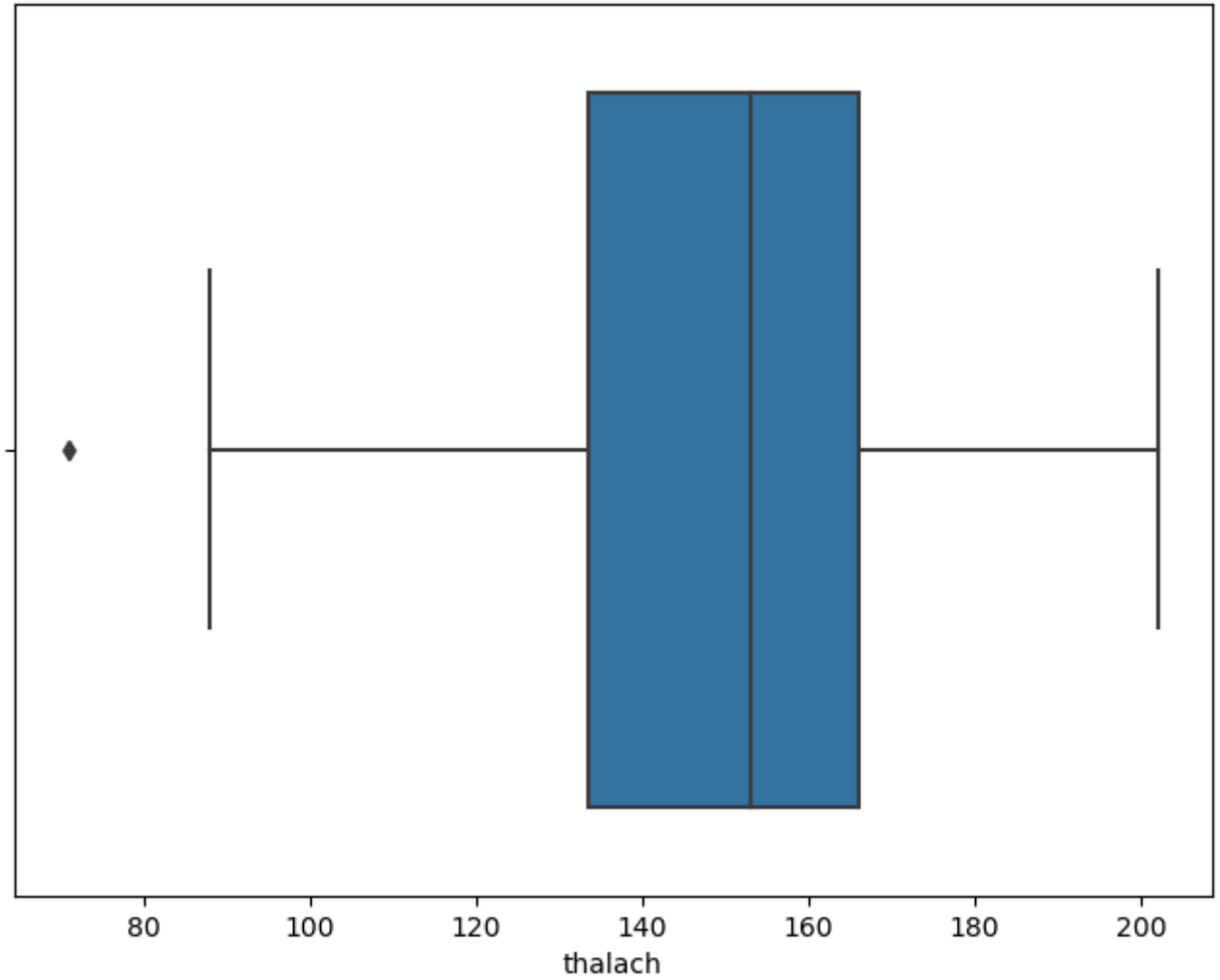



## thalach Variable

In [78]:
```
df['thalach'].describe()
```

Out[78]:
```
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
```

In [79]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=df['thalach'])
plt.show()
```

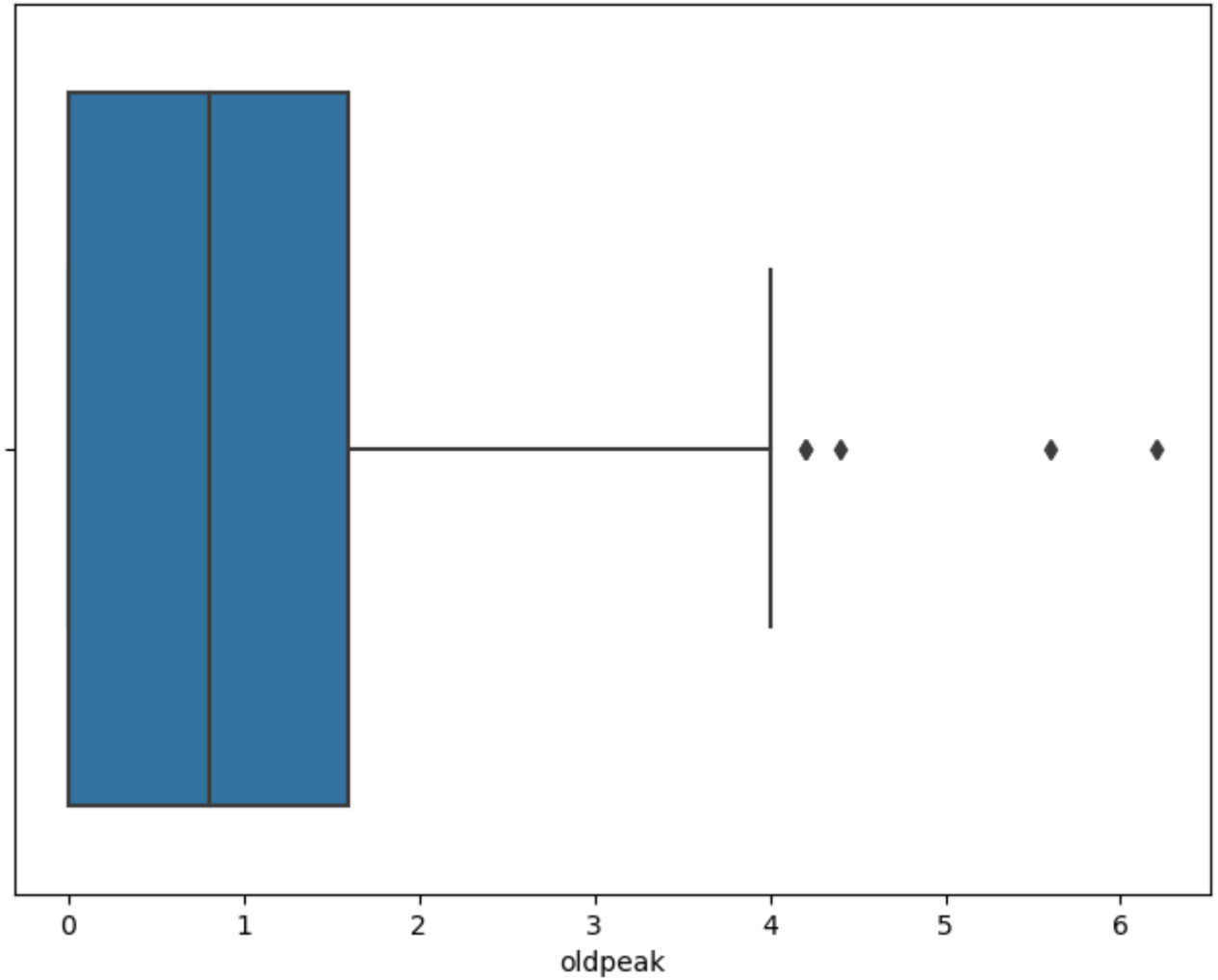



# oldpeak Variable

In [80]:
```
df['oldpeak'].describe()
```

Out[80]:
```
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
```

In [81]:
```
f,ax = plt.subplots(figsize=(8,6))
sns.boxplot(x=df['oldpeak'])
plt.show()
```




- 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.

In [ ]:

In [ ]:

In [ ]: