analysis-visualization-with-python

August 1, 2023

1 Extensive Analysis + Visualization with Python

Hello friends,

Heart disease or **Cardiovascular disease (CVD)** is a class of diseases that involve the heart or blood vessels. Cardiovascular diseases are the leading cause of death globally. This is true in all areas of the world except Africa. Together CVD resulted in 17.9 million deaths (32.1%) in 2015. Deaths, at a given age, from CVD are more common and have been increasing in much of the developing world, while rates have declined in most of the developed world since the 1970s.

So, in this kernel, I have conducted **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.

I hope you learn and enjoy this kernel.

So, your upvote would be highly appreciated.

1.1 Table of Contents

The table of contents for this project are as follows: -

1.2 1. Introduction to EDA

Back to Table of Contents

Several questions come to mind when we come across a new dataset. The below list shed light on some of these questions:-

- What is the distribution of the dataset?
- Are there any missing numerical values, outliers or anomalies in the dataset?
- What are the underlying assumptions in the dataset?
- Whether there exists relationships between variables in the dataset?
- How to be sure that our dataset is ready for input in a machine learning algorithm?
- How to select the most suitable algorithm for a given dataset?

So, how do we get answer to the above questions?

The answer is **Exploratory Data Analysis**. It enable us to answer all of the above questions.

1.3 2. Objectives of EDA

Back to Table of Contents

The objectives of the EDA are as follows:-

i. To get an overview of the distribution of the dataset.

ii. Check for missing numerical values, outliers or other anomalies in the dataset.

iii.Discover patterns and relationships between variables in the dataset.

iv. Check the underlying assumptions in the dataset.

1.4 3. Types of EDA

Back to Table of Contents

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:-

i. 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.

ii. 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.

iii. 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.

iv. 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

Enough of theory, now let the journey begin.

The first step in the EDA journey is to import the libraries.

1.5 4. Import libraries

Back to Table of Contents

[1]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load in

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

```python
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

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

# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/heart-disease-uci/heart.csv
```

We can see that the input folder contains one input file named `heart.csv`.

[2]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as st
%matplotlib inline

sns.set(style="whitegrid")
```

[3]:
```python
# ignore warnings

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

I have imported the libraries. The next step is to import the datasets.

1.6 5. Import dataset

Back to Table of Contents

I will import the dataset with the usual `pandas read_csv()` function which is used to import CSV (Comma Separated Value) files.

[4]:
```python
df = pd.read_csv('/kaggle/input/heart-disease-uci/heart.csv')
```

1.7 6. Exploratory Data Analysis

Back to Table of Contents

The scene has been set up. Now let the actual fun begin.

Check shape of the dataset

- It is a good idea to first check the shape of the dataset.

[5]:
```python
# print the shape
print('The shape of the dataset : ', df.shape)
```

```
The shape of the dataset :  (303, 14)
```

Now, we can see that the dataset contains 303 instances and 14 variables.

Preview the dataset

[6]:
```python
# preview dataset
df.head()
```

[6]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0      2.3      0
1   37    1   2       130   250    0        1      187      0      3.5      0
2   41    0   1       130   204    0        0      172      0      1.4      2
3   56    1   1       120   236    0        1      178      0      0.8      2
4   57    0   0       120   354    0        1      163      1      0.6      2

   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

Summary of dataset

[7]:
```python
# summary of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Dataset description

- The dataset contains several columns which are as follows -
  - age : age in years
  - sex : (1 = male; 0 = female)
  - cp : chest pain type
  - trestbps : resting blood pressure (in mm Hg on admission to the hospital)
  - chol : serum cholestoral in mg/dl
  - fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
  - restecg : resting electrocardiographic results
  - thalach : maximum heart rate achieved
  - exang : exercise induced angina (1 = yes; 0 = no)
  - 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 flourosopy
  - thal : 3 = normal; 6 = fixed defect; 7 = reversable defect
  - target : 1 or 0

Check the data types of columns

- The above `df.info()` command gives us the number of filled values along with the data types of columns.
- If we simply want to check the data type of a particular column, we can use the following command.

[8]:
```
df.dtypes
```

[8]:
```
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.

Statistical properties of dataset

[9]:
```
# statistical properties of dataset
df.describe()
```

[9]:
```
              age         sex          cp    trestbps        chol         fbs  \
count  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
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

Important points to note

- The above command `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')`

View column names

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

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

1.8 7. Univariate analysis

Back to Table of Contents

1.8.1 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).
- So, in this section, I will analyze the `target` variable.

Check the number of unique values in `target` variable

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

[11]:
```
2
```

We can see that there are 2 unique values in the `target` variable.

View the unique values in `target` variable

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

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

Comment 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

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

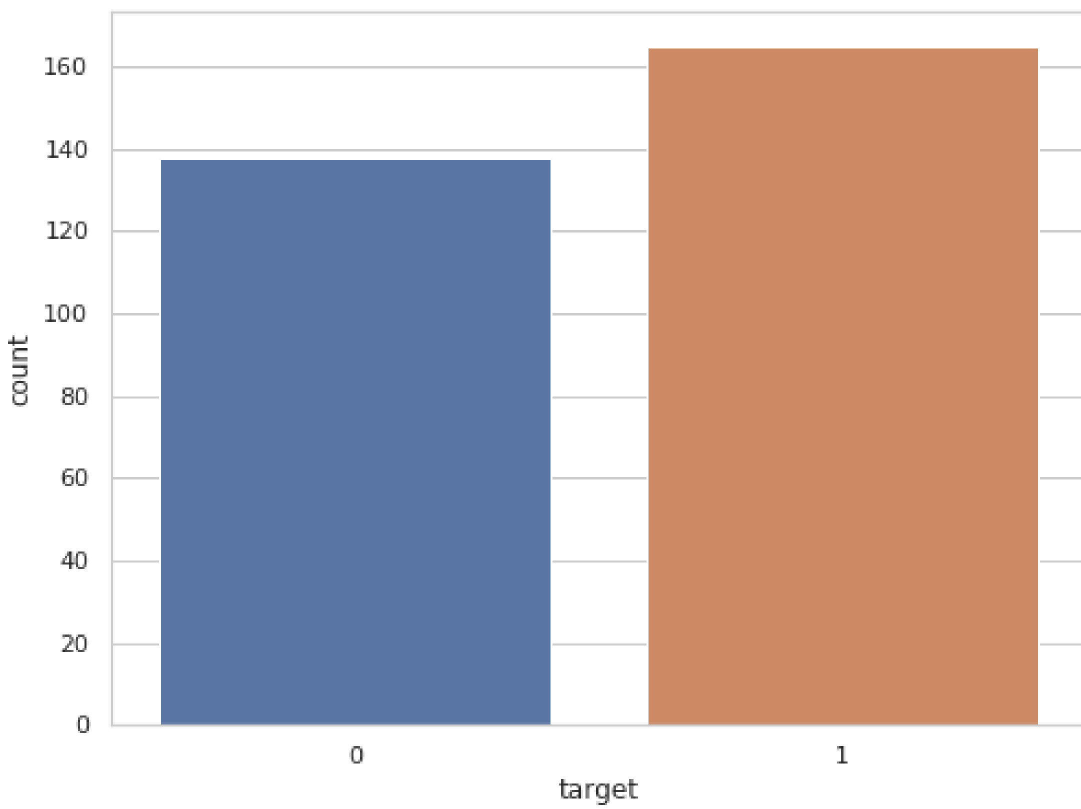
[13]:
```
1    165
0    138
Name: target, dtype: int64
```

Comment

- 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.
- We can visualize this information below.

Visualize frequency distribution of target variable

[14]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="target", data=df)
plt.show()
```



Interpretation

- The above plot confirms the findings that -
  - There are 165 patients suffering from heart disease, and
  - There are 138 patients who do not have any heart disease.

Frequency distribution of `target` variable wrt `sex`

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

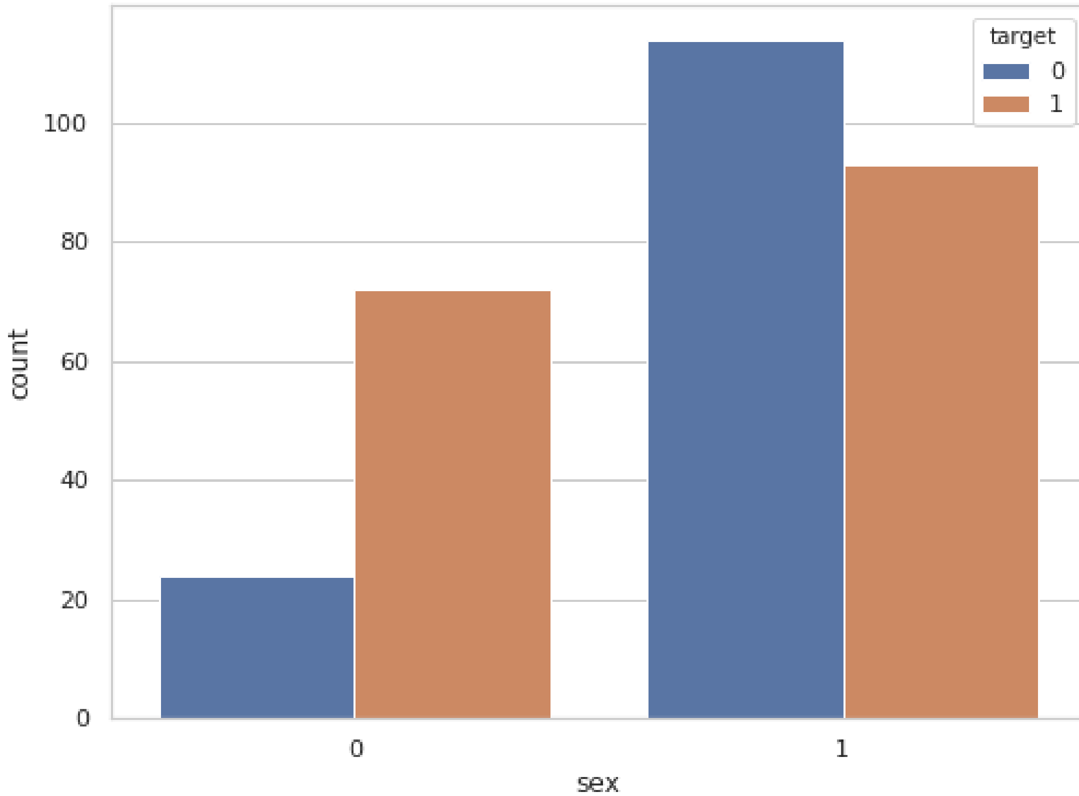
[15]:
```
sex  target
0    1          72
     0          24
1    0         114
     1          93
Name: target, dtype: int64
```

Comment

- `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 -

[16]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="sex", hue="target", data=df)
plt.show()
```

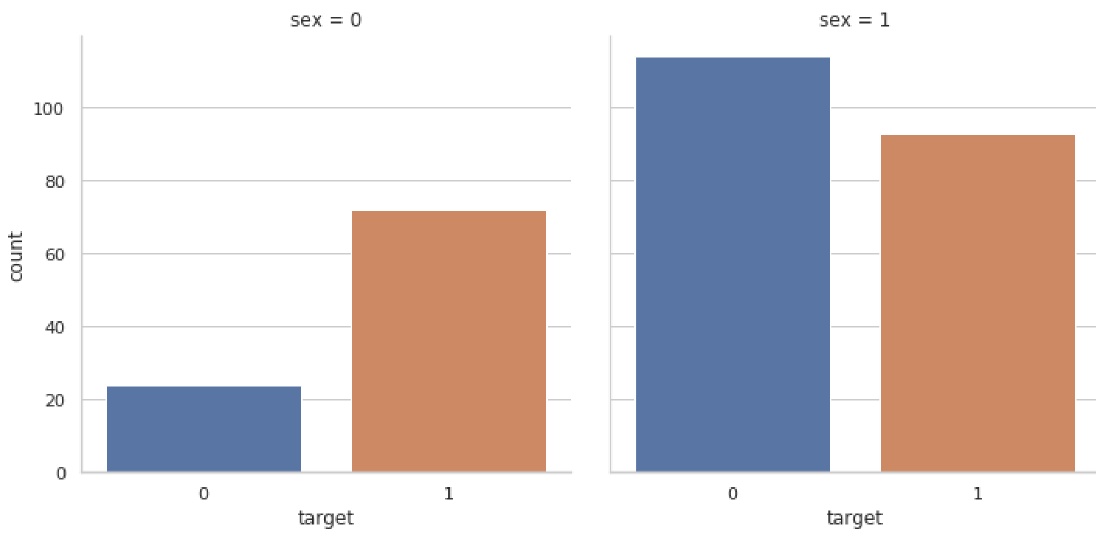


Interpretation

- We can see that the values of `target` variable are plotted wrt `sex` : (1 = male; 0 = female).
- `target` variable also contains two integer values 1 and 0 : (1 = Presence of heart disease; 0 = Absence of heart disease)
- The above plot confirms our findings that -
  - 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.

Alternatively, we can visualize the same information as follows :

[17]:
```
ax = sns.catplot(x="target", col="sex", data=df, kind="count", height=5,␣
↪aspect=1)
```

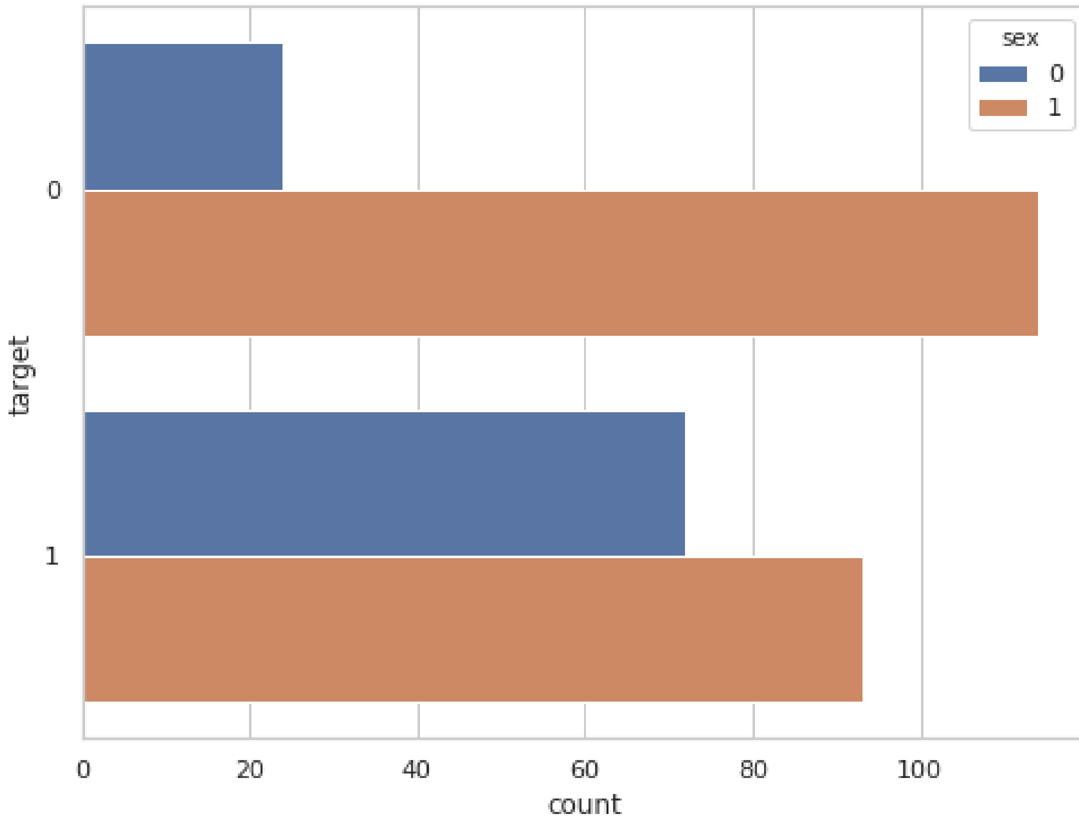


Comment

- The above plot segregate the values of `target` variable and plot on two different columns labelled as (sex = 0, sex = 1).
- I think it is more convinient way of interpret the plots.

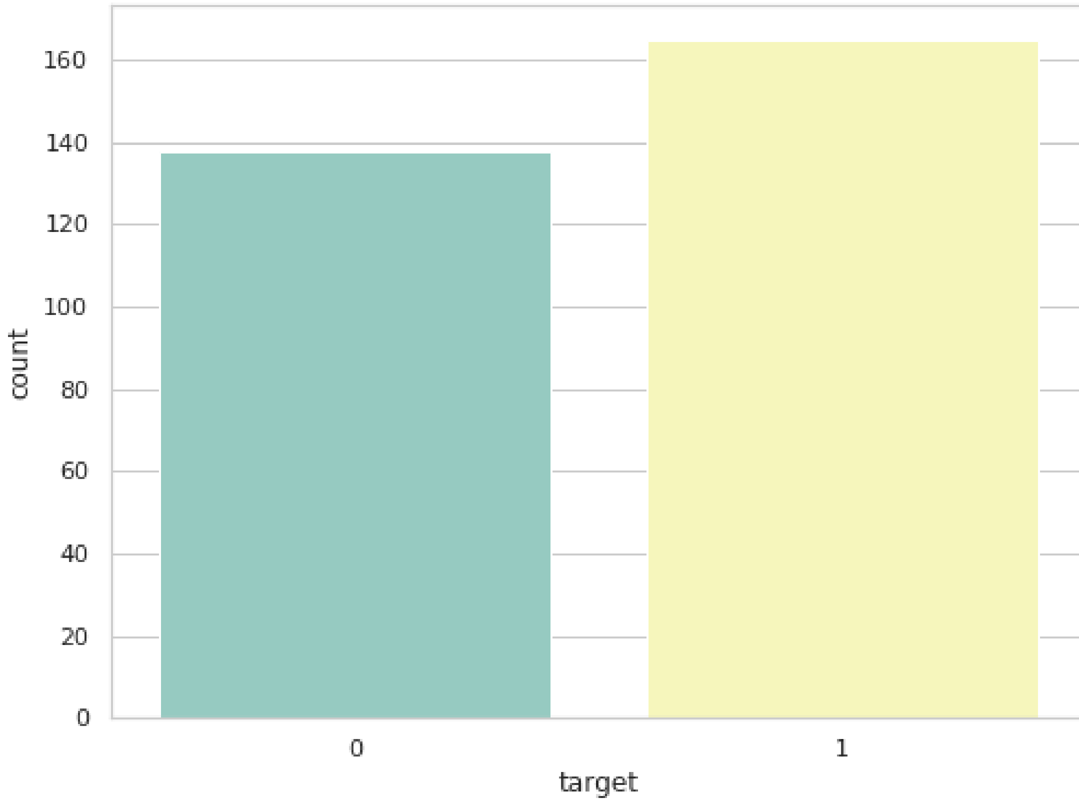
We can plot the bars horizontally as follows :

[18]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(y="target", hue="sex", data=df)
plt.show()
```



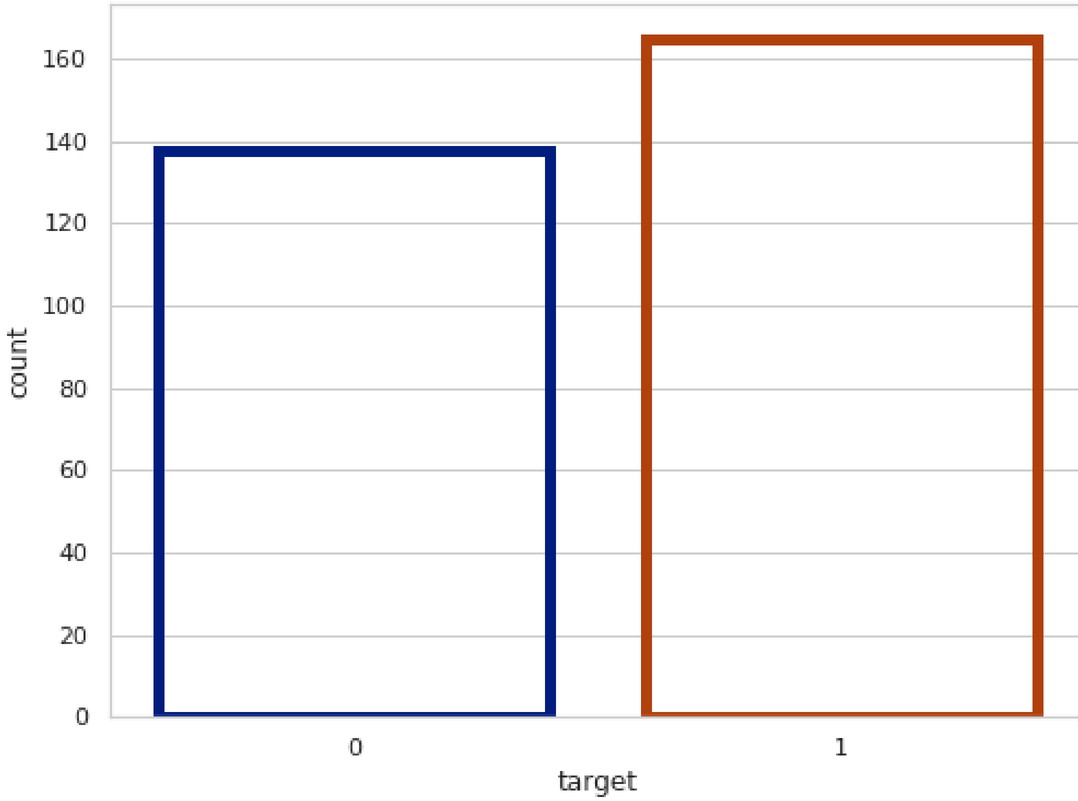
We can use a different color palette as follows :

[19]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="target", data=df, palette="Set3")
plt.show()
```



We can use `plt.bar` keyword arguments for a different look :

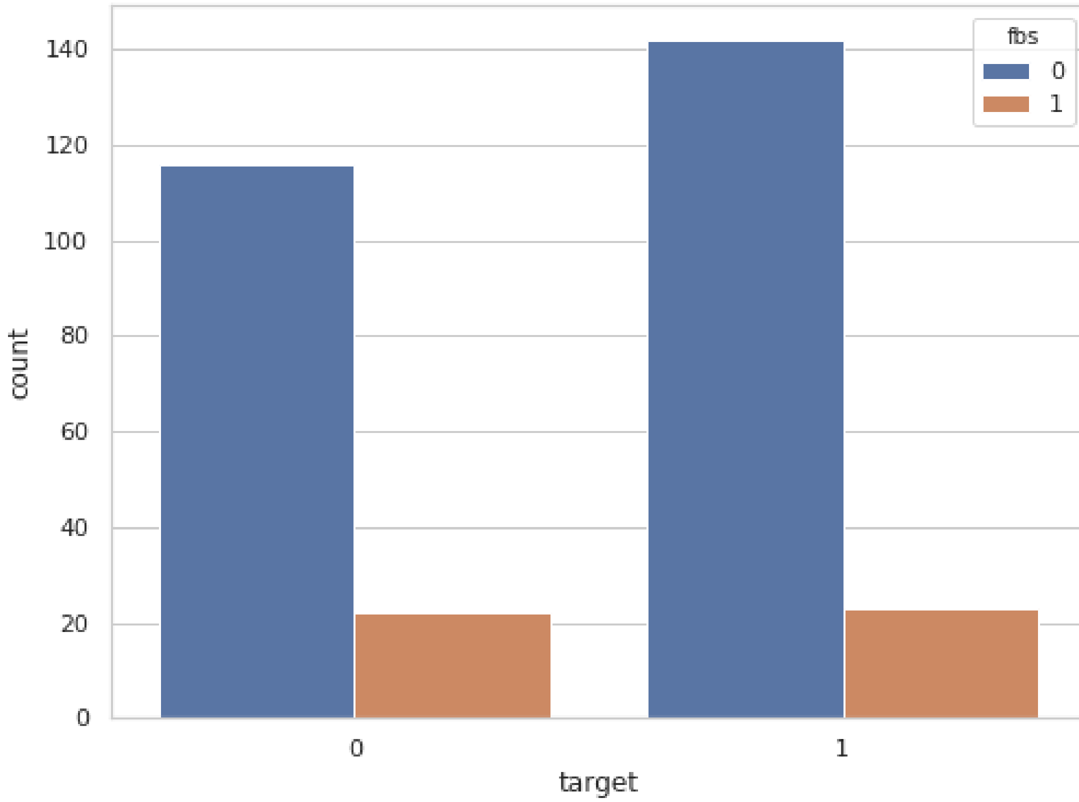
[20]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="target", data=df, facecolor=(0, 0, 0, 0), linewidth=5,␣
 ↪edgecolor=sns.color_palette("dark", 3))
plt.show()
```



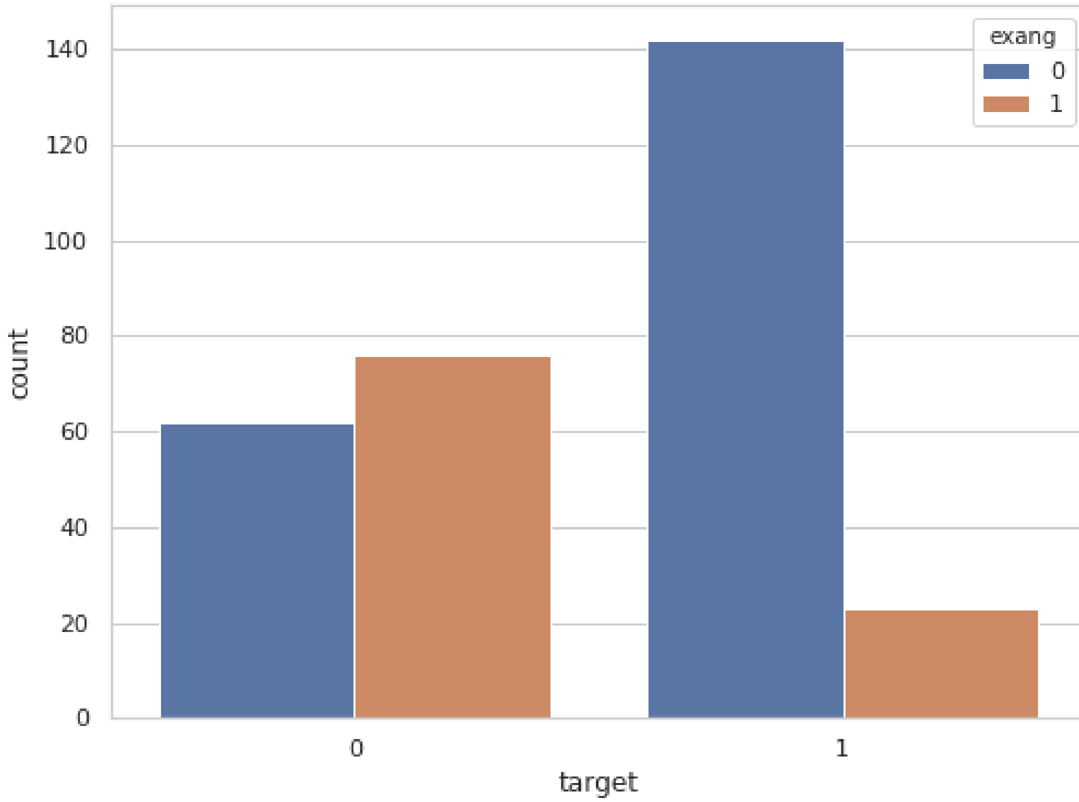
Comment

- I have visualize the `target` values distribution wrt `sex`.
- We can follow the same principles and visualize the `target` values distribution wrt `fbs` `(fasting blood sugar)` and `exang` `(exercise induced angina)`.

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



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



1.8.2 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.

1.9 8. Bivariate Analysis

Back to Table of Contents

1.9.1 Estimate correlation coefficients

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

[23]:
```
correlation = df.corr()
```

The target variable is `target`. So, we should check how each attribute correlates with the `target` variable. We can do it as follows:-

[24]:
```
correlation['target'].sort_values(ascending=False)
```

[24]:
```
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.

1.9.2 Analysis of target and cp variable

Explore cp variable

- `cp` stands for chest pain type.
- First, I will check number of unique values in `cp` variable.

[25]:
```
df['cp'].nunique()
```

[25]:
```
4
```

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

Now, I will view its frequency distribution as follows :

[26]:
```
df['cp'].value_counts()
```

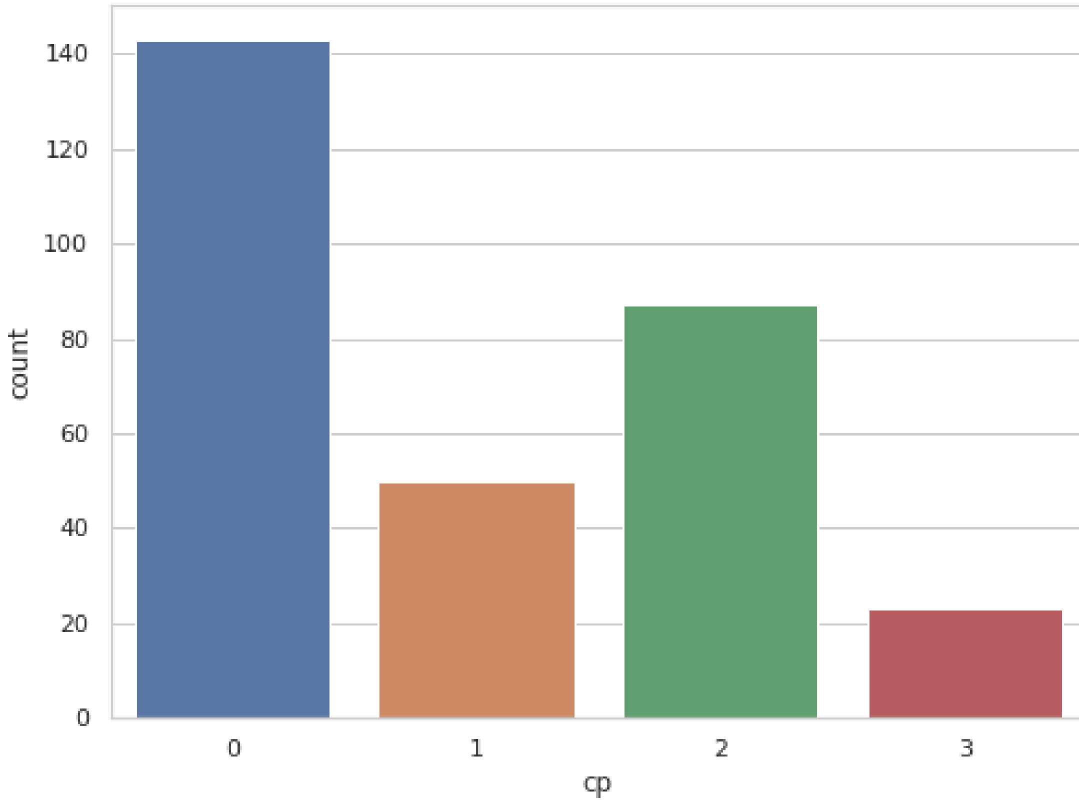
[26]:
```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

Comment

- It can be seen that `cp` is a categorical variable and it contains 4 types of values - 0, 1, 2 and 3.

Visualize the frequency distribution of `cp` variable

[27]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="cp", data=df)
plt.show()
```



Frequency distribution of target variable wrt cp

[28]:
```
df.groupby('cp')['target'].value_counts()
```

[28]:
```
cp  target
0   0         104
    1          39
1   1          41
    0           9
2   1          69
    0          18
3   1          16
    0           7
Name: target, dtype: int64
```

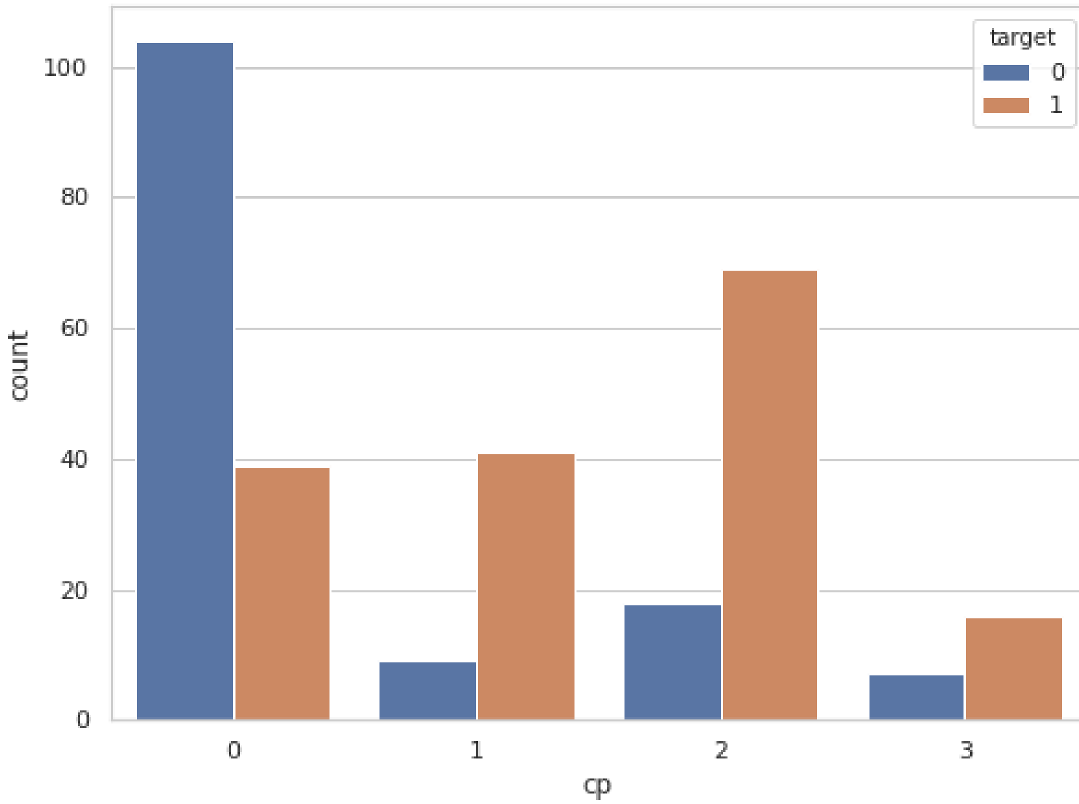
Comment

- `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 this information below.

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

[29]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.countplot(x="cp", hue="target", data=df)
plt.show()
```

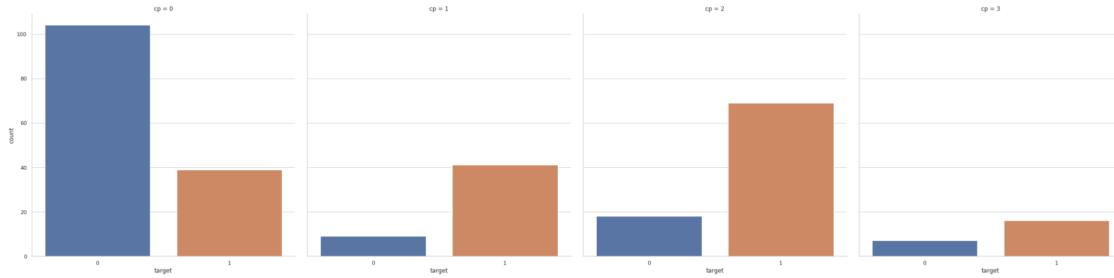


Interpretation

- 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 :

[30]:
```
ax = sns.catplot(x="target", col="cp", data=df, kind="count", height=8,␣
 ↪aspect=1)
```



1.9.3 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 :

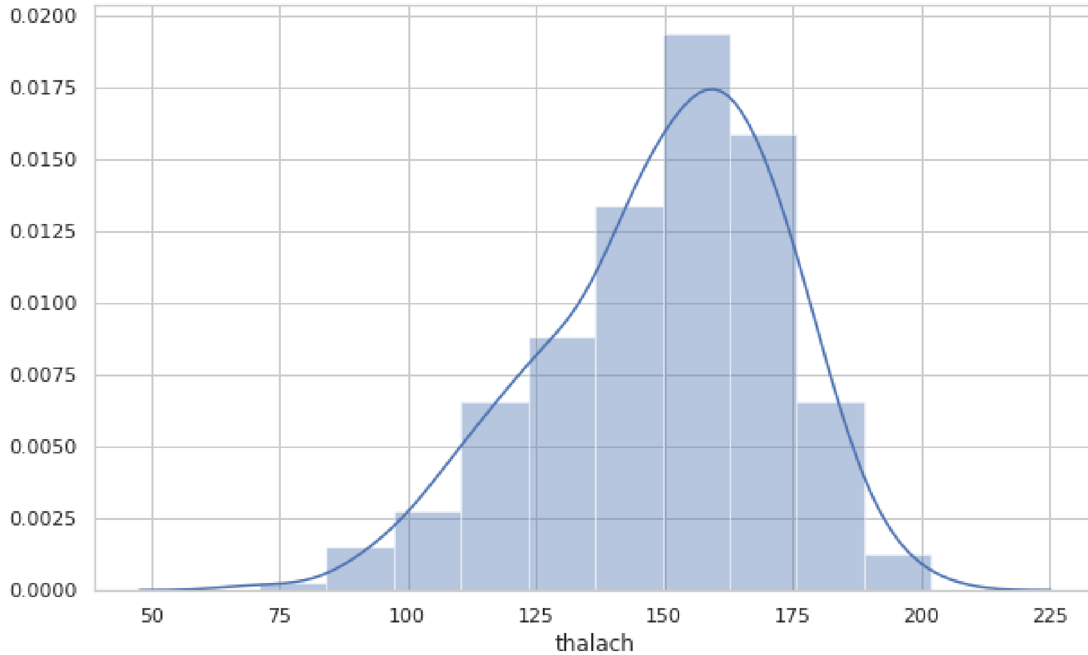
[31]:
```
df['thalach'].nunique()
```

[31]: 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

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

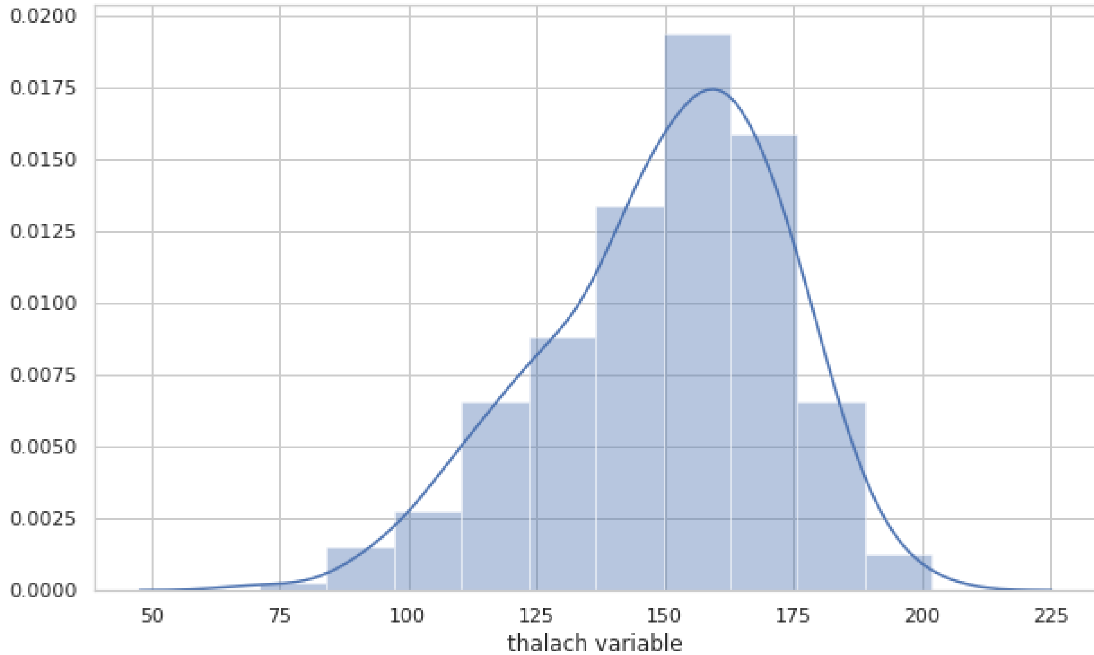


Comment

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

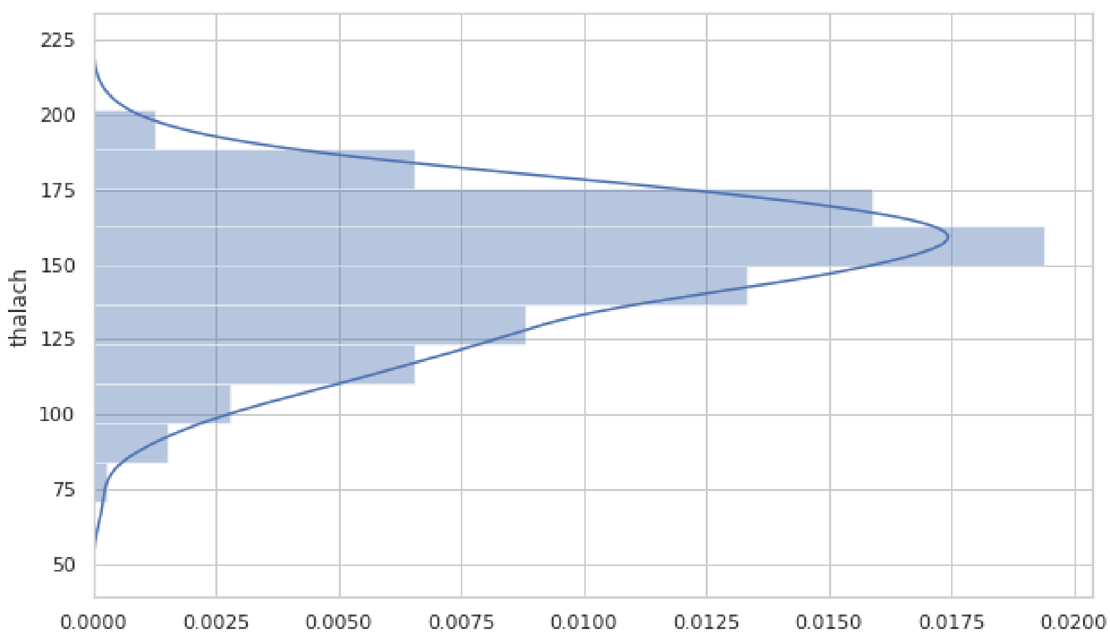
We can use Pandas series object to get an informative axis label as follows :

[33]:
```
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:-

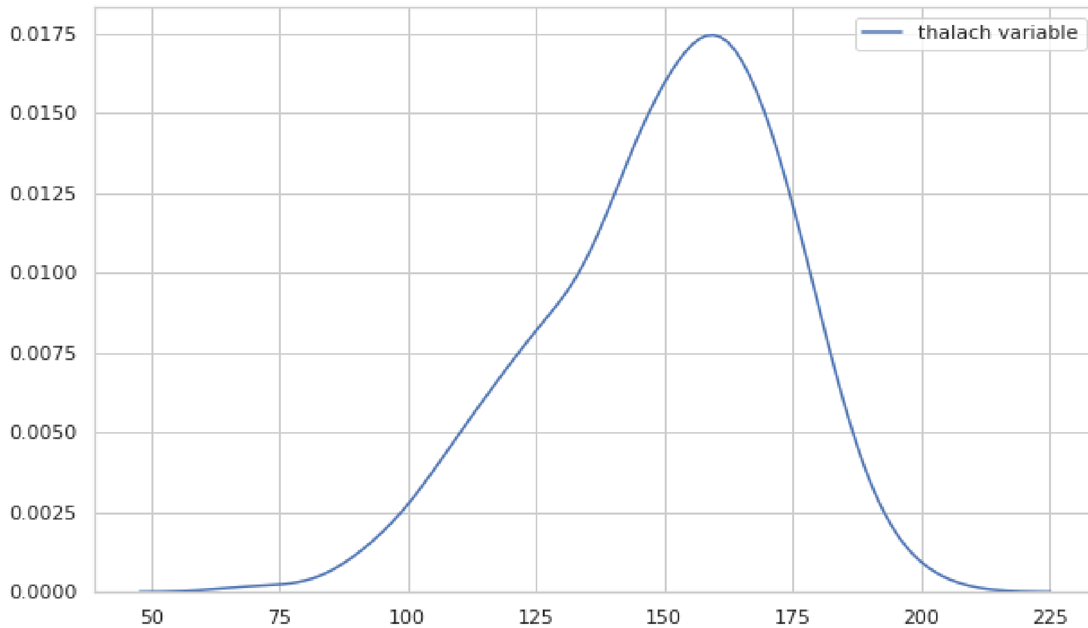
[34]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
ax = sns.distplot(x, bins=10, vertical=True)
plt.show()
```



Seaborn Kernel Density Estimation (KDE) Plot

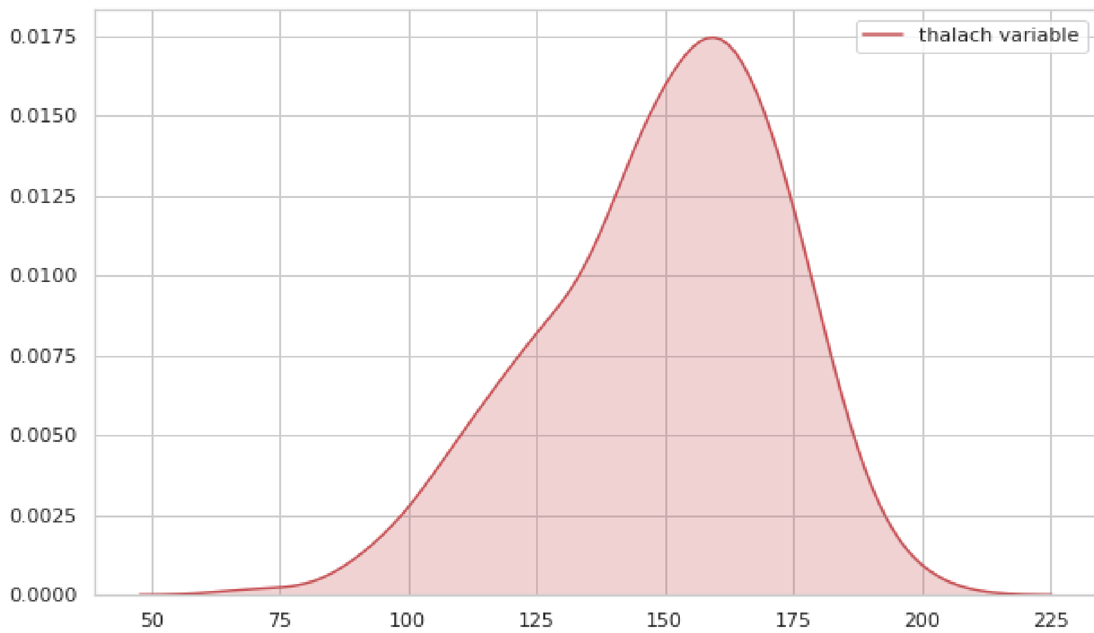
- The kernel density estimate (KDE) plot is a useful tool for plotting the shape of a distribution.
- The KDE plot plots the density of observations on one axis with height along the other axis.
- We can plot a KDE plot as follows :

[35]:
```
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:

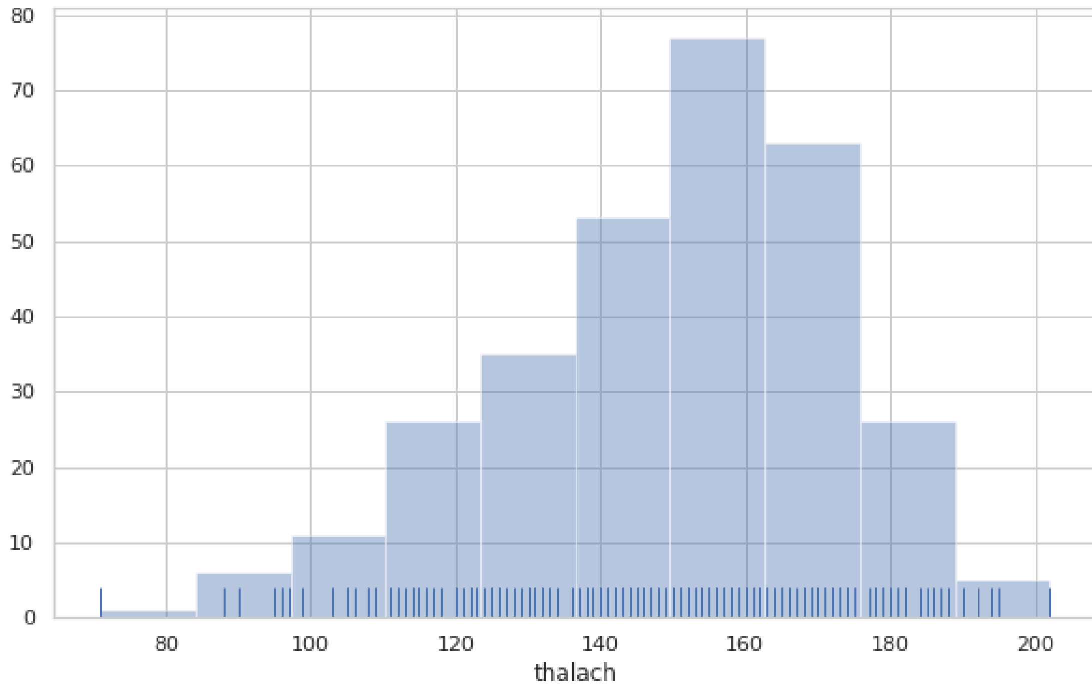
[36]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['thalach']
x = pd.Series(x, name="thalach variable")
ax = sns.kdeplot(x, shade=True, color='r')
plt.show()
```



Histogram

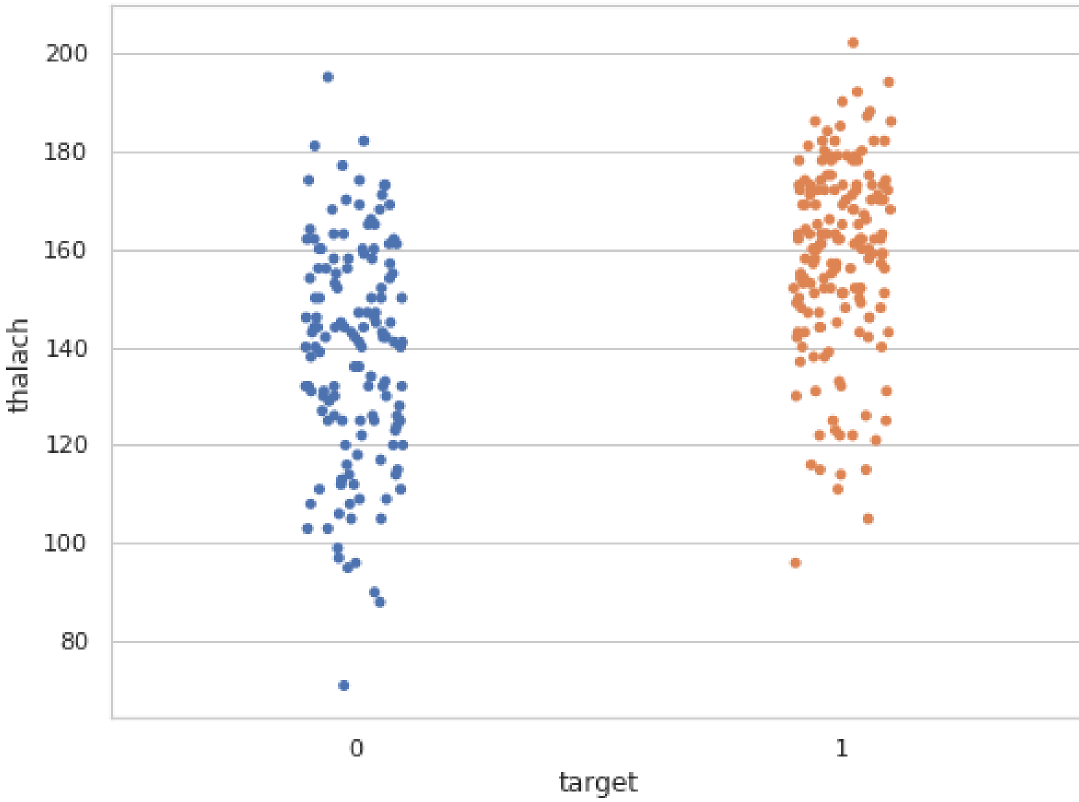
- A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.
- We can plot a histogram as follows :

[37]:
```
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`

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

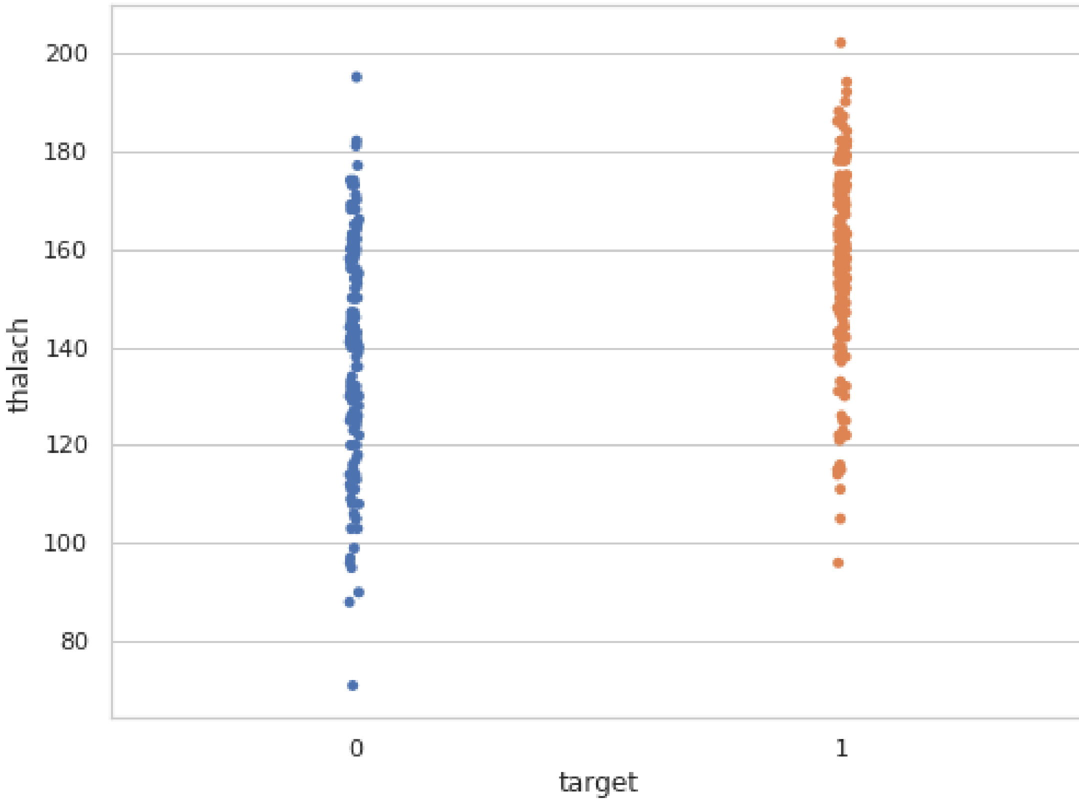


Interpretation

- 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).

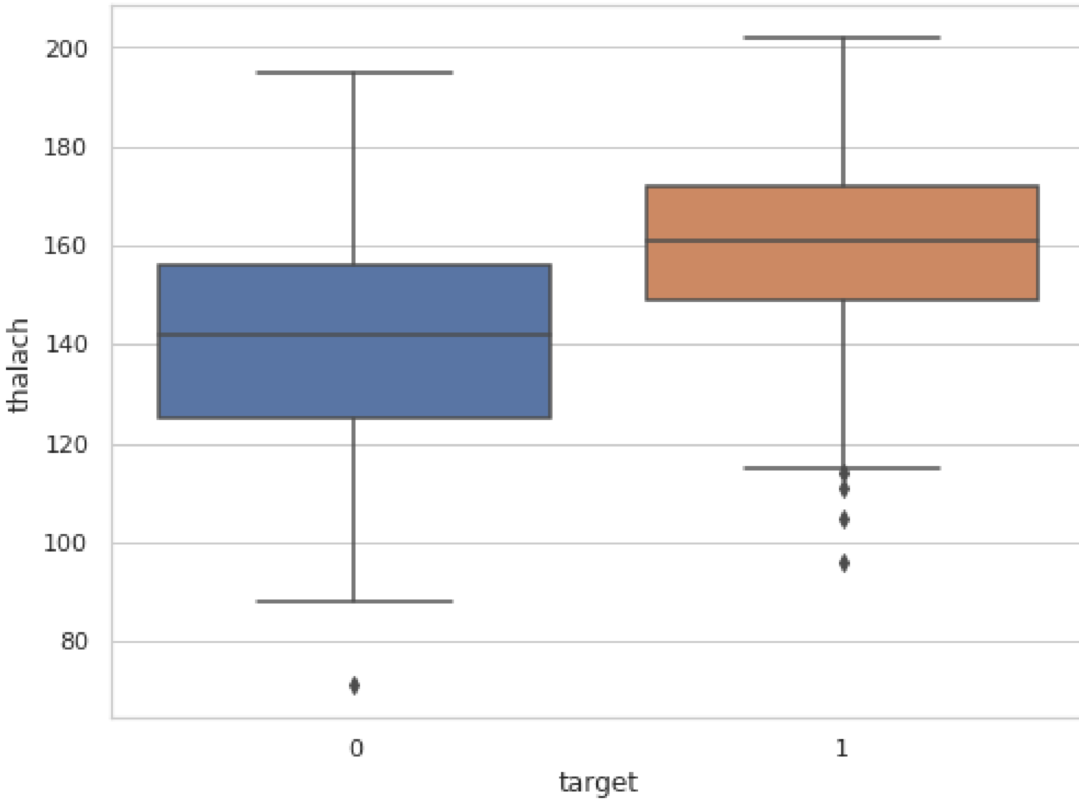
We can add jitter to bring out the distribution of values as follows :

[39]:
```
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

[40]:
```
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).

1.9.4 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).

1.10 9. Multivariate analysis

Back to Table of Contents

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

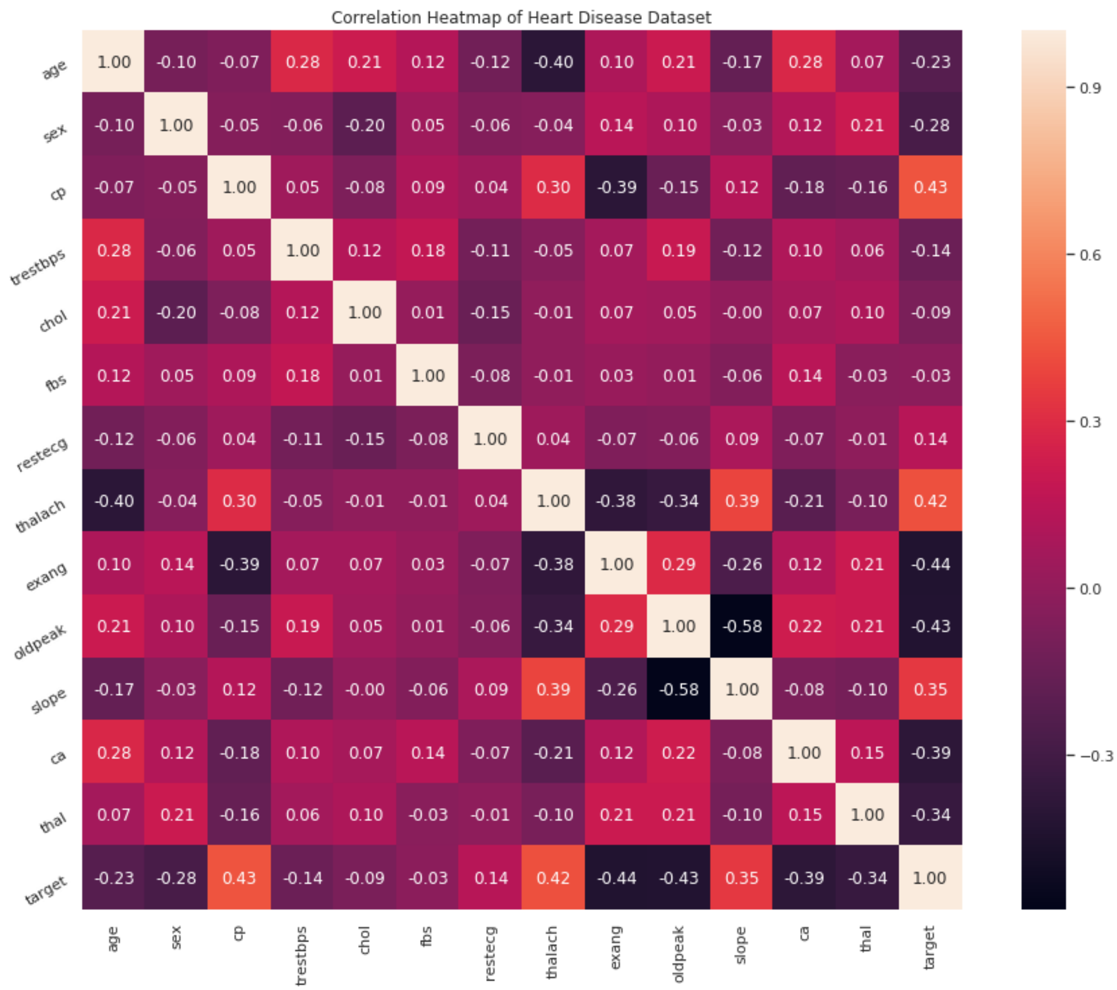
1.10.1 Discover patterns and relationships

- 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`.

1.10.2 Heat Map

[41]:

```
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()
```

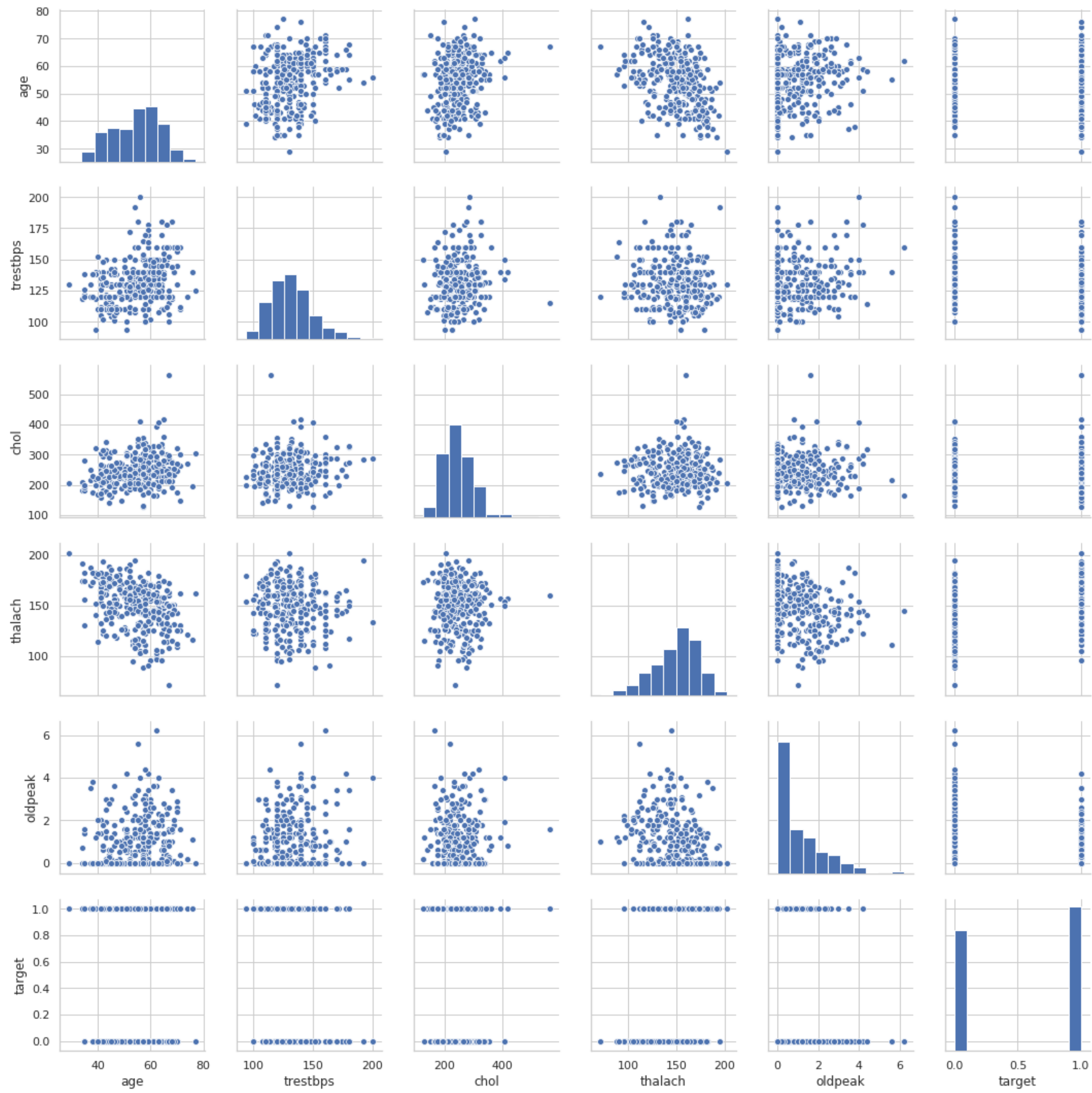


Interpretation From the above correlation heat map, we can conclude that :-

- `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).

1.10.3 Pair Plot

[42]:
```
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.

1.10.4 Analysis of `age` and other variables

Check the number of unique values in `age` variable

[43]:
```
df['age'].nunique()
```

[43]:
```
41
```

View statistical summary of `age` variable

[44]:
```
df['age'].describe()
```

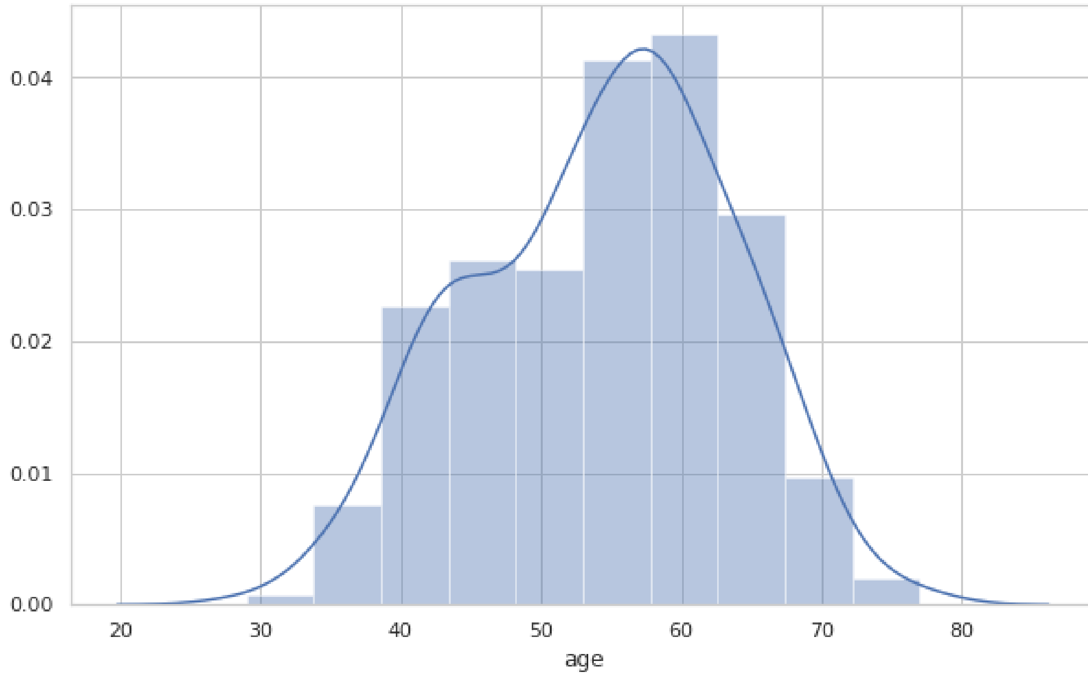
[44]:
```
count    303.000000
mean      54.366337
std        9.082101
min       29.000000
25%       47.500000
50%       55.000000
75%       61.000000
max       77.000000
Name: age, dtype: float64
```

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.

[45]:
```
f, ax = plt.subplots(figsize=(10,6))
x = df['age']
ax = sns.distplot(x, bins=10)
plt.show()
```



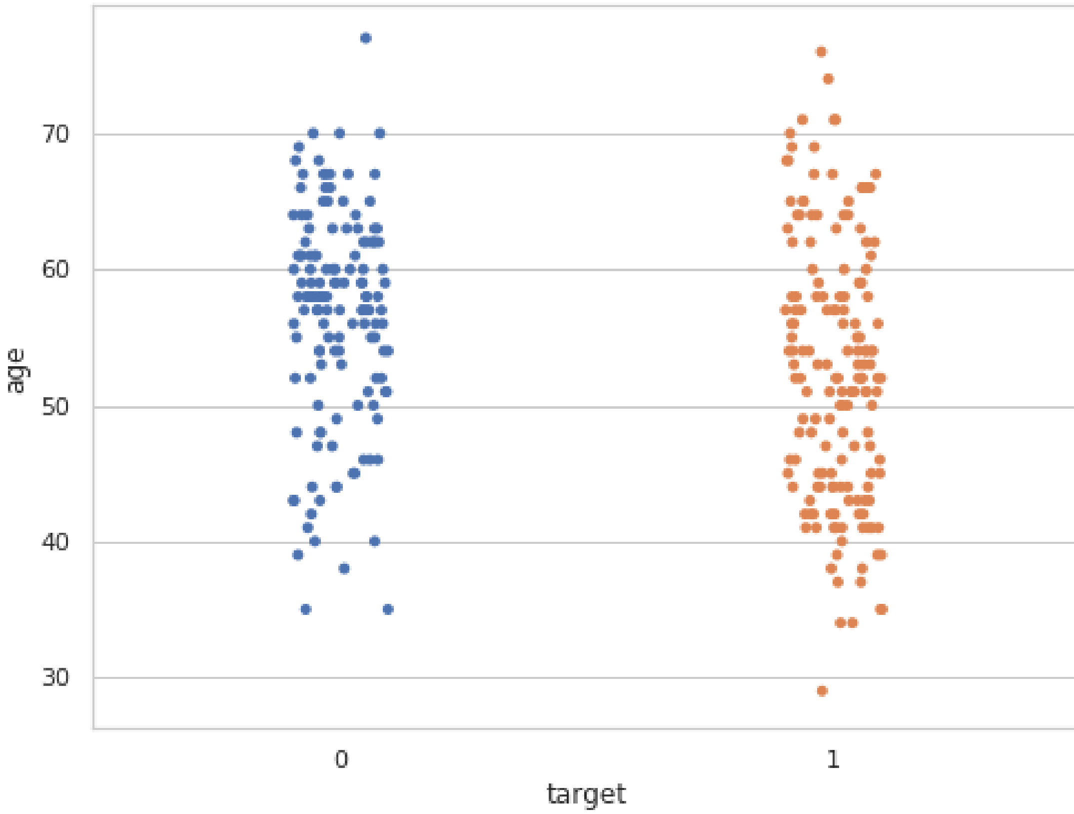
Interpretation

- The `age` variable distribution is approximately normal.

1.10.5 Analyze `age` and `target` variable

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

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

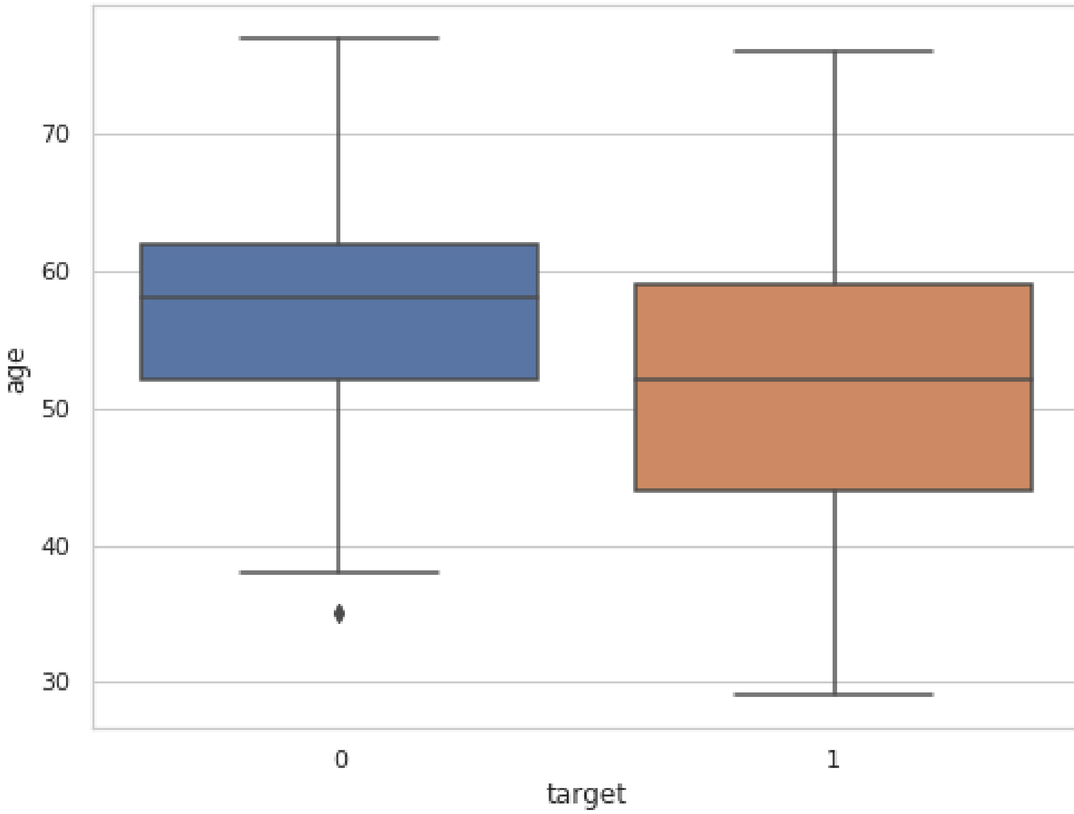


Interpretation

- We can see that the people suffering from heart disease (target = 1) and people who are not suffering from heart disease (target = 0) have comparable ages.

Visualize distribution of age variable wrt target with boxplot

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



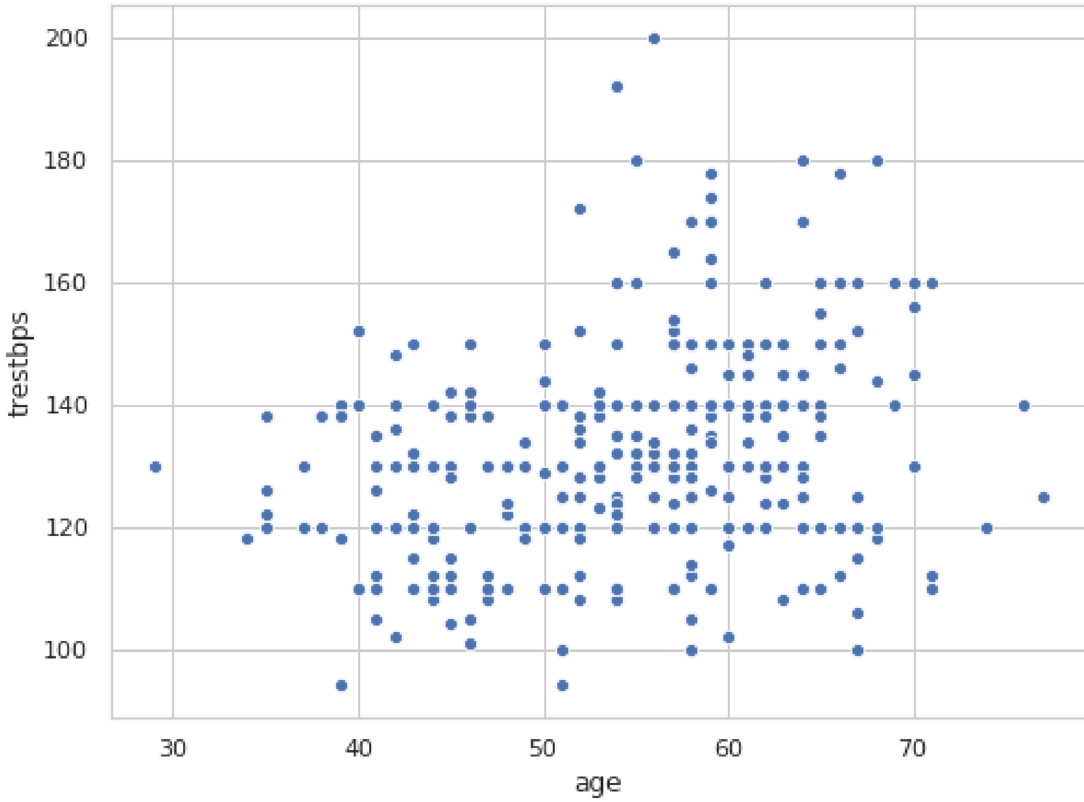
Interpretation

- 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.

1.10.6 Analyze `age` and `trestbps` variable

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

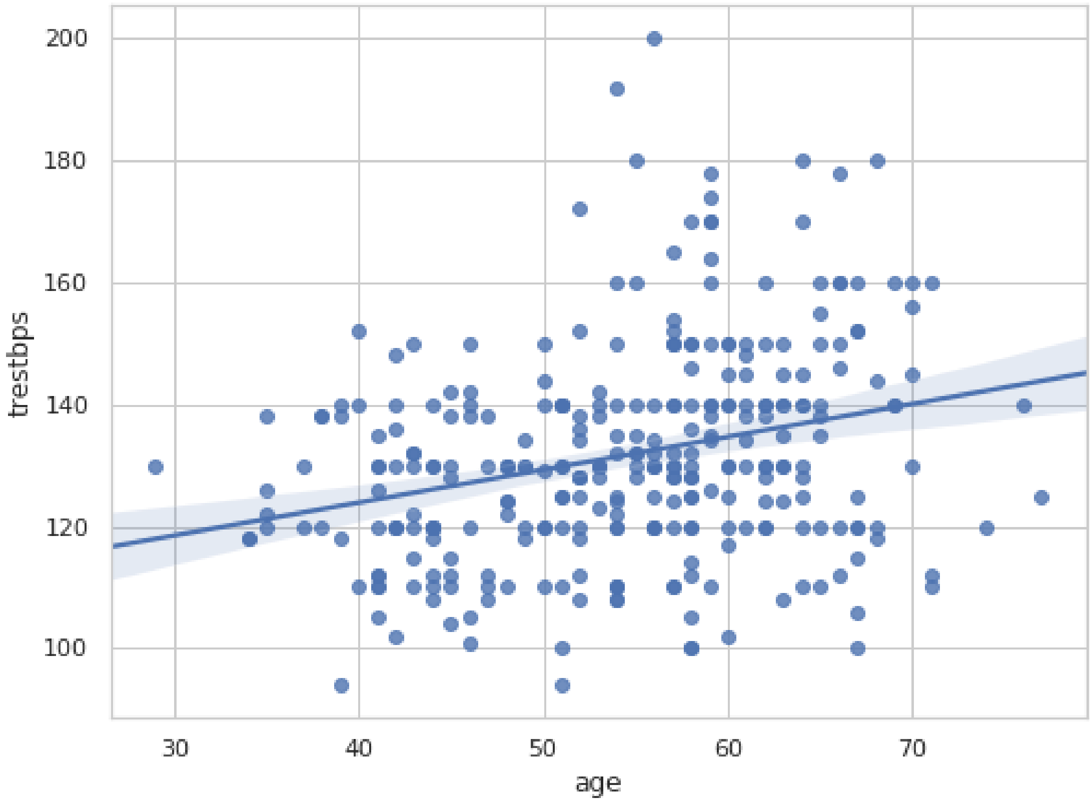
[48]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.scatterplot(x="age", y="trestbps", data=df)
plt.show()
```



Interpretation

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

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

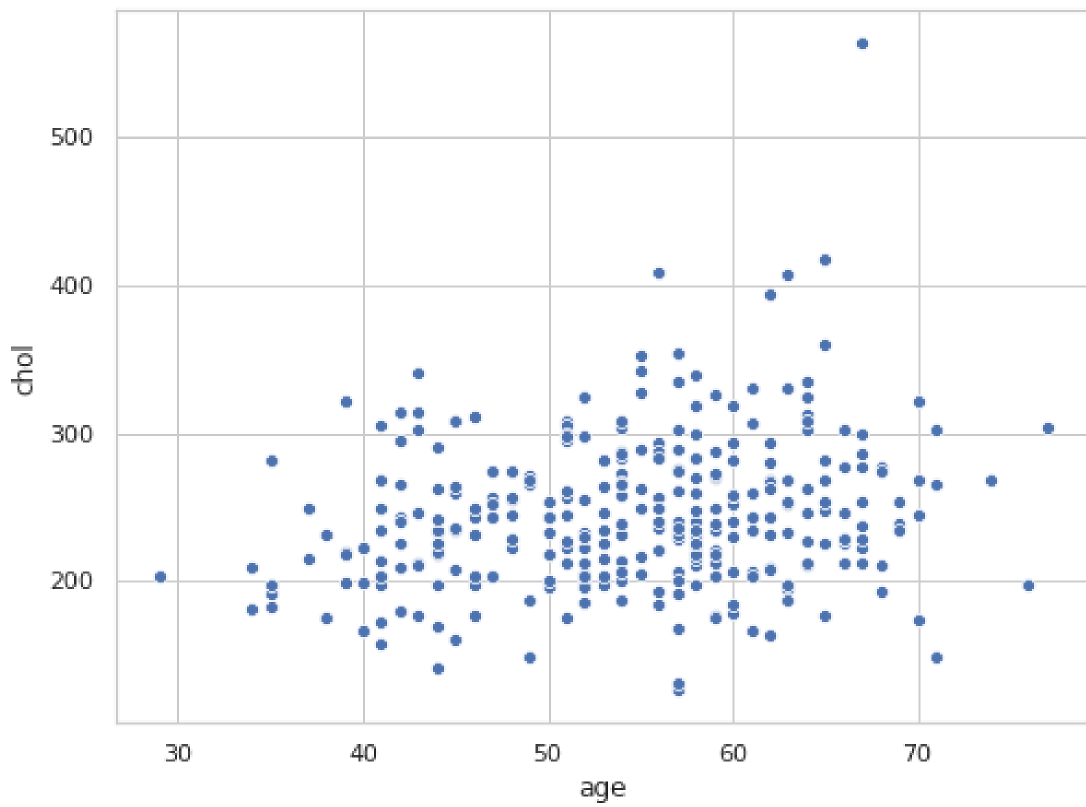


Interpretation

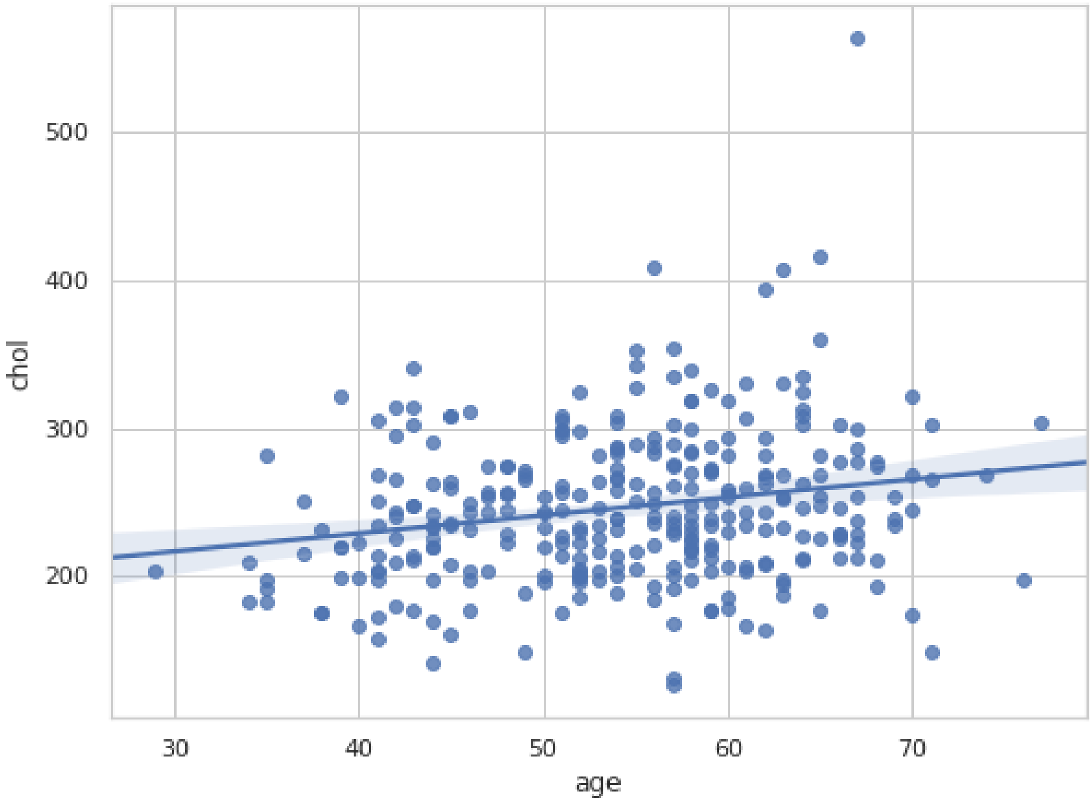
- The above line shows that linear regression model is not good fit to the data.

1.10.7 Analyze age and chol variable

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



[51]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="age", y="chol", data=df)
plt.show()
```

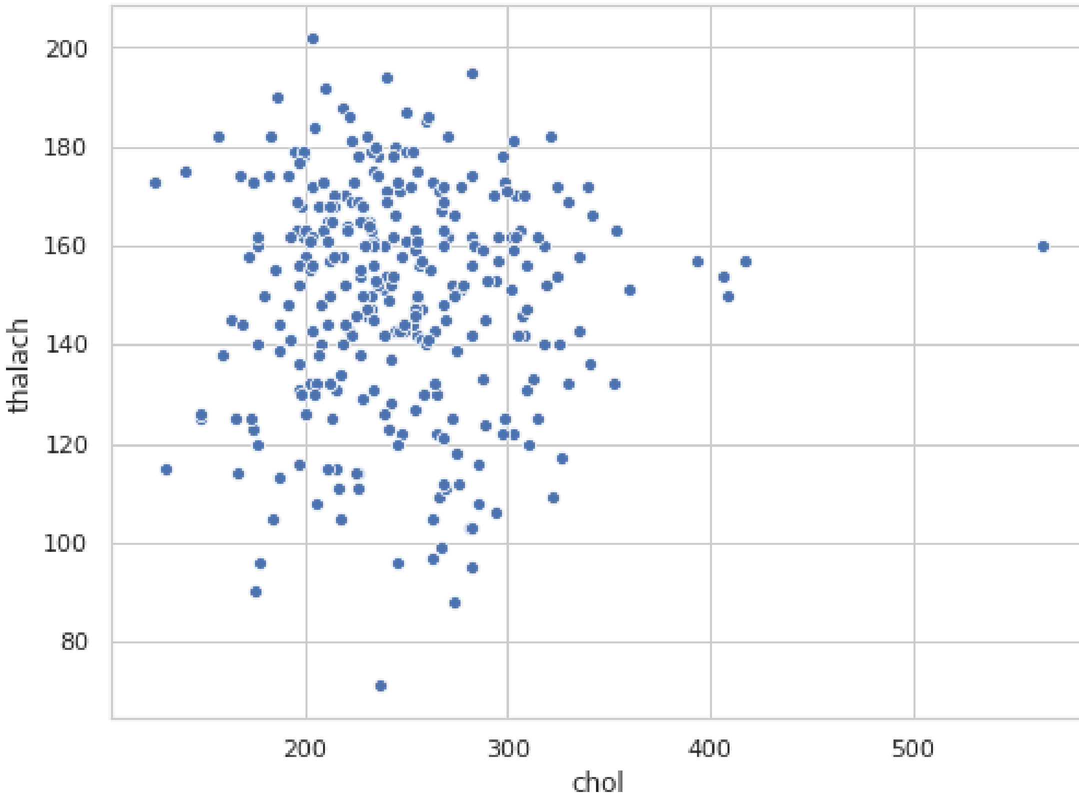


Interpretation

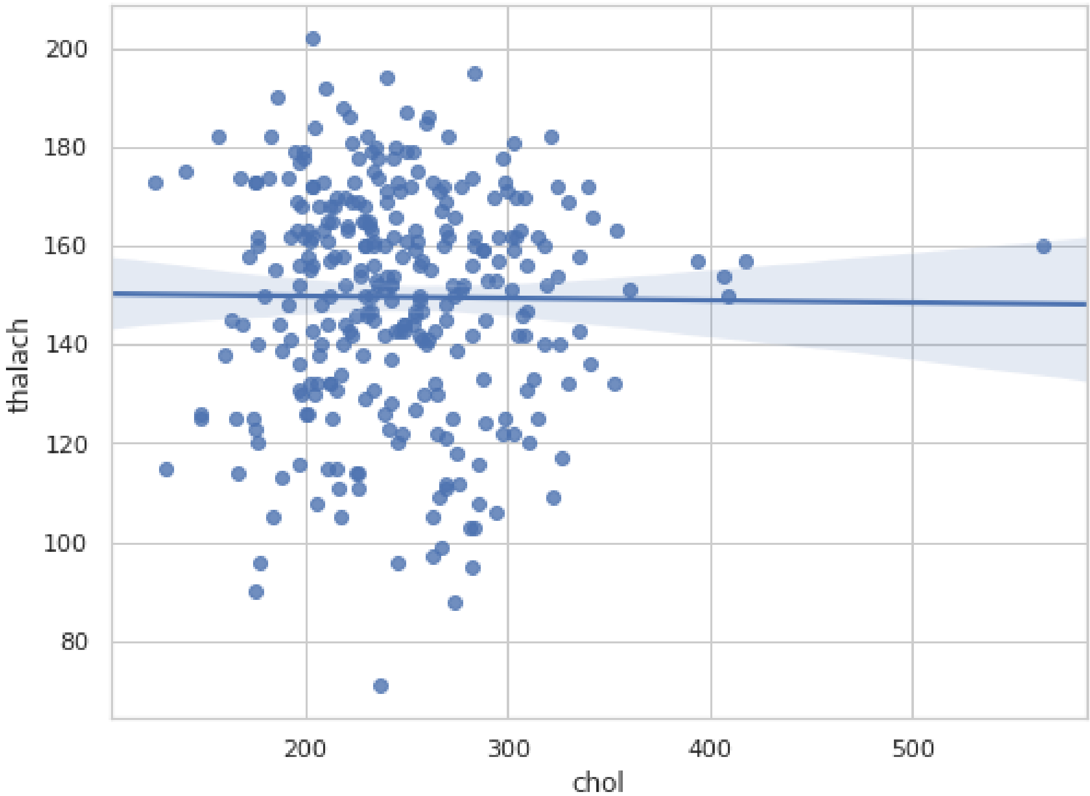
- The above plot confirms that there is a slighly positive correlation between `age` and `chol` variables.

1.10.8 Analyze `chol` and `thalach` variable

[52]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.scatterplot(x="chol", y = "thalach", data=df)
plt.show()
```



[53]:
```
f, ax = plt.subplots(figsize=(8, 6))
ax = sns.regplot(x="chol", y="thalach", data=df)
plt.show()
```



Interpretation

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

1.11 10. 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.

1.11.1 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.

1.11.2 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.

[54]:
```python
# check for missing values

df.isnull().sum()
```

[54]:
```
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
```

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

1.12 11. 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)

[55]:
```
#assert that there are no missing values in the dataframe

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

[56]:
```
#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.

1.13 12. Outlier detection

Back to Table of Contents

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

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

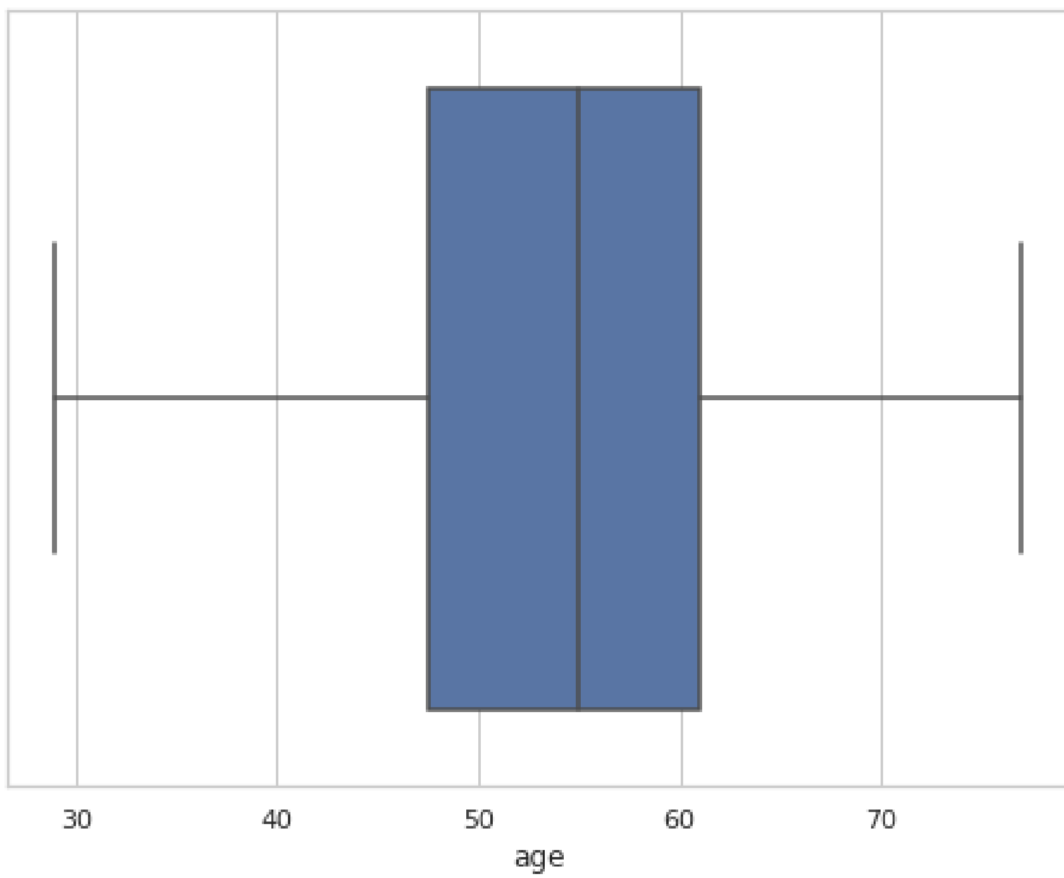
1.13.1 age variable

[57]:
```
df['age'].describe()
```

[57]:
```
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

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



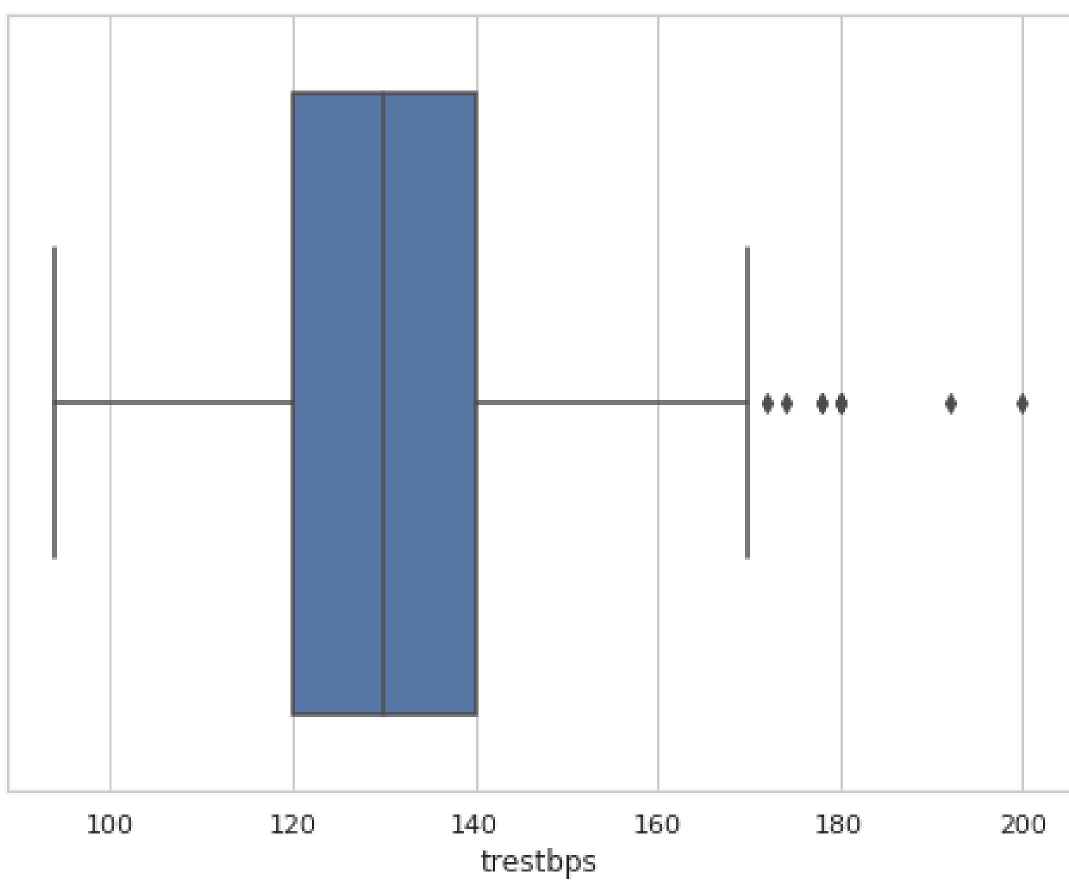
1.13.2 trestbps variable

[59]:
```
df['trestbps'].describe()
```

[59]:
```
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

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



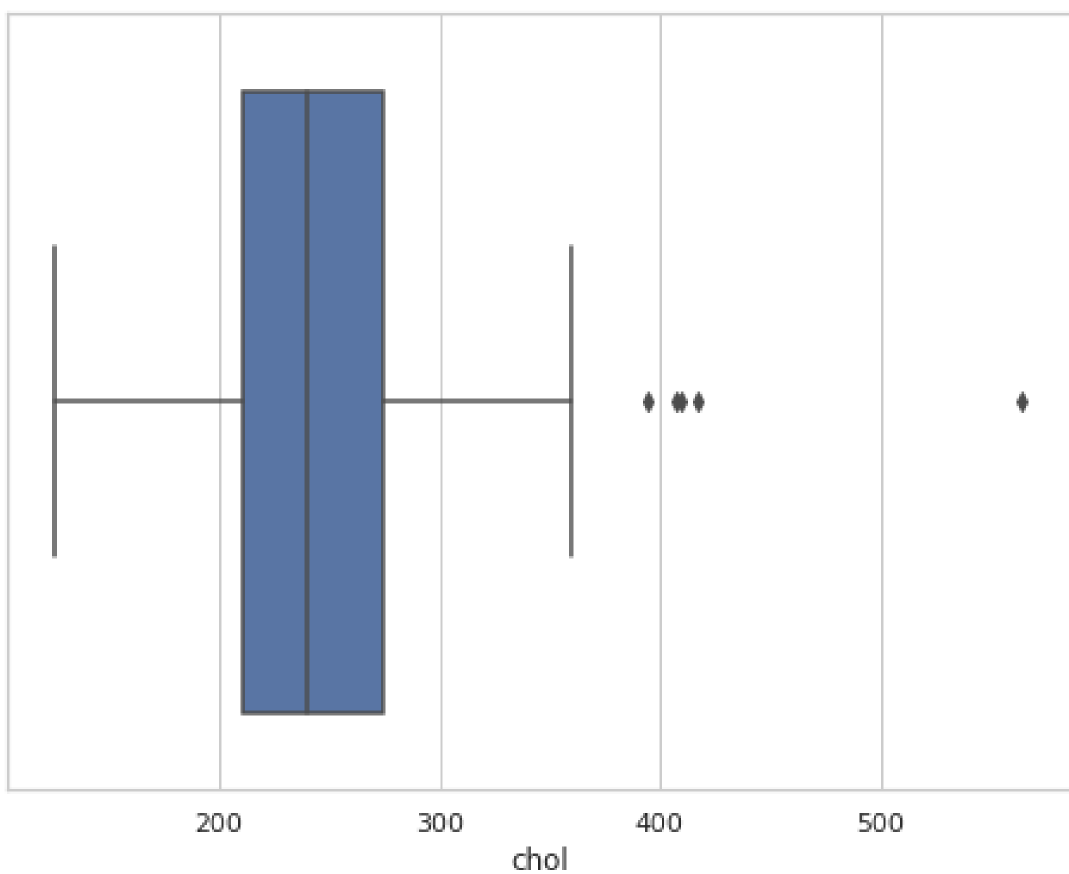
1.13.3 chol variable

[61]:
```
df['chol'].describe()
```

[61]:
```
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

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



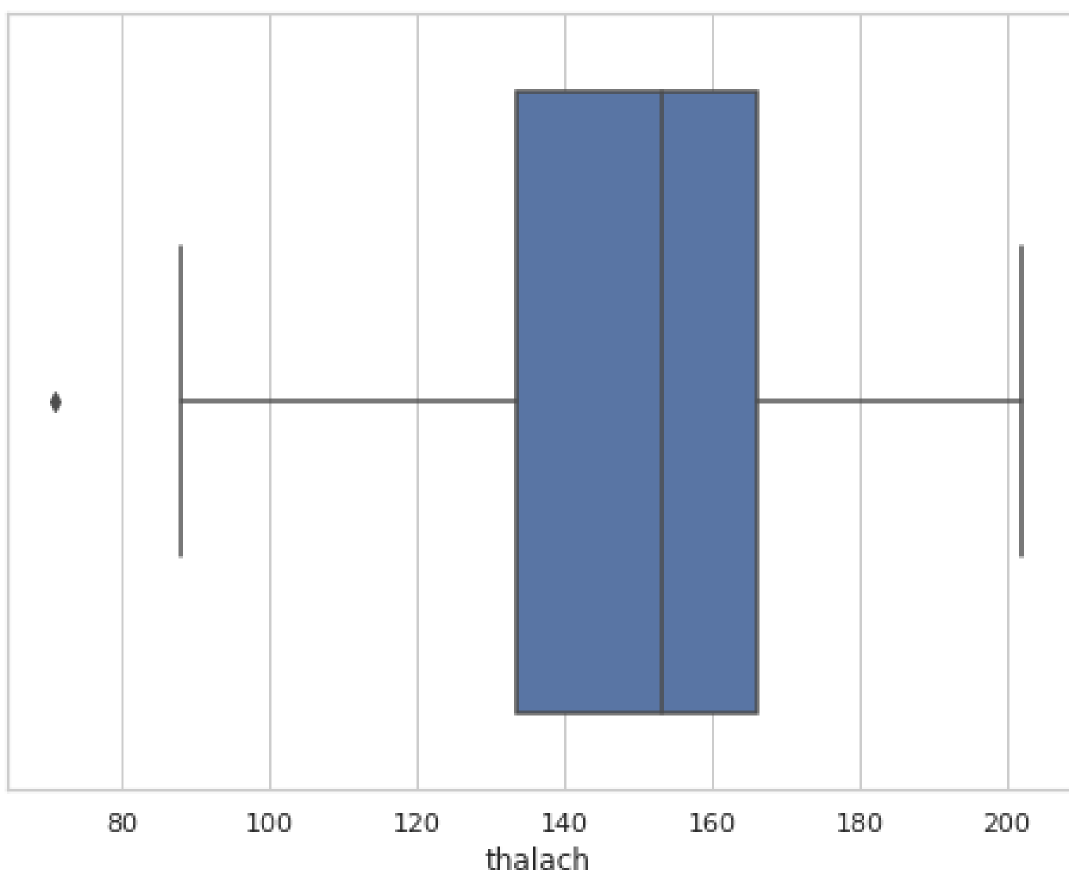
1.13.4 thalach variable

[63]:
```
df['thalach'].describe()
```

[63]:
```
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

[64]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x=df["thalach"])
plt.show()
```



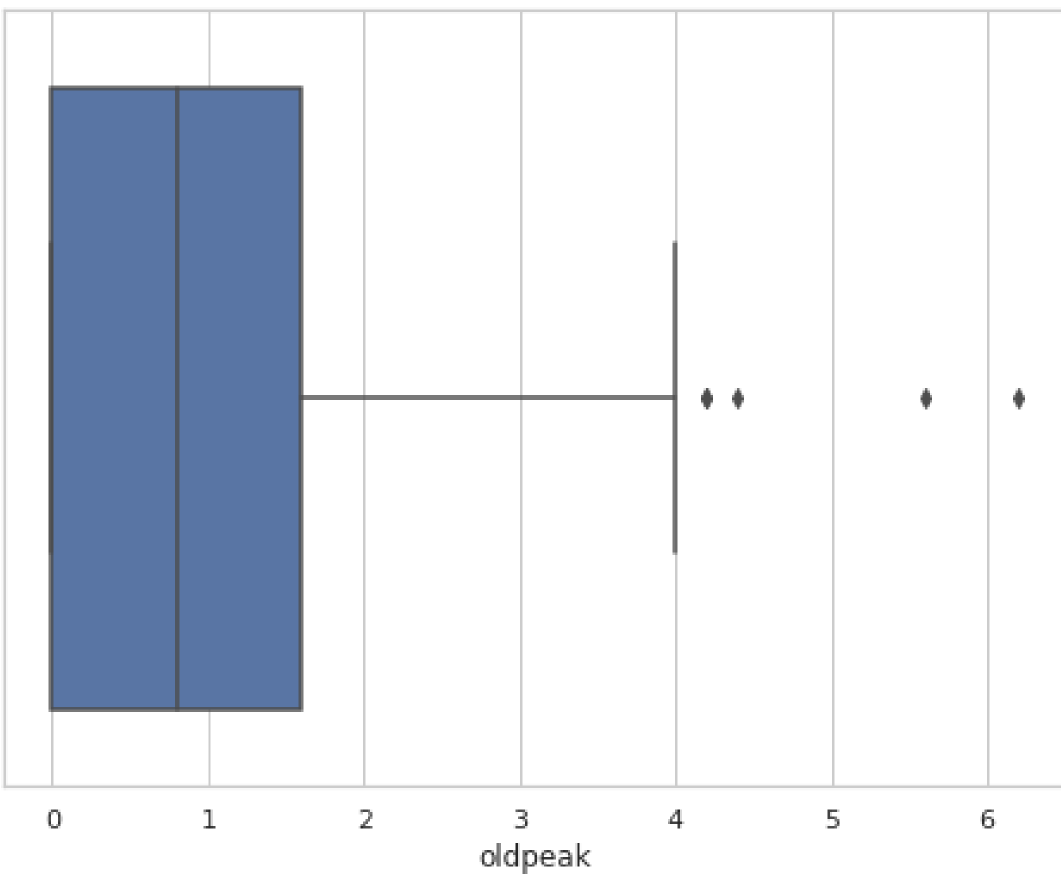
1.13.5 oldpeak variable

[65]:
```
df['oldpeak'].describe()
```

[65]:
```
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

[66]:
```
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.

1.14 13. Conclusion

Back to Table of Contents

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

1.15 14. References

Back to Table of Contents

The following references are used to create this kernel

- Think Stats - Exploratory Data Analysis in Python by Allen B Downey
- Seaborn API reference
- My other kernel

Go to Top