Data Analytics with Python

Course End Project

Heart Data Analysis

HETAL SANGANI

Data Analyst

Masters Capstone

Get Certified. Get Ahead.

Table of Contents:

Understanding Business Problem:

Cardiovascular diseases are one of the leading causes of deaths globally. To identify the causes and develop a system to predict potential heart attacks in an effective manner is necessary. The data presented has all the information about relevant factors that might have an impact on cardiovascular health. The data needs to be studied in detail for further analysis.

Objective: To analyze the dataset that will help to create a model that will predict the heart disease based on various input features

Domain: Healthcare

Dataset: Heart dataset (data.xlsx)

Variable Description:

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 cholesterol 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

Data Understanding:

importing necessary library required and understanding the dataset

```
# import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

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

```
# Load the data
df = pd.read_excel("data.xlsx")
```

```
# Print first few rows of data
df.head()
```

	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

df.head() returns the first 5 rows of the dataset and df.tail() to get last 5 rows.

```python
# Display the column names
df.columns
```

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

```python
# Shape of the data(rows, columns)
df.shape
```

```
(303, 14)
```

```python
# Summary of the data
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
```

df.info() method returns information about the whole DataFrame including the index data type and columns, non-null values, and memory usage.

```python
# Basic statistics for numeric columns(Descriptive statistics)
df.describe()
```

	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	303.00000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.72937
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.02260
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.00000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.00000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.00000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.00000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.00000

df.describe() method generates descriptive statistics for us. For numeric data, the result's index will include count, mean, std, min, max as well as lower, 50(median) and upper percentiles.

We can easily notice that the minimum age is 29 and the maximum age is 77. We can also see mean and median values of age are almost the same.

Data Cleaning:

In this part of the EDA. We will check :

- Missing Values
- Duplicated Values

The purpose of data cleaning is to get our data ready to analyze and visualize.

```
# Check for missing values
df.isnull().sum()
```

```
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
```

When combining .isnull() method with .sum() we can sum up all the missing values for each column.

There are no missing values in this dataset. We will now proceed to analyze the data, observe patterns, and identify outliers with the help of visualization methods.

Now we will check for duplicated values.

```
# Check duplicated values
df[df.duplicated()]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2	1

Dataset has only one duplicated row. We can simply drop this row using the drop_duplicates() method.

```python
# Drop the duplicated rows
df.drop_duplicates()
```

	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

302 rows × 14 columns

Data Visualization:

Now, We understood our dataset in general and checked the missing values. We also deleted duplicated values from the data frame.

The next part is data visualization! We have to perform univariate, bivariate and multivariate analysis to see the distribution and relationship between variables.

We will use the seaborn library for statistical data visualization. Seaborn is a data visualization library based on matplotlib.

Univariate Analysis

The purpose of the univariate analysis is to understand the distribution of values for a single variable.

We will perform univariate analysis by using visualization techniques.

Univariate Analysis for Numerical Features

```python
# Unique values in the target variable
df['target'].value_counts()
```

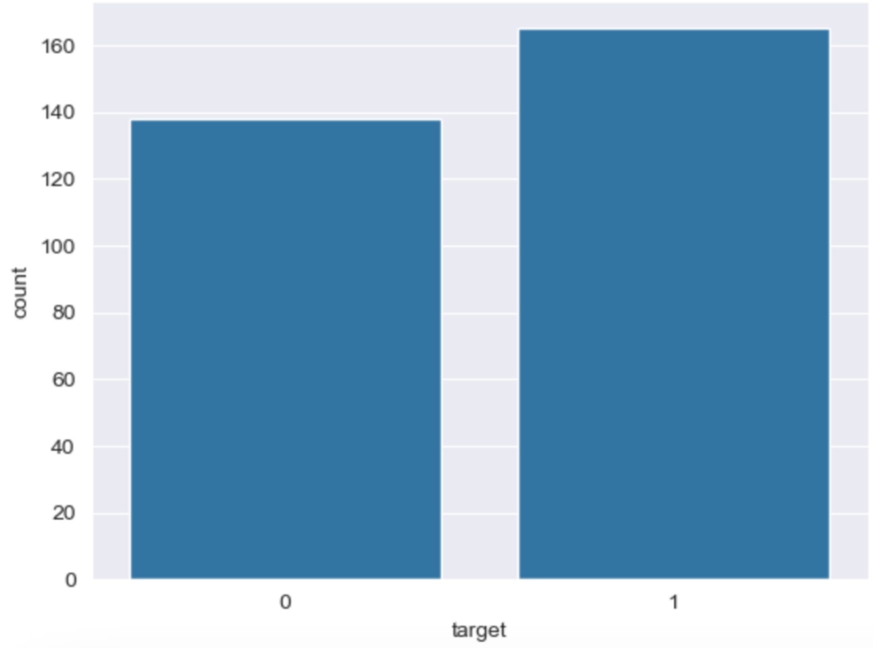
```
target
1    165
0    138
Name: count, dtype: int64
```

There are two unique values in target column.

1 = Defective Heart and 0 = Healthy Heart

```
# Using seaborn
sns.barplot(df['target'].value_counts())
```

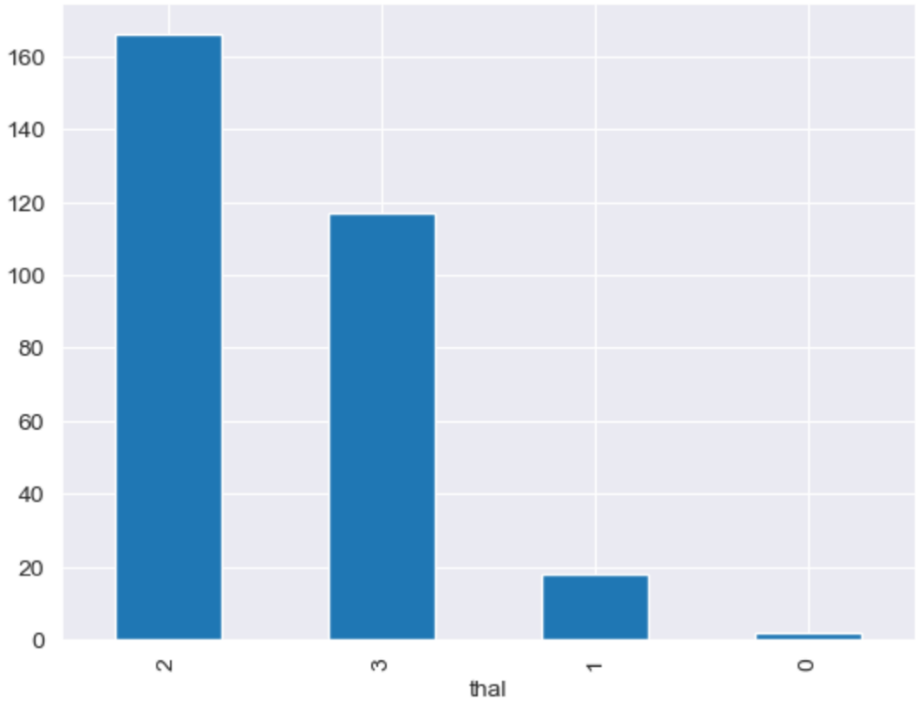
```
<Axes: xlabel='target', ylabel='count'>
```



From the above bar graph, we can conclude even when the distribution is not exactly 50:50, but still the data is good enough to use on machine learning algorithms and to predict standard metrics like Accuracy scores. So, we do not need to resample this dataset.

```
# For other columns
df['thal'].value_counts().plot(kind='bar')
```

<Axes: xlabel='thal'>



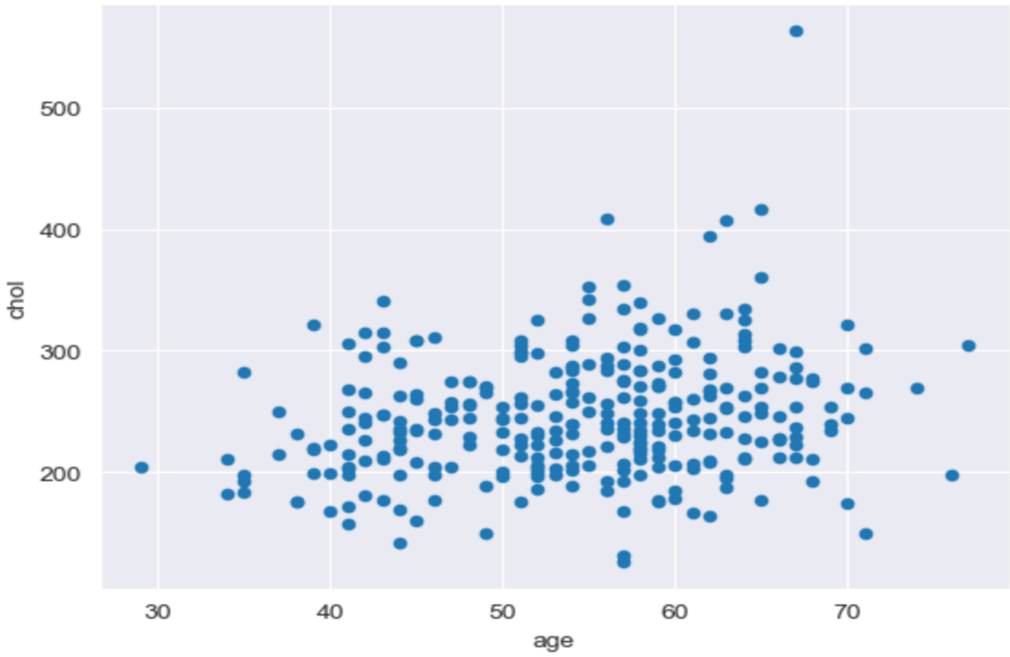
Bivariate Analysis

Bivariate analysis is the analysis of exactly two variables. We will use bivariate analysis to find relationships between two variables.

For bivariate analysis, we usually use boxplot(categorical vs numerical), scatterplot(numerical vs numerical).

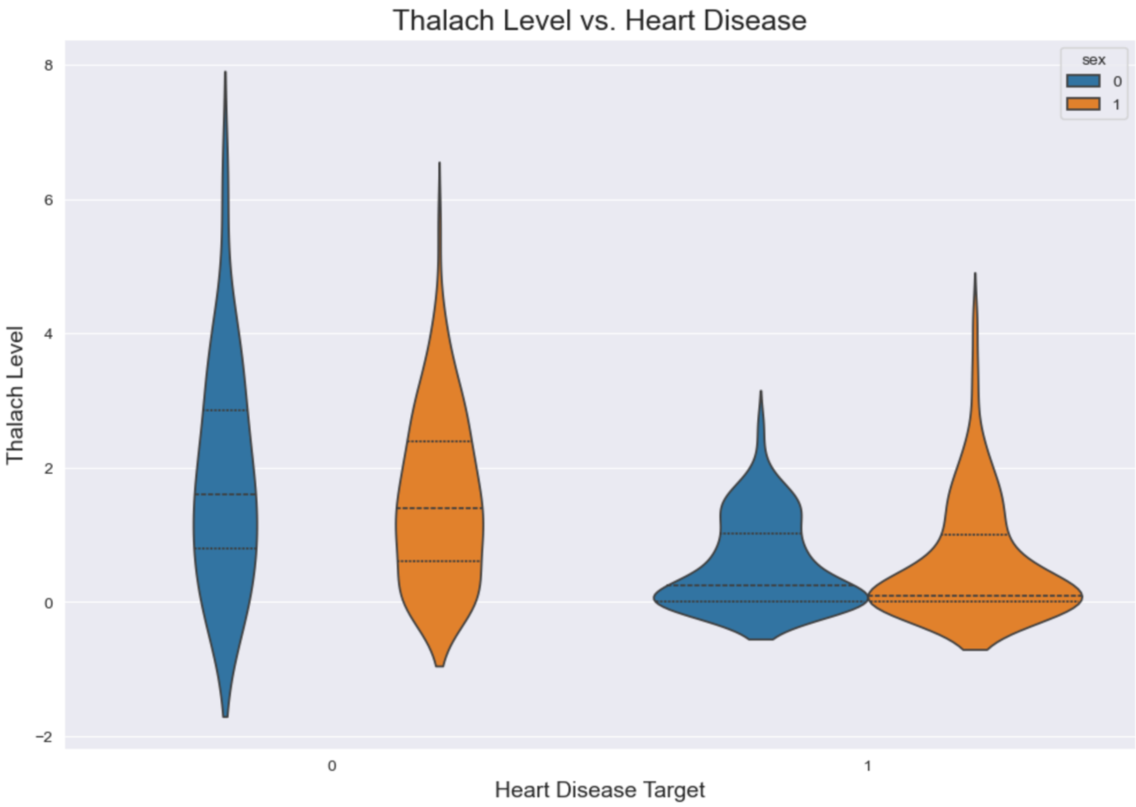
```python
# Scatter plot
df.plot(x='age', y='chol', kind='scatter')
```

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



The scatterplot shows us when age is increasing, cholesterol level is also increasing. That means cholesterol levels increase with age, increases the risk of heart attack or stroke.

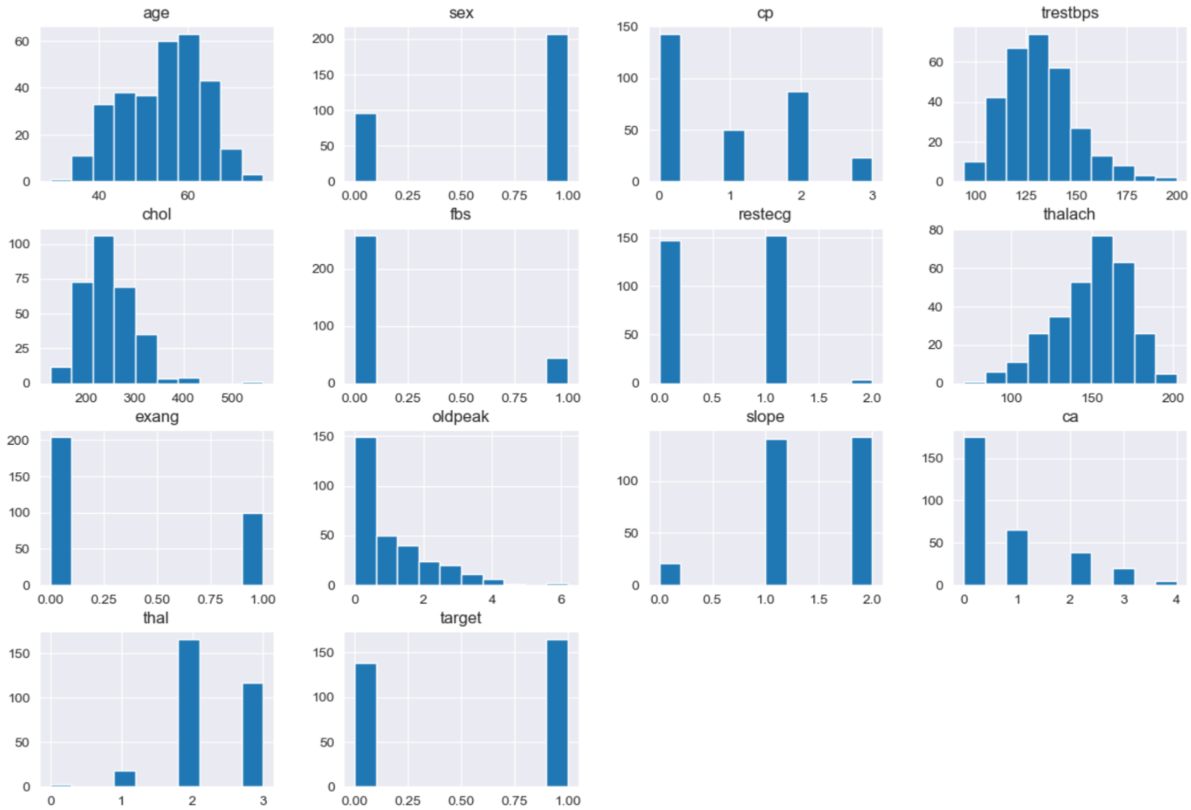
```
# Violin Plot
plt.figure(figsize=(12,8))
sns.violinplot(x= 'target', y= 'oldpeak', hue="sex", inner='quartile', data= df)
plt.title('Thalach Level vs. Heart Disease',size=18)
plt.xlabel('Heart Disease Target', size=14)
plt.ylabel('Thalach Level', size=14)
plt.show()
```



We can see that the overall shape & distribution for negative & positive patients differ vastly. Positive patients exhibit a lower median for ST depression level & thus a great distribution of their data is between 0 & 2, while negative patients are between 1 & 3. In addition, we don't see many differences between male & female target outcomes.

```
# Histogram

df.hist(figsize=(15,10))
plt.show()
```



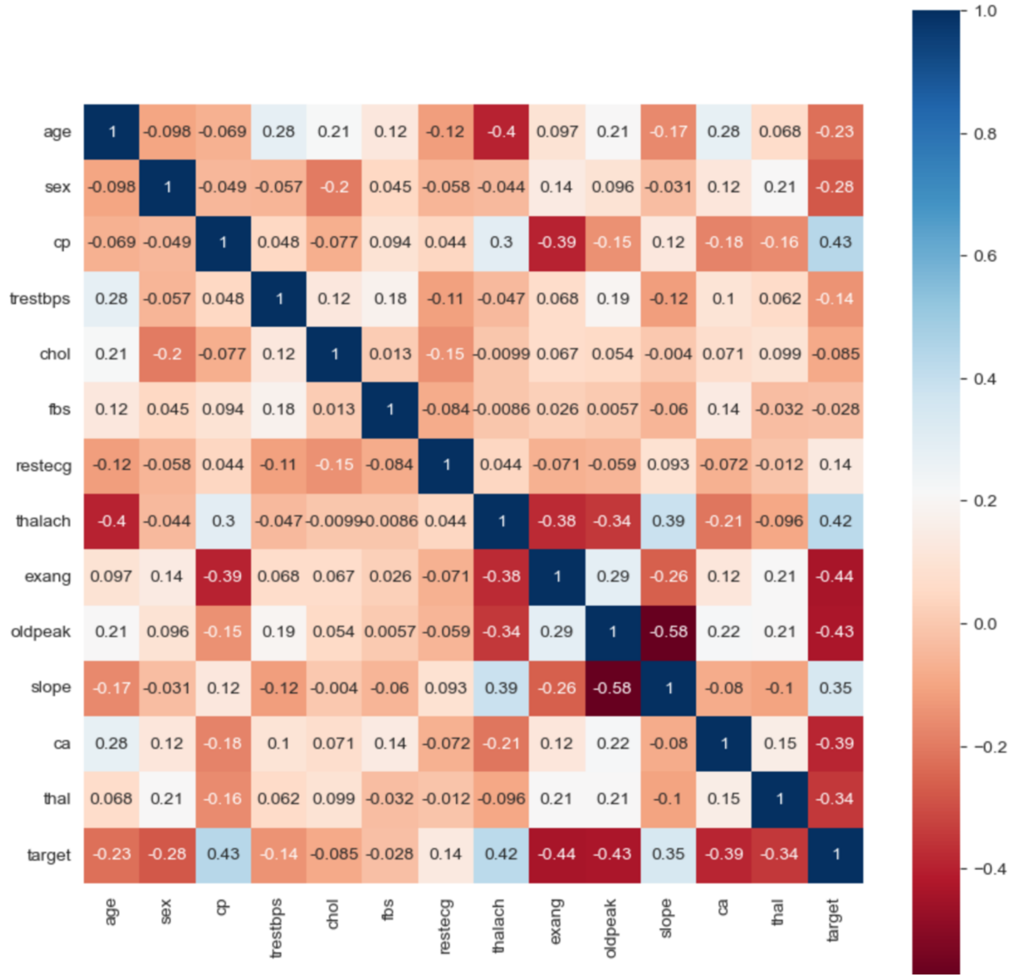
From the above histogram plots, we can see that the features are skewed and not normally distributed. Also, the scales are different between one and another.

Correlation

Correlation is used to test relationships between quantitative variables or categorical variables. It's a measure of how things are related. The heatmap() method shows us the relationship between numeric variables.

We will combine the .corr() method with heatmap so that we will be able to see the relationship in the graph.

```python
# Co-relation between variables
plt.figure(figsize = (10,10))
sns.heatmap(df.corr(), annot=True, square=True, cmap='RdBu')
plt.show()
```

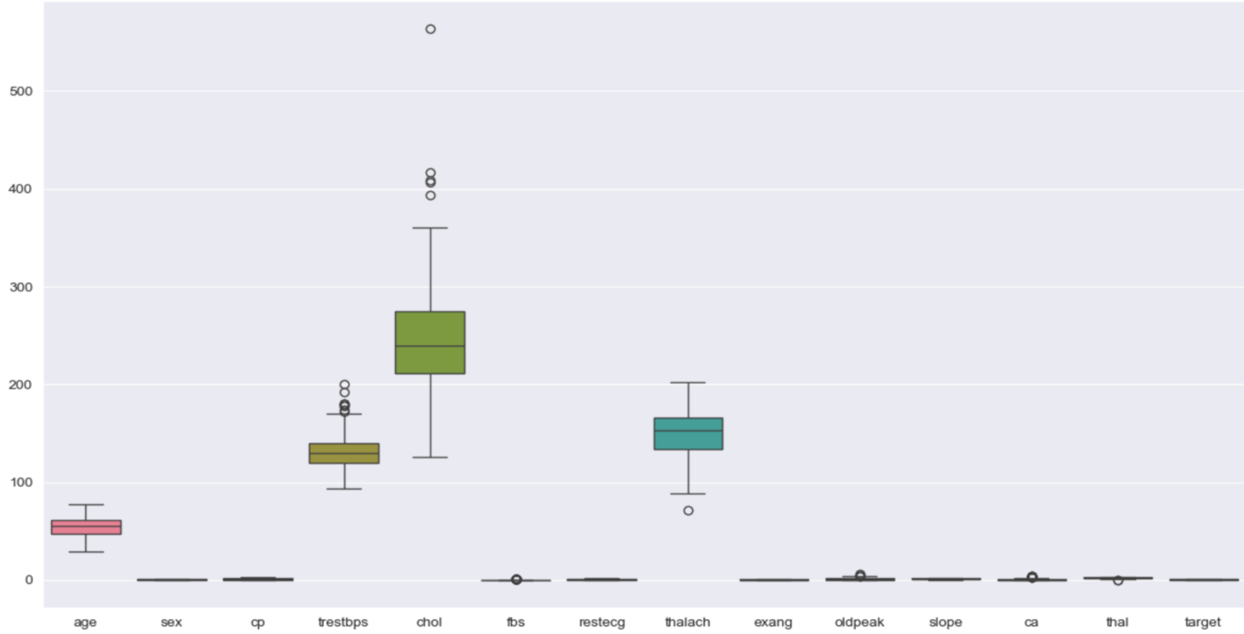


From the above Heatmap, we can see that cp and thalach are the features with highest positive correlation whereas exang, oldpeak ,ca and thal are negatively correlated. While other features do not hold much correlation with the response variable "target".

Outlier Detection

```
# Boxplot

plt.figure(figsize = (15,8))
sns.set_style('darkgrid')
sns.boxplot(df)
plt.show()
```



Train – Test Split

Now we will distribute the data into training and testing datasets using the train_test_split() function.

```
# Unique values in the target variable
df['target'].value_counts()
```

```
target
1    165
0    138
Name: count, dtype: int64
```

```
from sklearn.model_selection import train_test_split

X = df.drop(['target'], axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(212, 13) (91, 13) (212,) (91,)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
# Evaluation on Train and Test Data
from sklearn.metrics import accuracy_score

train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

train_accuracy = accuracy_score(y_train, train_pred)
test_accuracy = accuracy_score(y_test, test_pred)

print('Train Accuracy:', train_accuracy)
print('Test Accuracy:', test_accuracy)
```

```
Train Accuracy: 0.8679245283018868
Test Accuracy: 0.8131868131868132
```

Conclusion

Thus, from accuracy score we conclude a good outcome as 81% is the ideal accuracy.