

▼ Heart Disease Prediction using Machine Learning Approach

Heart Disease (including Coronary Heart Disease, Hypertension, and Stroke) remains the No. 1 cause of death in the US. The Heart Disease and Stroke Statistics—2019 Update from the American Heart Association indicates that:

- 116.4 million, or 46% of US adults are estimated to have hypertension. These are findings related to the new 2017 Hypertension Clinical Practice Guidelines.
- On average, someone dies of CVD every 38 seconds. About 2,303 deaths from CVD each day, based on 2016 data.
- On average, someone dies of a stroke every 3.70 minutes. About 389.4 deaths from stroke each day, based on 2016 data.

In this machine learning project, we have collected the dataset from UCI

([https://archive.ics.uci.edu/ml/datasets/statlog+\(heart\)](https://archive.ics.uci.edu/ml/datasets/statlog+(heart))) and we will be using Machine Learning to make predictions on whether a person is suffering from Heart Disease or not.

Problem Statement

- Complete analysis of Heart Disease UCI dataset.
- To predict whether a person has a heart disease or not based on the various biological and physical parameters.

Machine Learning Algorithms

- Random Forest Classifier
- K-Nearest Neighbors Classifier
- Decision Tree Classifier
- Naive Bayes Classifier

▼ Import libraries

Let's first import all the necessary libraries. We will use `numpy` and `pandas` to start with. For visualization, we will use `pyplot` subpackage of `matplotlib`, use `rcParams` to add styling to the plots and `rainbow` for colors and `seaborn`. For implementing Machine Learning models and processing of data, we will use the `sklearn` library.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
import seaborn as sns
%matplotlib inline
```

For processing the data, we'll import a few libraries. To split the available dataset for testing and training, we'll use the `train_test_split` method. To scale the features, we are using `StandardScaler`.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
from warnings import filterwarnings
filterwarnings("ignore")
```

For model validation, we'll import a few libraries.

```
#model validation
from sklearn.metrics import log_loss,roc_auc_score,precision_score,f1_score,recall_score,roc_curve,a
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,fbeta_score,matth
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix

#extra
from sklearn.pipeline import make_pipeline, make_union
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import SelectFwe, f_regression
```

Next, we will import all the Machine Learning algorithms

- K-Nearest Neighbors Classifier
- Random Forest Classifier
- Decision Tree Classifier
- Naive Bayes Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
```

▼ Import dataset

Now that we have all the libraries we will need, we can import the dataset and take a look at it. The dataset is stored in the file `dataset.csv`. We'll use the `pandas read_csv` method to read the dataset.

```
dataset = pd.read_csv('dataset.csv',sep=',',encoding="utf-8")
```

▼ Data Preparation and Data Exploration

```
type(dataset)
```

```
pandas.core.frame.DataFrame
```

```
dataset.shape
```

```
(270, 14)
```

The dataset is now loaded into the variable `dataset`. We'll just take a glimpse of the data using the `describe()` and `info()` methods before we actually start processing and visualizing it.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   age         270 non-null   int64  
 1   sex         270 non-null   int64  
 2   cp          270 non-null   int64  
 3   trestbps    270 non-null   int64  
 4   chol        270 non-null   int64  
 5   fbs         270 non-null   int64  
 6   restecg     270 non-null   int64  
 7   thalach     270 non-null   int64  
 8   exang       270 non-null   int64  
 9   oldpeak     270 non-null   float64 
10   slope       270 non-null   int64  
11   ca          270 non-null   int64  
12   thal        270 non-null   int64  
13   target      270 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 29.7 KB
```

Looks like the dataset has a total of 270 rows and there are no missing values. There are a total of 13 features along with one target value which we wish to find.

```
dataset.columns
```

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

```
dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thal
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000
mean	54.433333	0.677778	3.174074	131.344444	249.659259	0.148148	1.022222	149.677778

The scale of each feature column is different and quite varied as well. While the maximum for age reaches 77, the maximum of chol (serum cholestoral) is 564.

dataset

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	0
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	1
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	0
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	0
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	0
...
265	52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
266	44	1	2	120	263	0	0	173	0	0.0	1	0	7	0
267	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
268	57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
269	67	1	4	160	286	0	2	108	1	1.5	2	3	3	0

270 rows × 14 columns

dataset.head()

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	0
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	1
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	0
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	0
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	0

dataset.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
```

So, we have no missing values

```
dataset.apply(lambda x:len(x.unique()))
```

```
age          41
sex          2
cp           4
trestbps     47
chol        144
fbs          2
restecg      3
thalach      90
exang        2
oldpeak      39
slope        3
ca           4
thal         3
target       2
dtype: int64
```

```
print('cp ',dataset['cp'].unique())
print('fbs ',dataset['fbs'].unique())
print('restecg ',dataset['restecg'].unique())
print('exang ',dataset['exang'].unique())
print('slope ',dataset['slope'].unique())
print('ca ',dataset['ca'].unique())
print('thal ',dataset['thal'].unique())
```

```
cp  [4 3 2 1]
fbs  [0 1]
restecg  [2 0 1]
exang  [0 1]
slope  [2 1 3]
ca  [3 0 1 2]
thal  [3 7 6]
```

Dataset Description:

This dataset consists of 13 features and a target variable. The detailed description of all the features are as follows:

1. **Age:** Patients Age in years (Numeric)

2. **Sex:** Gender of patient (Male - 1, Female - 0)(Nominal)

3. **Chest Pain Type:** Type of chest pain experienced by patient categorized into :(Nominal)

- Value 1: Typical angina
- Value 2: Atypical angina
- Value 3: Non-anginal pain
- Value 4: Asymptomatic

(Angina: Angina is caused when there is not enough oxygen-rich blood flowing to a certain part of the heart. The arteries of the heart become narrow due to fatty deposits in the artery walls. The narrowing of arteries means that blood supply to the heart is reduced, causing angina.)

4. **resting bps:** Level of blood pressure at resting mode in mm/HG (Numerical)

5. **cholesterol:** Serum cholesterol in mg/dl (Numeric) (Cholesterol means the blockage for blood supply in the blood vessels)

6. **fasting blood sugar:** Blood sugar levels on fasting > 120 mg/dl represents as 1 in case of true and 0 as false (Nominal) (blood sugar taken after a long gap between a meal and the test. Typically, it's taken before any meal in the morning.)

7. **resting ecg:** Result of electrocardiogram while at rest are represented in 3 distinct values: (Nominal)

- Value 0: Normal
- Value 1: Having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
- Value 2: Showing probable or definite left ventricular hypertrophy by Estes' criteria.

(ECG values taken while person is on rest which means no exercise and normal functioning of heart is happening)

8. **oldpeak:** Exercise induced ST-depression in comparison with the state of rest (Numeric)

(ST Depression is the difference between value of ECG at rest and after exercise. An electrocardiogram records the electrical signals in your heart. It's a common and painless test used to quickly detect heart problems and monitor your heart's health. Electrocardiograms — also called ECGs or EKGs — are often done in a doctor's office, a clinic or a hospital room. ECG machines are standard equipment in operating rooms and ambulances. Some personal devices, such as smart watches.)

9. **ST slope:** ST segment measured in terms of slope during peak exercise (Nominal)

- Value 1: Upsloping
 - Value 2: Flat
 - Value 3: Downsloping
-

10. **ca:** Number of major blood vessels (0-3)(Numeric)

(Fluoroscopy is an imaging technique that uses X-rays to obtain real-time moving images of the interior of an object. In its primary application of medical imaging, a fluoroscope allows a physician to see the internal structure and function of a patient, so that the pumping action of the heart or the motion of swallowing, for example, can be watched)

11. **exang**: Exercise induced angina (1 = yes; 0 = no)

(is chest pain while exercising or doing any physical activity.)

12. **thal**: Thallium stress test

- Value 3: normal
 - Value 6: fixed defect
 - Value 7: reversibe defect
-

13. **thalach**: Maximum heart rate achieved in bpm(Numeric)

Target variable

14. **target**: It is the target variable which we have to predict 1 means patient is suffering from heart risk and 0 means patient is normal. (1 = no disease; 2 = disease)

▼ Data Visualization

Now let's see various visual representations of the data to understand more about relationship between various features.

▼ Distribution of Heart disease (target variable)

It's always a good practice to work with a dataset where the target classes are of approximately equal size. Thus, let's check for the same.

```
fig, (ax1) = plt.subplots(nrows=1, ncols=1, sharey=False, figsize=(14,6))

ax1 = dataset['target'].value_counts().plot.pie( x="Heart disease" ,y ='no.of patients',
        autopct = "%1.0f%%",labels=["Heart Disease","Normal"], startangle = 60,ax=ax1);
ax1.set(title = 'Percentage of Heart disease patients in Dataset')
plt.show()
```

The two classes are not exactly 50% each but the ratio is good enough to continue without dropping/increasing our data.

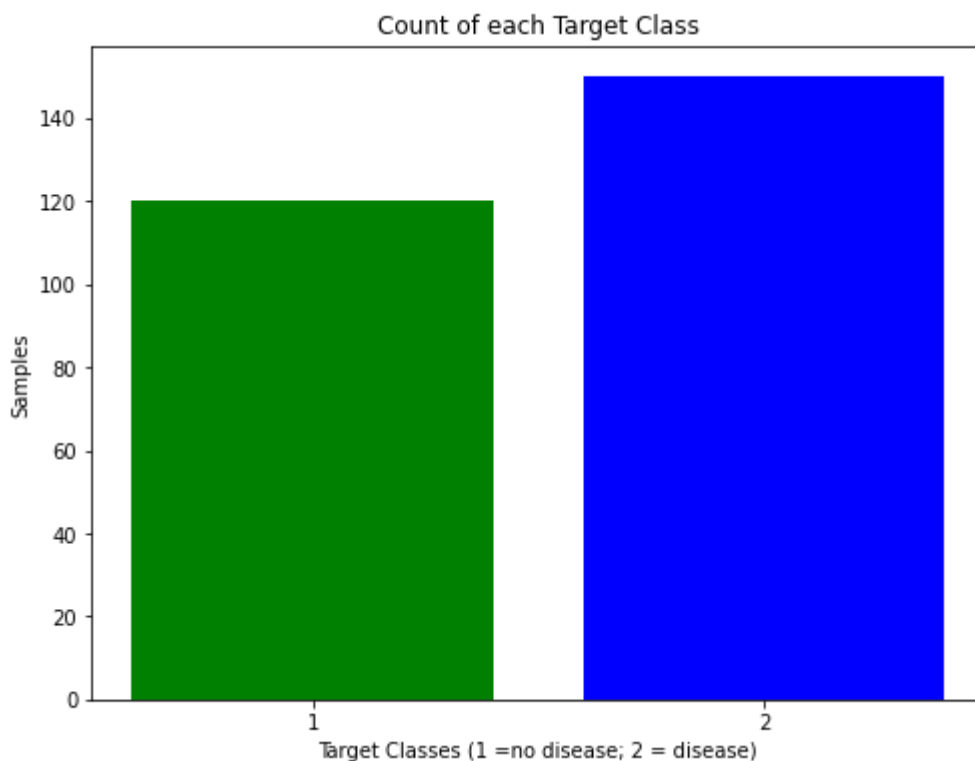
Heart Disease 

```
y = dataset["target"]
```



```
rcParams['figure.figsize'] = 8,6
plt.bar(dataset['target'].unique(), dataset['target'].value_counts(), color = ['blue', 'green'])
plt.xticks([1, 2])
plt.xlabel('Target Classes (1 =no disease; 2 = disease)')
plt.ylabel('Samples')
plt.title('Count of each Target Class')
target_temp = dataset.target.value_counts()
print(target_temp)
```

```
1    150
2    120
Name: target, dtype: int64
```



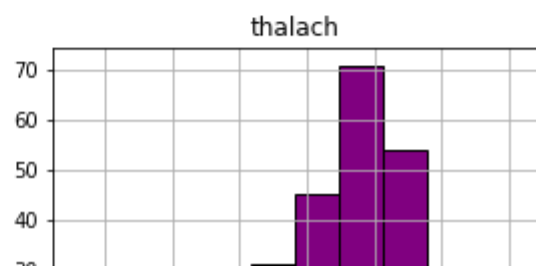
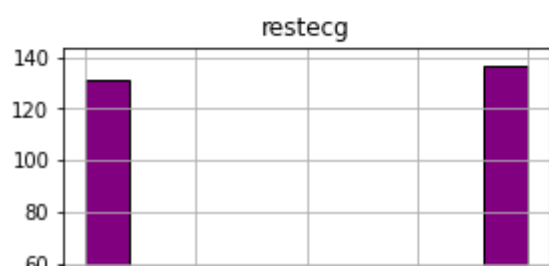
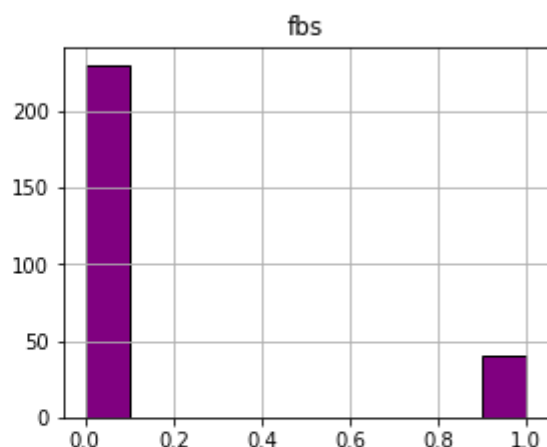
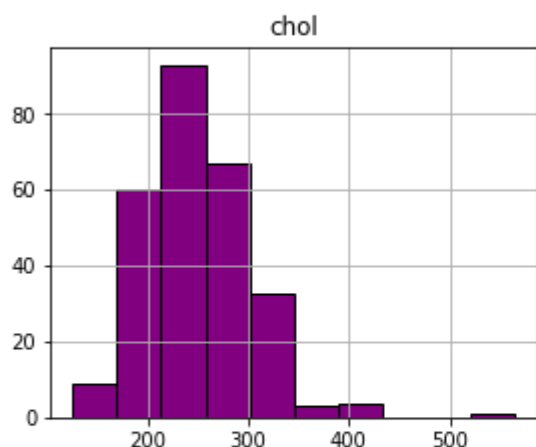
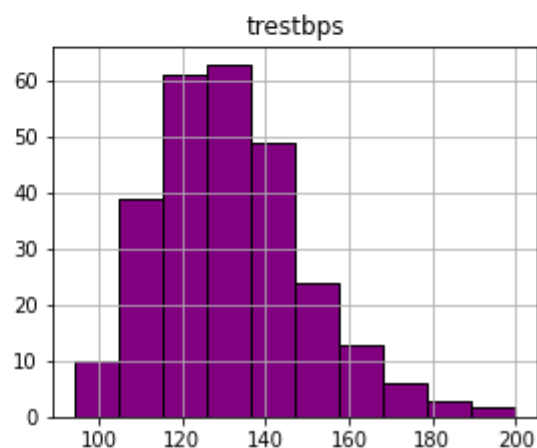
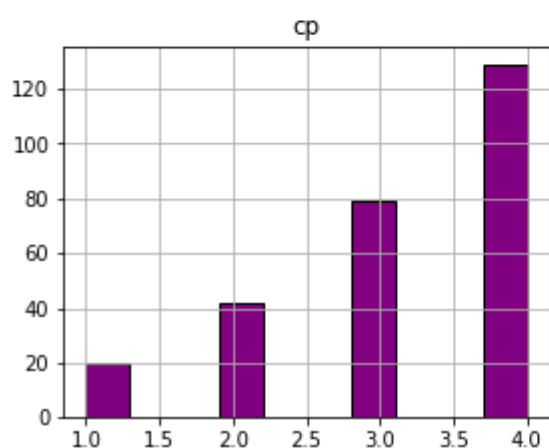
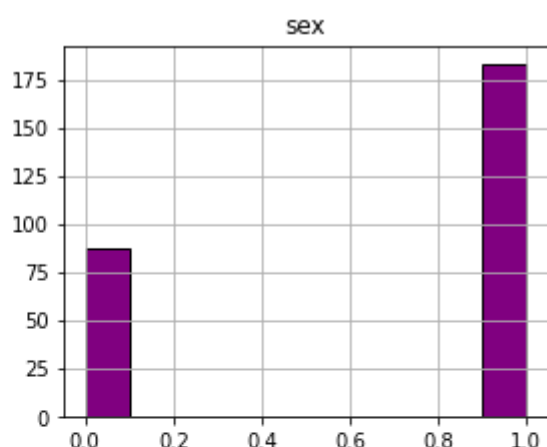
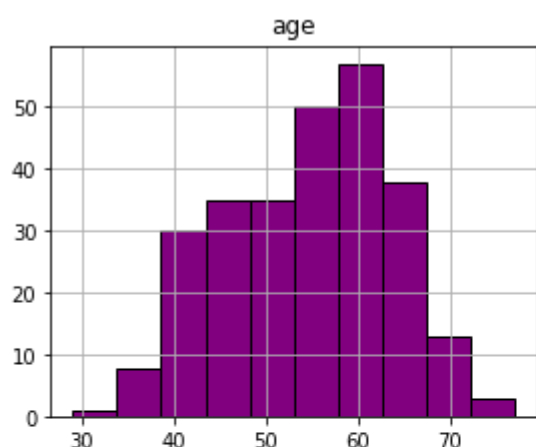
From the total dataset of 270 patients, 150 (56%) have a heart disease (target=2)

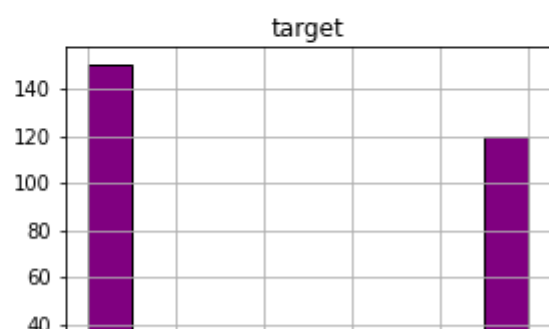
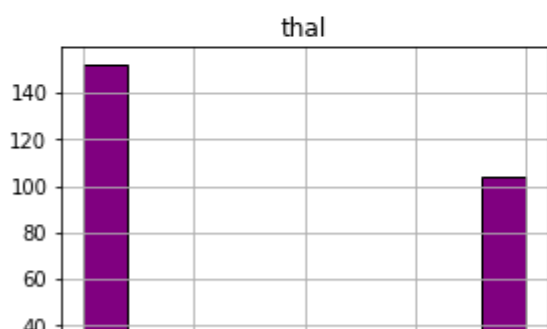
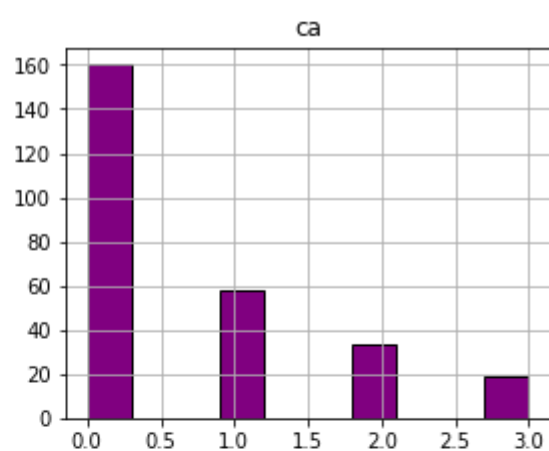
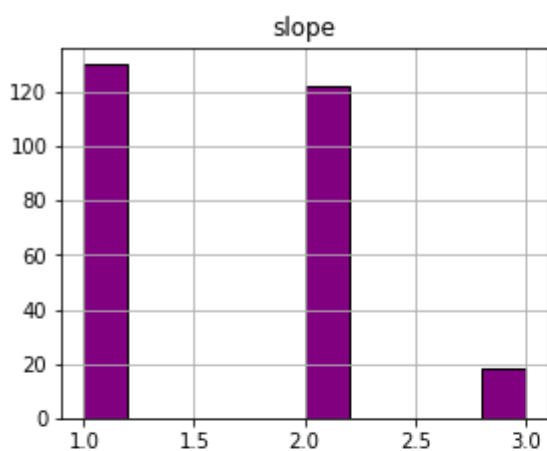
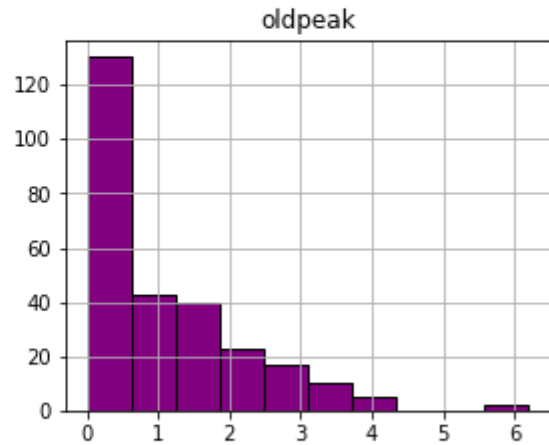
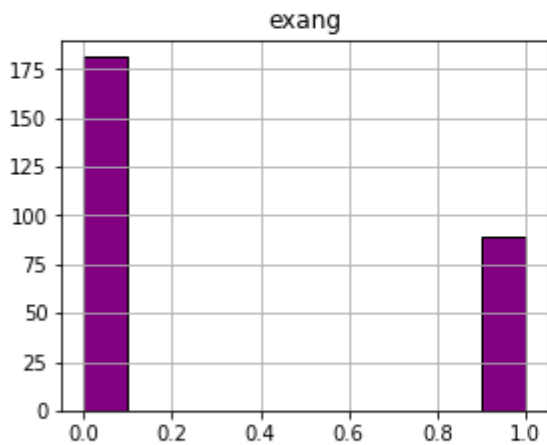
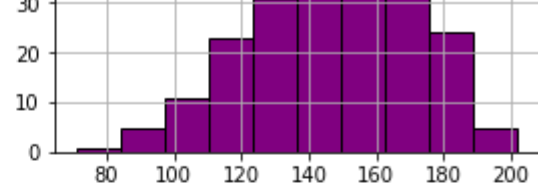
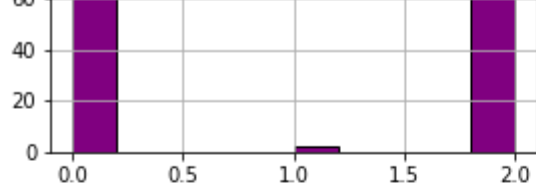
Next, we'll take a look at the histograms for each variable.

```
dataset.hist(edgecolor='black',layout = (7, 2),
             figsize = (10, 30),
             color=['purple'])
```



```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5c3bfd0>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5c194d0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5bce910>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5b82d90>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5b46290>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5afd790>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5ab2d10>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5a74190>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5a741d0>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5a2a7d0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a59a3110>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a59d9610>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x7f80a598eb10>,
      <matplotlib.axes._subplots.AxesSubplot object at 0x7f80a5944fd0>]],
      dtype=object)
```





Taking a look at the histograms above, I can see that each feature has a different range of distribution. Thus, using scaling before our predictions should be of great use. Also, the categorical features do stand out.

- ▼ **Exploratory Data Analysis (EDA)**
- ▼ **Gender distribution based on heart disease**

```
dataset["sex"].unique()

array([1, 0])

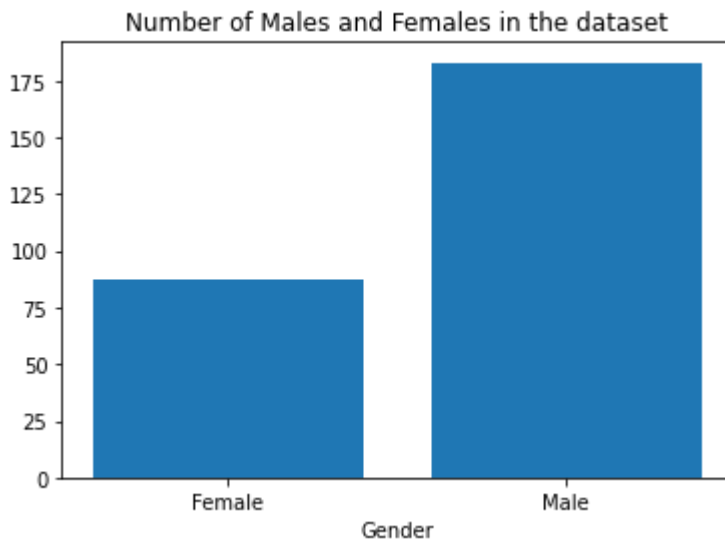
# Number of males and females
```

```

F = dataset[dataset["sex"] == 0].count()["target"]
M = dataset[dataset["sex"] == 1].count()["target"]

# Create a plot
figure, ax = plt.subplots(figsize = (6, 4))
ax.bar(x = ['Female', 'Male'], height = [F, M])
plt.xlabel('Gender')
plt.title('Number of Males and Females in the dataset')
plt.show()

```

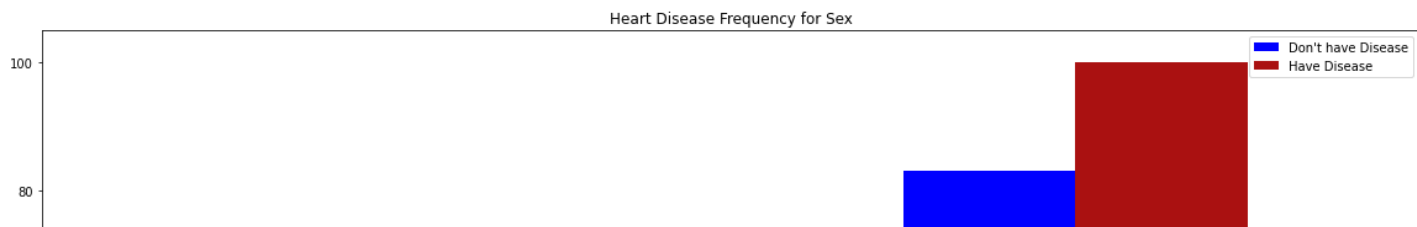


Heart Disease frequency for gender

```

pd.crosstab(dataset.sex, dataset.target).plot(kind="bar", figsize=(20,10), color=['blue', '#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Don't have Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()

```



```
countFemale = len(dataset[dataset.sex == 0])
countMale = len(dataset[dataset.sex == 1])
print("Percentage of Female Patients:{:.2f}%".format((countFemale)/(len(dataset.sex))*100))
print("Percentage of Male Patients:{:.2f}%".format((countMale)/(len(dataset.sex))*100))
```

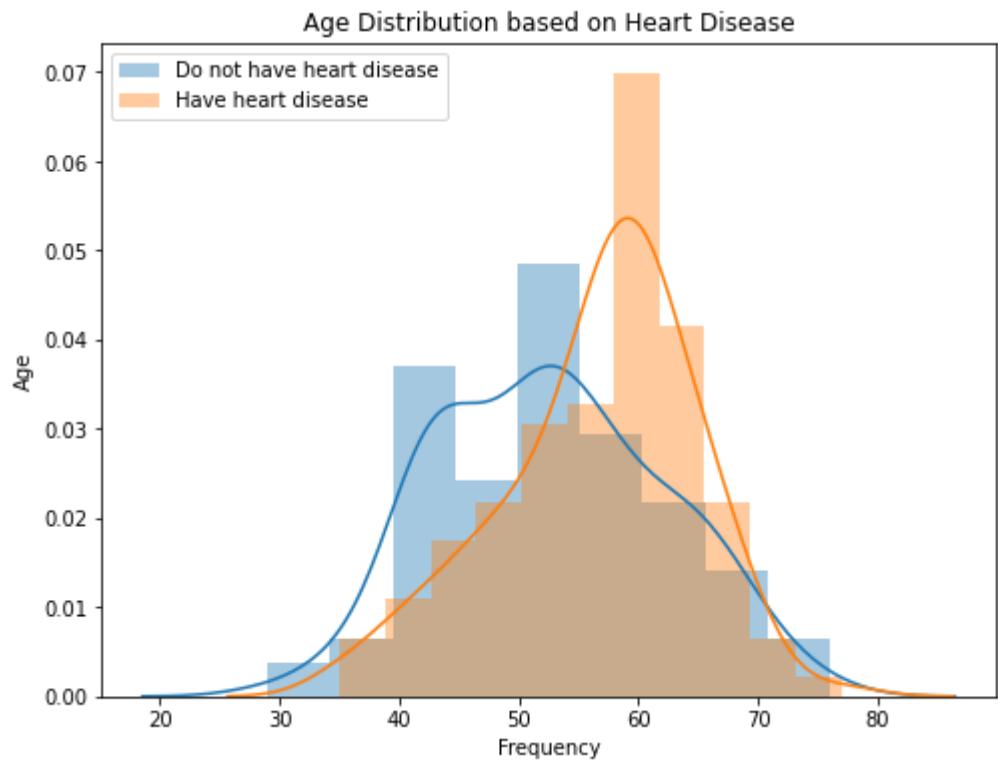
Percentage of Female Patients:32.22%
 Percentage of Male Patients:67.78%



▼ Age distribution based on heart disease



```
# Display age distribution based on heart disease
sns.distplot(dataset[dataset['target'] == 1]['age'], label='Do not have heart disease')
sns.distplot(dataset[dataset['target'] == 2]['age'], label = 'Have heart disease')
plt.xlabel('Frequency')
plt.ylabel('Age')
plt.title('Age Distribution based on Heart Disease')
plt.legend()
plt.show()
```



Get min, max and average of the age of the people do not have heart disease

```
print('Min age of people who do not have heart disease: ', min(dataset[dataset['target'] == 1]['age'])
print('Max age of people who do not have heart disease: ', max(dataset[dataset['target'] == 1]['age'])
print('Average age of people who do not have heart disease: ', dataset[dataset['target'] == 1]['age'].mean())
```

Min age of people who do not have heart disease: 29
 Max age of people who do not have heart disease: 76

Average age of people who do not have heart disease: 52.70666666666666

Get min, max and average of the age of the people have heart disease

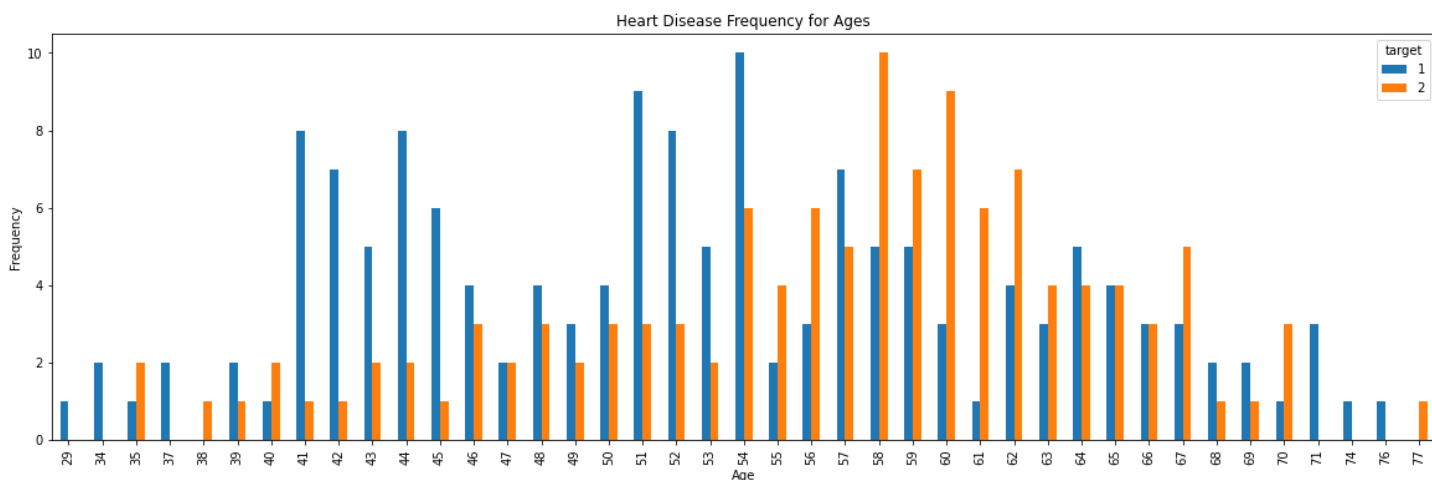
```
print('Min age of people who have heart disease: ', min(dataset[dataset['target'] == 2]['age']))
print('Max age of people who have heart disease: ', max(dataset[dataset['target'] == 2]['age']))
print('Average age of people who have heart disease: ', dataset[dataset['target'] == 2]['age'].mean())
```

Min age of people who have heart disease: 35
Max age of people who have heart disease: 77
Average age of people who have heart disease: 56.59166666666667

From the data, we can say that the heart disease infects the old and young people, and the probability of the old people to be infected is higher than young people.

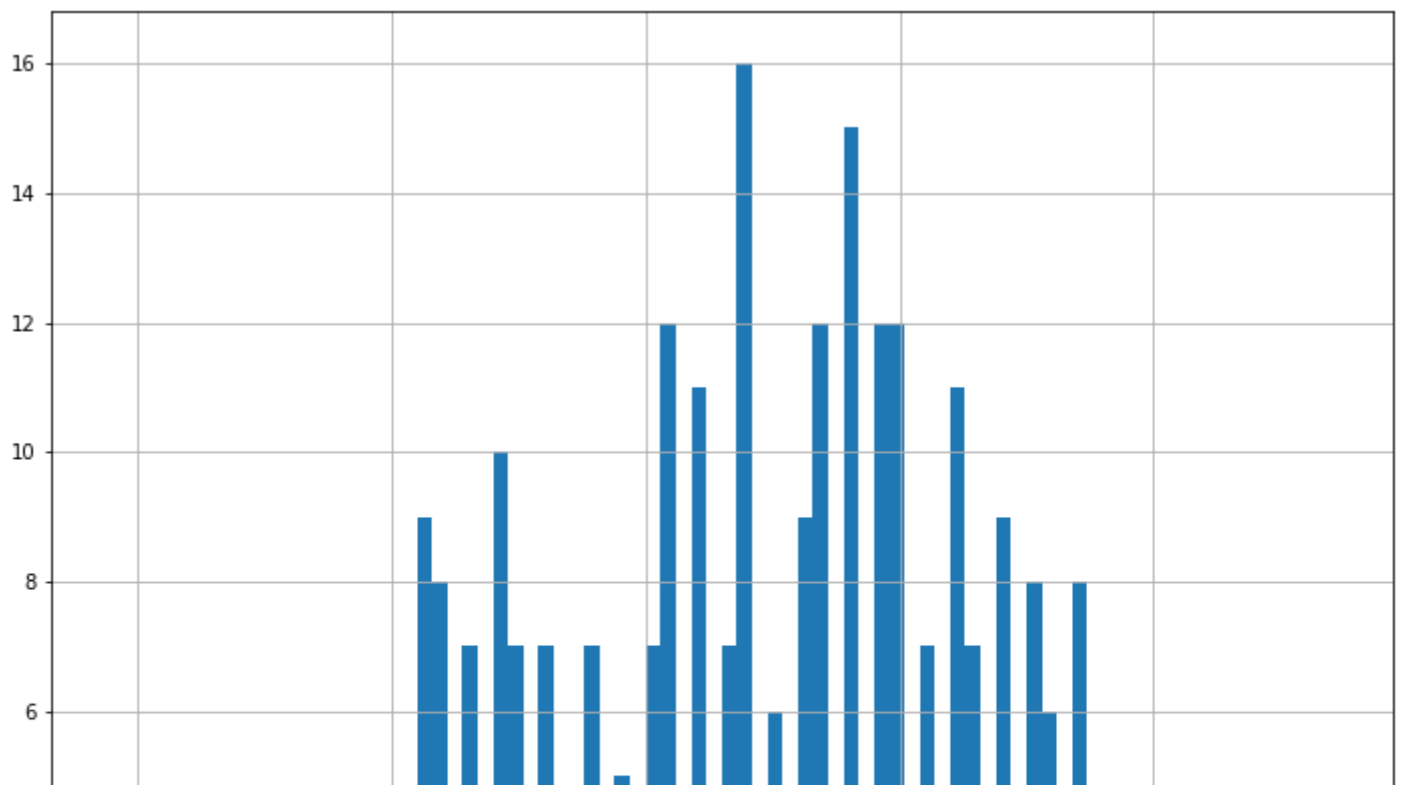
Heart Disease Frequency for ages

```
pd.crosstab(dataset.age, dataset.target).plot(kind="bar", figsize=(20, 6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



```
plt.figure(figsize=(12, 10))
dataset.age.hist(bins=80)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f80a54041d0>
```



```
print(f"The most of the patients have a mean age of : {dataset.age.mean()}")
```

The most of the patients have a mean age of : 54.43333333333333



▼ Distribution of Categorical features



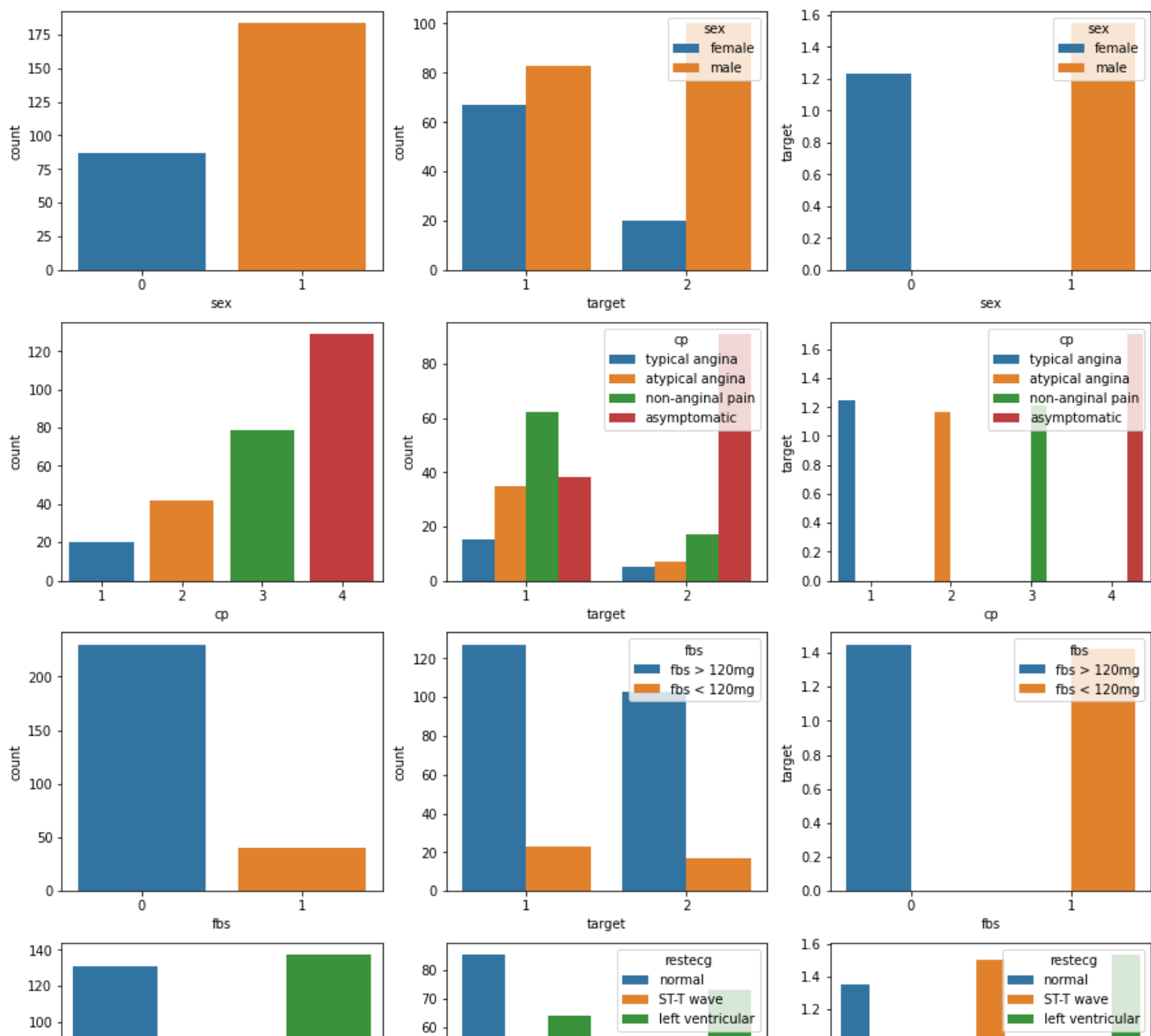
```
categorical = [('sex', ['female', 'male']),
               ('cp', ['typical angina', 'atypical angina', 'non-anginal pain', 'asymptomatic']),
               ('fbs', ['fbs > 120mg', 'fbs < 120mg']),
               ('restecg', ['normal', 'ST-T wave', 'left ventricular']),
               ('exang', ['yes', 'no']),
               ('slope', ['upsloping', 'flat', 'downsloping']),
               ('thal', ['normal', 'fixed defect', 'reversible defect'])]
```

```
def plotGrid(isCategorical):
    if isCategorical:
        [plotCategorical(x[0], x[1], i) for i, x in enumerate(categorical)]
    else:
        [plotContinuous(x[0], x[1], i) for i, x in enumerate(continuous)]
```

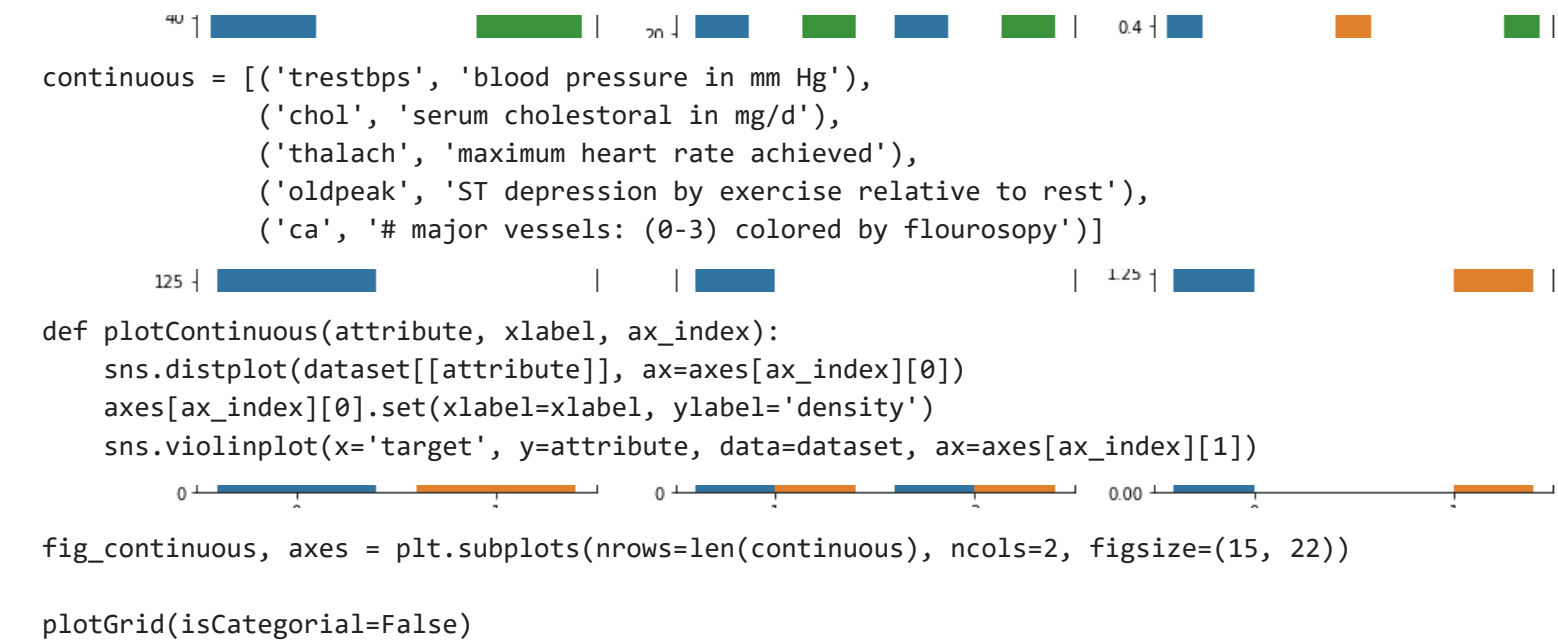
```
def plotCategorical(attribute, labels, ax_index):
    sns.countplot(x=attribute, data=dataset, ax=axes[ax_index][0])
    sns.countplot(x='target', hue=attribute, data=dataset, ax=axes[ax_index][1])
    avg = dataset[[attribute, 'target']].groupby([attribute], as_index=False).mean()
    sns.barplot(x=attribute, y='target', hue=attribute, data=avg, ax=axes[ax_index][2])

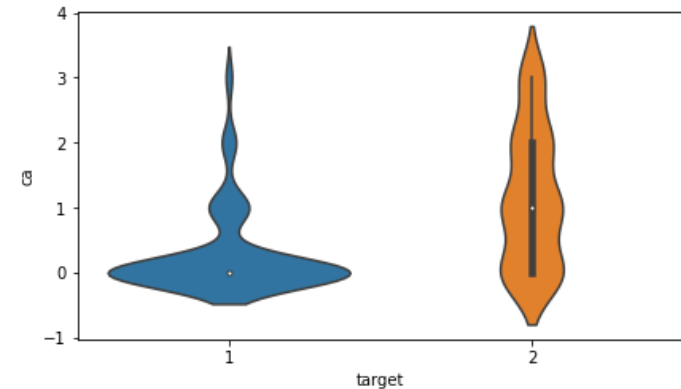
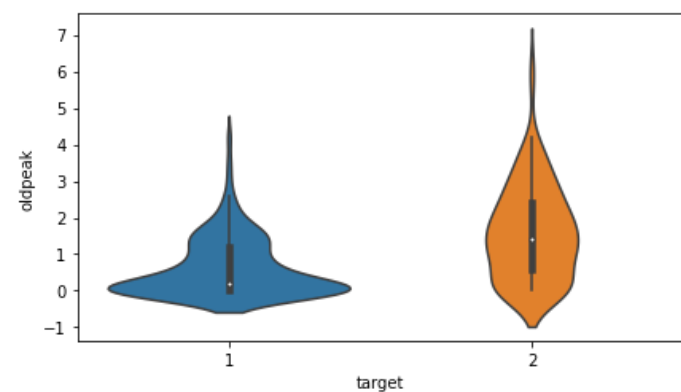
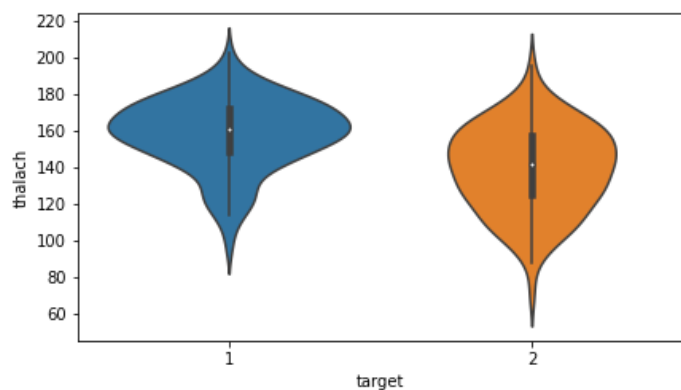
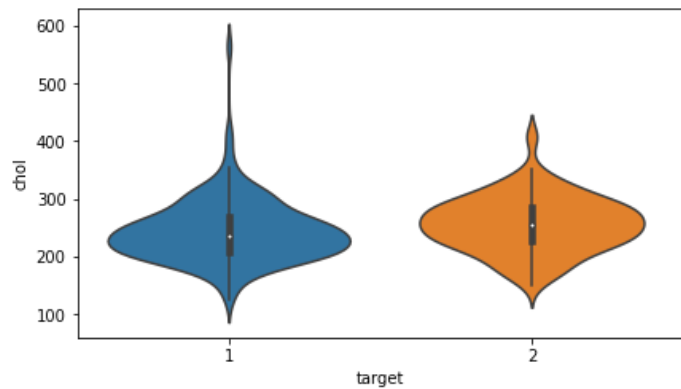
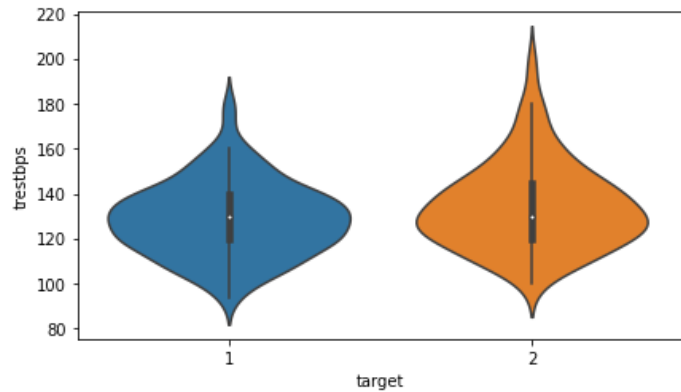
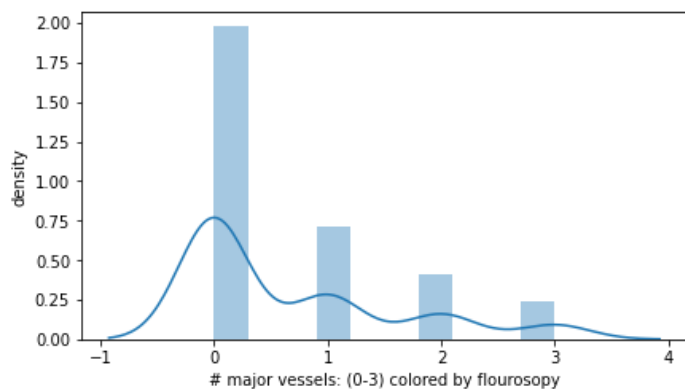
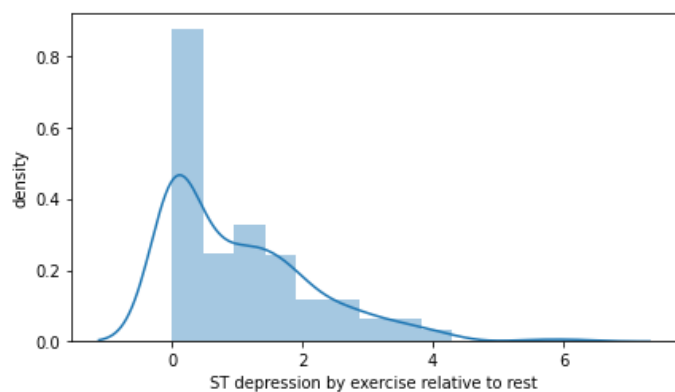
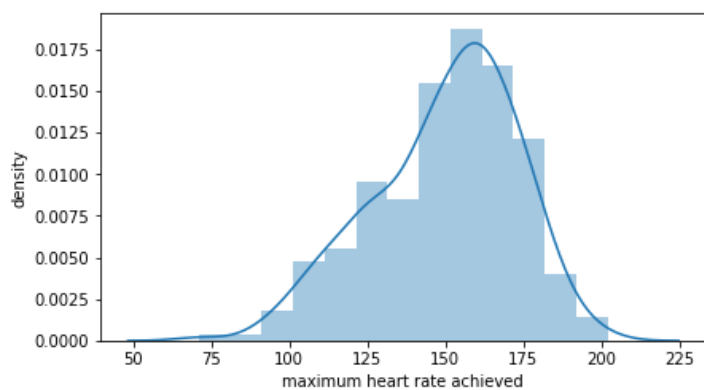
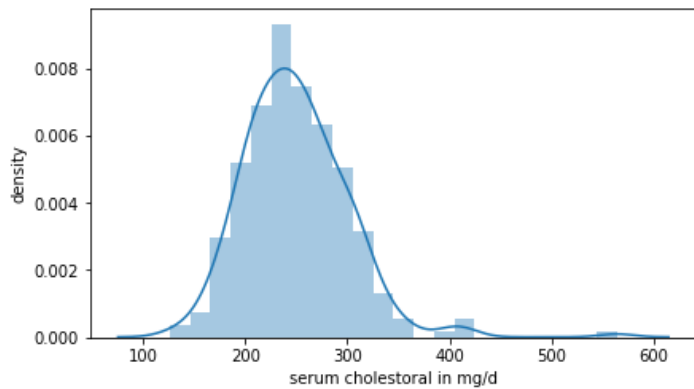
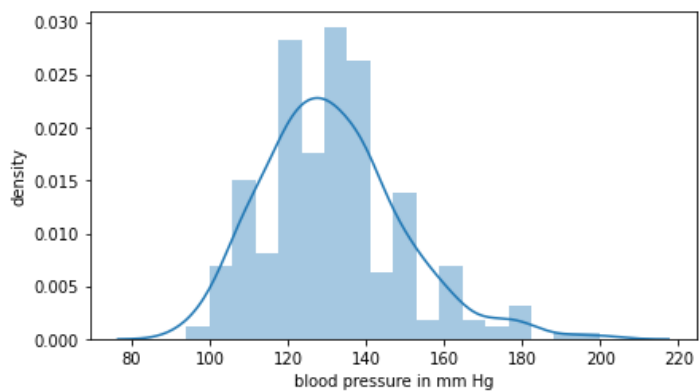
    for t, l in zip(axes[ax_index][1].get_legend().texts, labels):
        t.set_text(l)
    for t, l in zip(axes[ax_index][2].get_legend().texts, labels):
        t.set_text(l)
```

```
fig_categorical, axes = plt.subplots(nrows=len(categorical), ncols=3, figsize=(15, 30))  
  
plotGrid(True)
```



▼ Distribution of Continuous features





▼ PiePlots

```
fig, ax = plt.subplots(4,2, figsize = (14,14))
((ax1, ax2), (ax3, ax4), (ax5, ax6), (ax7, ax8)) = ax

labels = ["Male", "Female"]
values = dataset['sex'].value_counts().tolist()[2:]
ax1.pie(x=values, labels=labels, autopct="%1.1f%%",colors=['#AAb3ff','#CC80FF'],shadow=True, startangle=45)
ax1.set_title("Sex", fontdict={'fontsize': 12},fontweight = 'bold')

labels = ["Typical angina", "Atypical angina","non-anginal pain","asymptomatic"]
values = dataset['cp'].value_counts().tolist()
ax2.pie(x=values, labels=labels, autopct="%1.1f%%",colors=['#AAb3ff','#CC80FF','#DD00AA','#FF0099'],shadow=True, startangle=45)
ax2.set_title("Chest Pain", fontdict={'fontsize': 12},fontweight = 'bold')

labels = dataset['fbs'].value_counts().index.tolist()[2:]
values = dataset['fbs'].value_counts().tolist()
ax3.pie(x=values, labels=labels, autopct="%1.1f%%",colors=['#AAb3ff','#CC80FF'],shadow=True, startangle=45)
ax3.set_title("Fasting Blood Sugar", fontdict={'fontsize': 12},fontweight = 'bold')

labels = dataset['restecg'].value_counts().index.tolist()[3:]
values = dataset['restecg'].value_counts().tolist()
ax4.pie(x=values, labels=labels, autopct="%1.1f%%", colors=['#AAb3ff','#CC80FF','#DD00AA'],shadow=True, startangle=45)
ax4.set_title("Resting Blood Pressure", fontdict={'fontsize': 12},fontweight = 'bold')

labels = dataset['exang'].value_counts().index.tolist()[2:]
values = dataset['exang'].value_counts().tolist()
ax5.pie(x=values, labels=labels, autopct="%1.1f%%", colors=['#AAb3ff','#CC80FF'],shadow=True, startangle=45)
ax5.set_title("Exercise induced Angina", fontdict={'fontsize': 12},fontweight = 'bold')

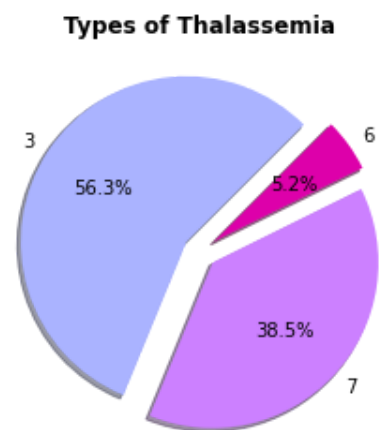
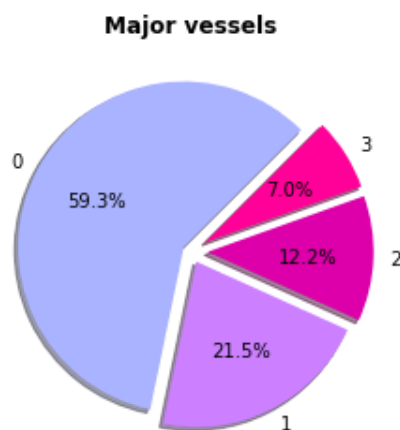
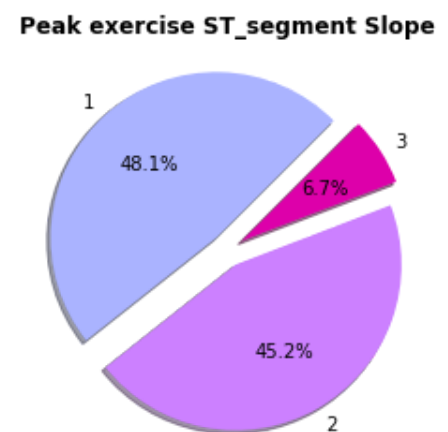
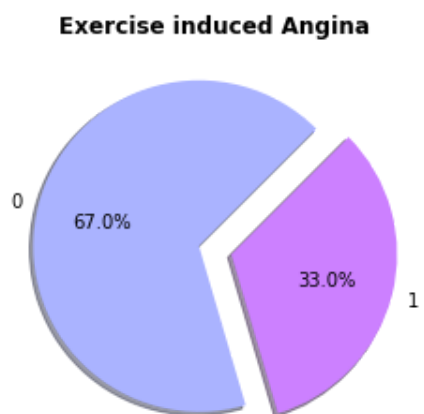
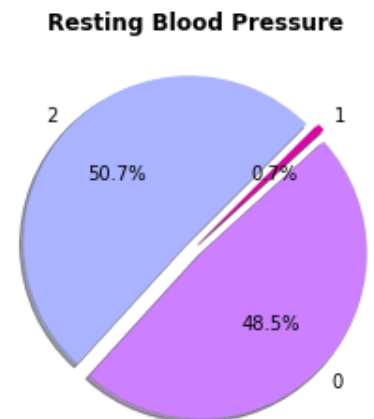
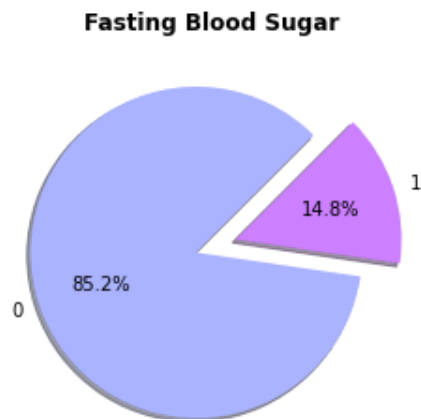
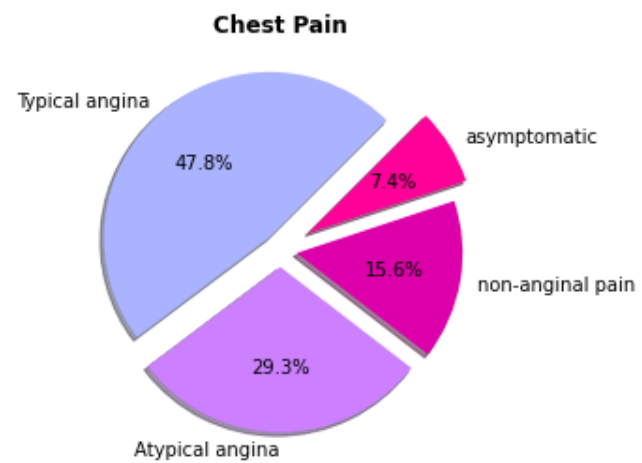
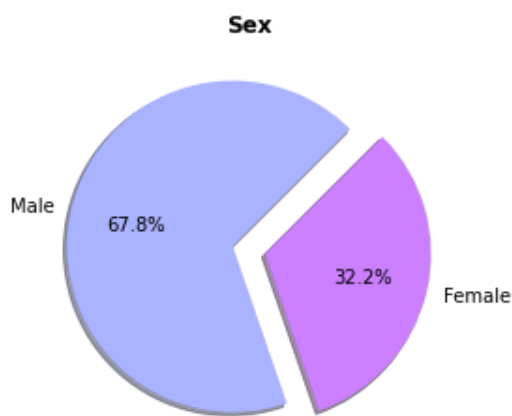
labels = dataset['slope'].value_counts().index.tolist()[3:]
values = dataset['slope'].value_counts().tolist()
ax6.pie(x=values, labels=labels, autopct="%1.1f%%", colors=['#AAb3ff','#CC80FF','#DD00AA'],shadow=True, startangle=45)
ax6.set_title("Peak exercise ST_segment Slope", fontdict={'fontsize': 12},fontweight = 'bold')

labels = dataset['ca'].value_counts().index.tolist()[4:]
values = dataset['ca'].value_counts().tolist()
ax7.pie(x=values, labels=labels, autopct="%1.1f%%", shadow=True, startangle=45,explode=[0.05, 0.07, 0.05, 0.05])
ax7.set_title("Major vessels", fontdict={'fontsize': 12},fontweight = 'bold')

labels = dataset['thal'].value_counts().index.tolist()[3:]
values = dataset['thal'].value_counts().tolist()
ax8.pie(x=values, labels=labels, autopct="%1.1f%%", shadow=True, startangle=45,explode=[0.1, 0.1, 0.1, 0.1])
ax8.set_title("Types of Thalassemia", fontdict={'fontsize': 12},fontweight = 'bold')

plt.tight_layout()
plt.show()

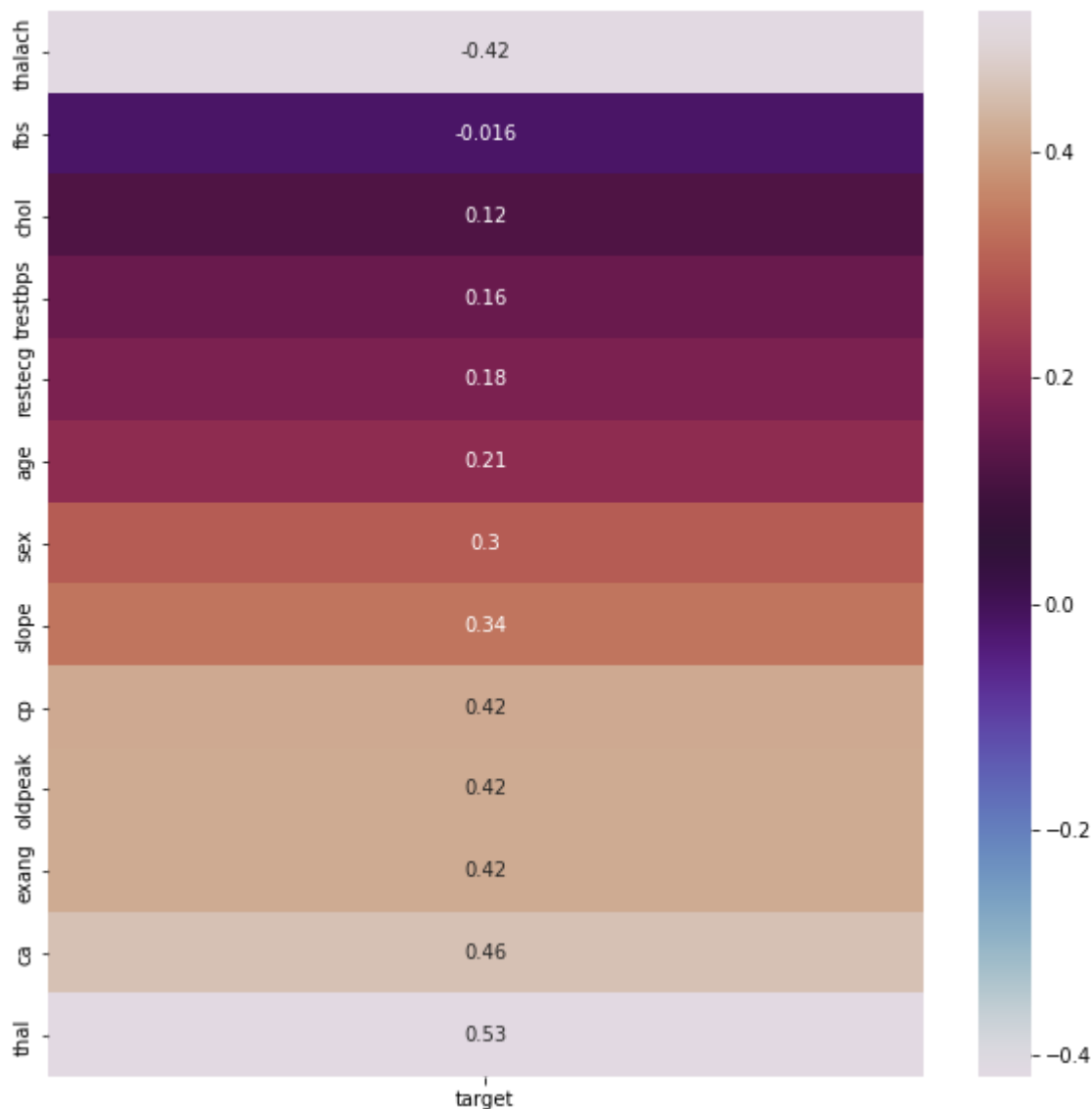
plt.savefig("PiePlots.png")
```



<Figure size 576x432 with 0 Axes>

▼ Target Correlations

```
plt.figure(figsize=(10,10))
sns.heatmap(pd.DataFrame(dataset.corr()['target']).sort_values(by='target').transpose().drop('target'))
plt.savefig("TargetCorrelations.png")
```



▼ Feature Importance

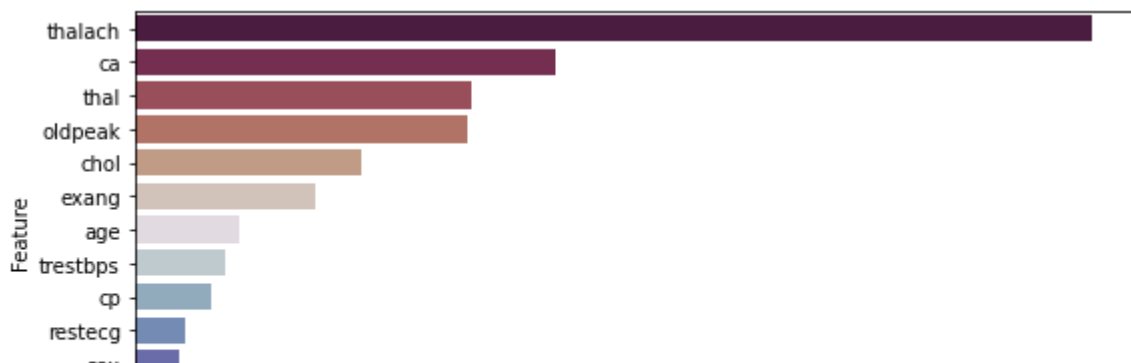
```

X = dataset.drop('target',axis=1)
Y = dataset['target']
from sklearn.feature_selection import SelectKBest, chi2
fs = SelectKBest(score_func=chi2, k='all')
fs.fit(X, Y)
per = []
for i in fs.scores_:
    per.append(round(((i/sum(fs.scores_))*100),3))

features_data = pd.DataFrame({'Feature':X.columns,'Scores':fs.scores_, 'Importance (%)':per}).sort_values('Importance (%)',ascending=False)

plt.figure(figsize=(9,4))
sns.barplot( 'Importance (%)','Feature',orient='h',data=features_data,palette='twilight_shifted_r')
insignificant = features_data.loc[features_data['Importance (%)']<0.005]['Feature'].unique()
features_data = features_data.set_index('Feature')
features_data
plt.savefig("FeatureImportance.png")

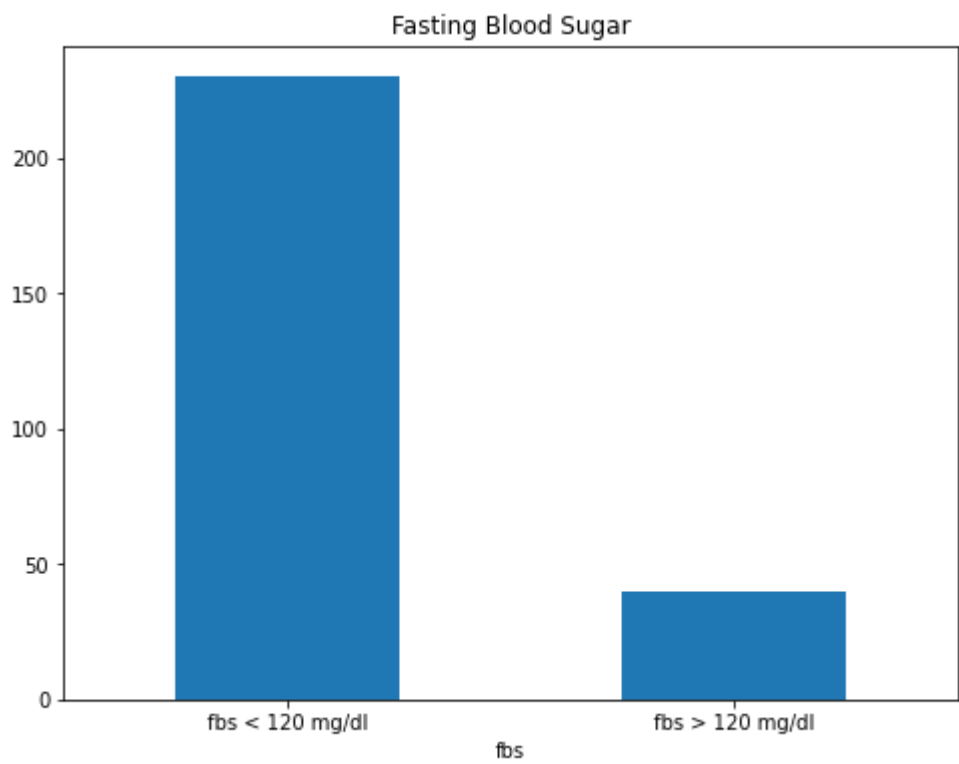
```



▼ Analysing Fasting Blood sugar [fbs]

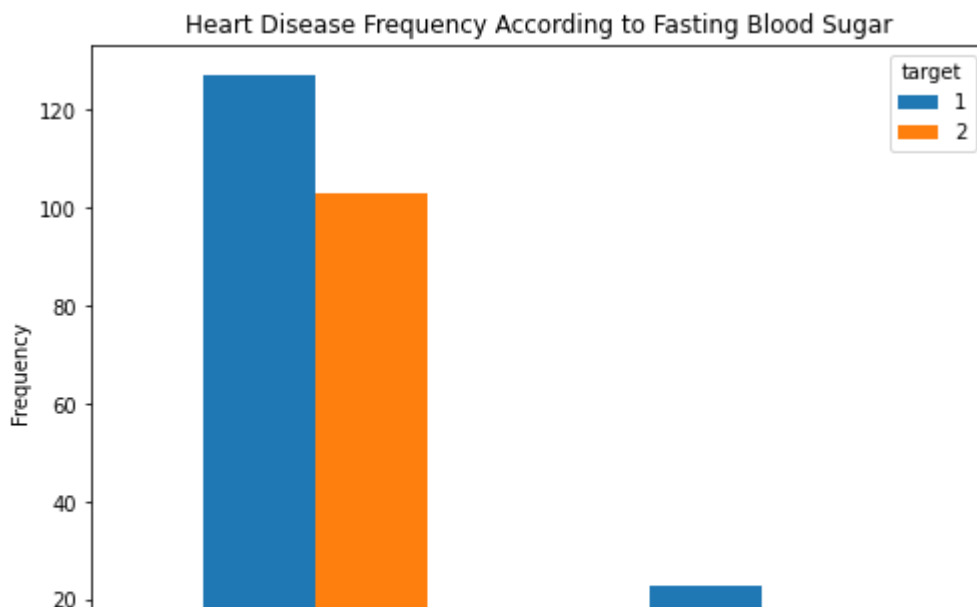
Heart disease according to Fasting Blood sugar

```
# Display fasting blood sugar in bar chart
dataset.groupby(dataset['fbs']).count()['target'].plot(kind = 'bar', title = 'Fasting Blood Sugar',
plt.xticks(np.arange(2), ('fbs < 120 mg/dl', 'fbs > 120 mg/dl'), rotation = 0)
plt.show()
```



▼ Display fasting blood sugar based on the target

```
pd.crosstab(dataset.fbs,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Fasting Blood Sugar')
plt.xlabel('Fasting Blood Sugar')
plt.xticks(np.arange(2), ('fbs < 120 mg/dl', 'fbs > 120 mg/dl'), rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



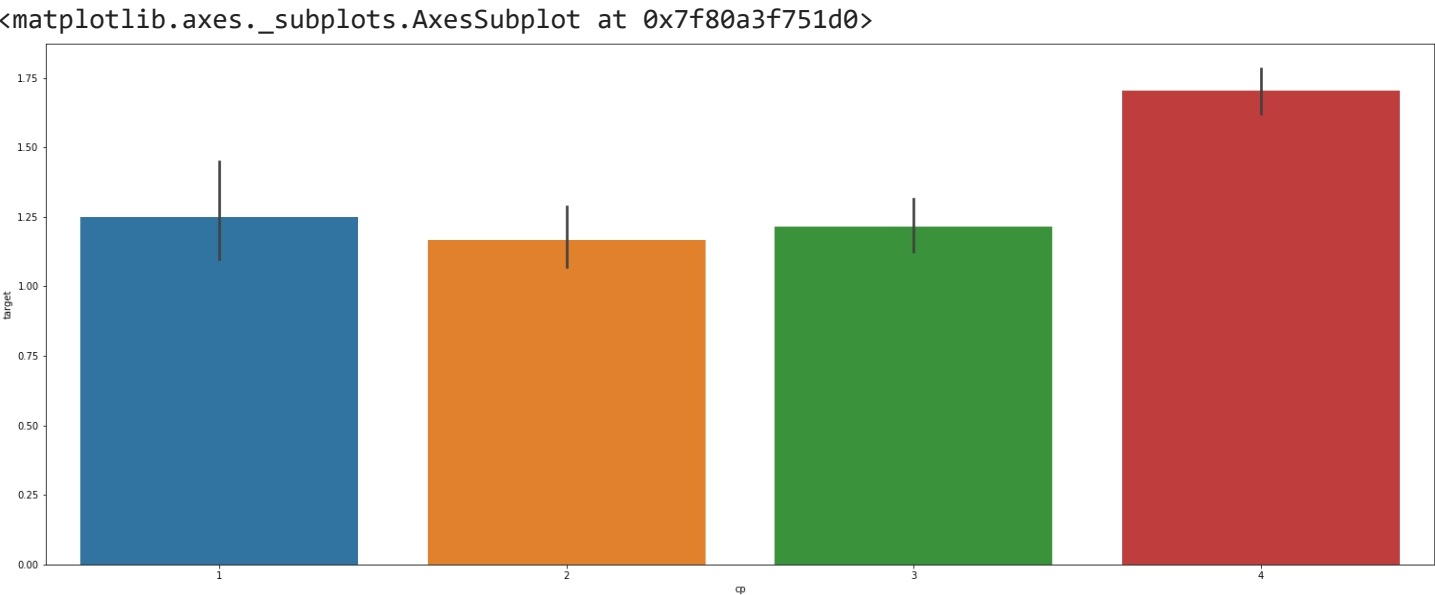
▼ Analysing the Chest Pain [cp] (4 types of chest pain)

[Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic]

```
dataset["cp"].unique()

array([4, 3, 2, 1])
```

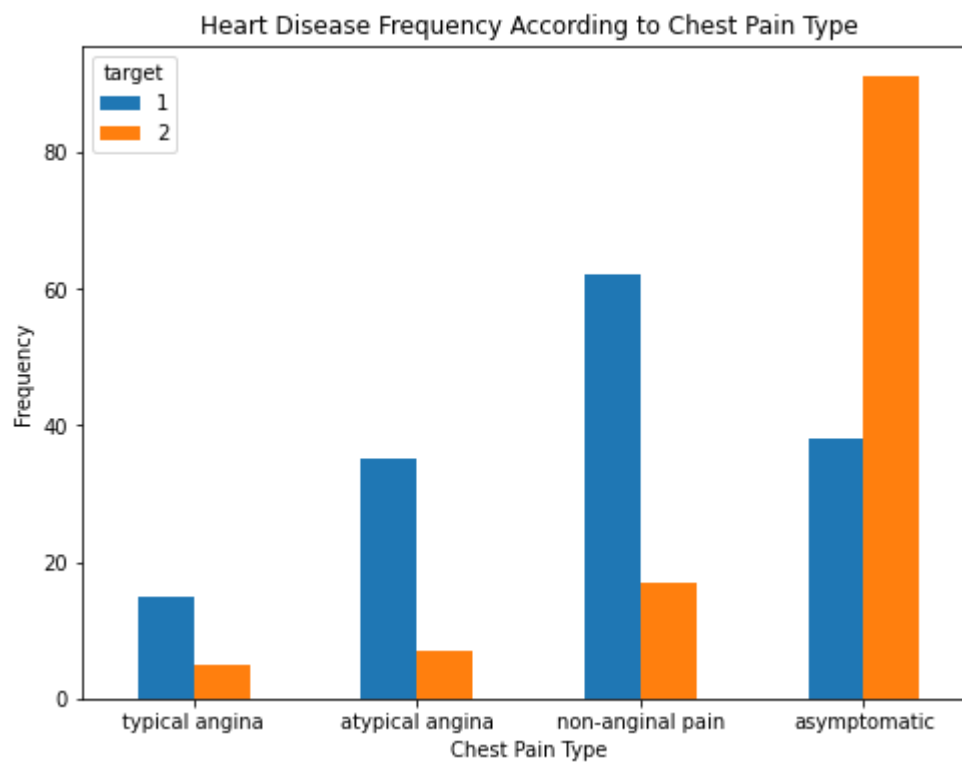
```
plt.figure(figsize=(26, 10))
sns.barplot(dataset["cp"],y)
```



Display chest pain types based on the target

```
pd.crosstab(dataset.cp,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(np.arange(4), ('typical angina', 'atypical angina', 'non-anginal pain', 'asymptomatic'),
```

```
plt.ylabel('Frequency')
plt.show()
```



▼ Analysing Resting Blood Pressure [trestbps]

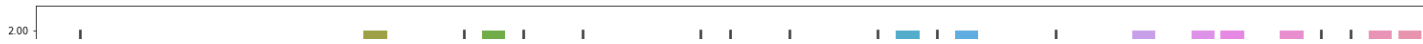
mm Hg on admission to the hospital

```
dataset["trestbps"].unique()
```

```
array([130, 115, 124, 128, 120, 110, 140, 150, 135, 142, 134, 112, 132,
       138, 160, 170, 144, 122, 152, 101, 126, 118, 136, 105, 174, 145,
       108, 156, 106, 104, 94, 146, 148, 178, 125, 100, 165, 180, 158,
       200, 117, 192, 123, 129, 102, 155, 172])
```

```
plt.figure(figsize=(26, 10))
sns.barplot(dataset["trestbps"],y)
```

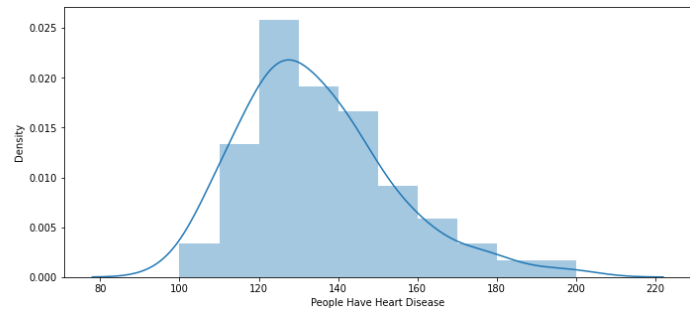
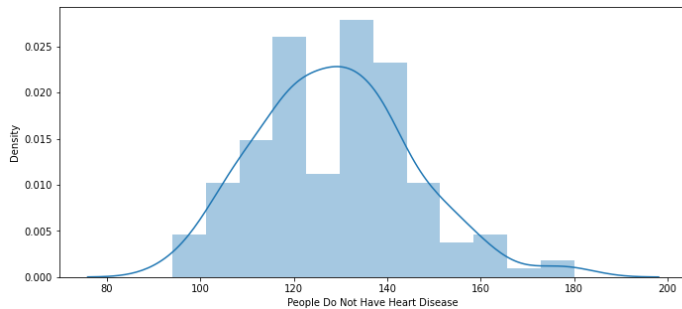
<matplotlib.axes._subplots.AxesSubplot at 0x7f80a403c310>



▼ Display blood pressure distribution based on heart disease



```
fig, (axis1, axis2) = plt.subplots(1, 2, figsize=(25, 5))
ax = sns.distplot(dataset[dataset['target'] == 1]['trestbps'], label='Do not have heart disease', ax=axis1)
ax.set(xlabel='People Do Not Have Heart Disease')
ax = sns.distplot(dataset[dataset['target'] == 2]['trestbps'], label='Have heart disease', ax=axis2)
ax.set(xlabel='People Have Heart Disease')
plt.show()
```



```
# Get min, max and average of the blood pressure of the people do not have heart diseases
print('Min blood pressure of people who do not have heart disease: ', min(dataset[dataset['target'] == 1]['trestbps']))
print('Max blood pressure of people who do not have heart disease: ', max(dataset[dataset['target'] == 1]['trestbps']))
print('Average blood pressure of people who do not have heart disease: ', dataset[dataset['target'] == 1]['trestbps'].mean())
```

```
Min blood pressure of people who do not have heart disease: 94
Max blood pressure of people who do not have heart disease: 180
Average blood pressure of people who do not have heart disease: 128.86666666666667
```

```
# Get min, max and average of the blood pressure of the people have heart diseases
print('Min blood pressure of people who have heart disease: ', min(dataset[dataset['target'] == 2]['trestbps']))
print('Max blood pressure of people who have heart disease: ', max(dataset[dataset['target'] == 2]['trestbps']))
print('Average blood pressure of people who have heart disease: ', dataset[dataset['target'] == 2]['trestbps'].mean())
```

```
Min blood pressure of people who have heart disease: 100
Max blood pressure of people who have heart disease: 200
Average blood pressure of people who have heart disease: 134.44166666666666
```

▼ Analysing the Resting Electrocardiographic Measurement [restecg]

(0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)

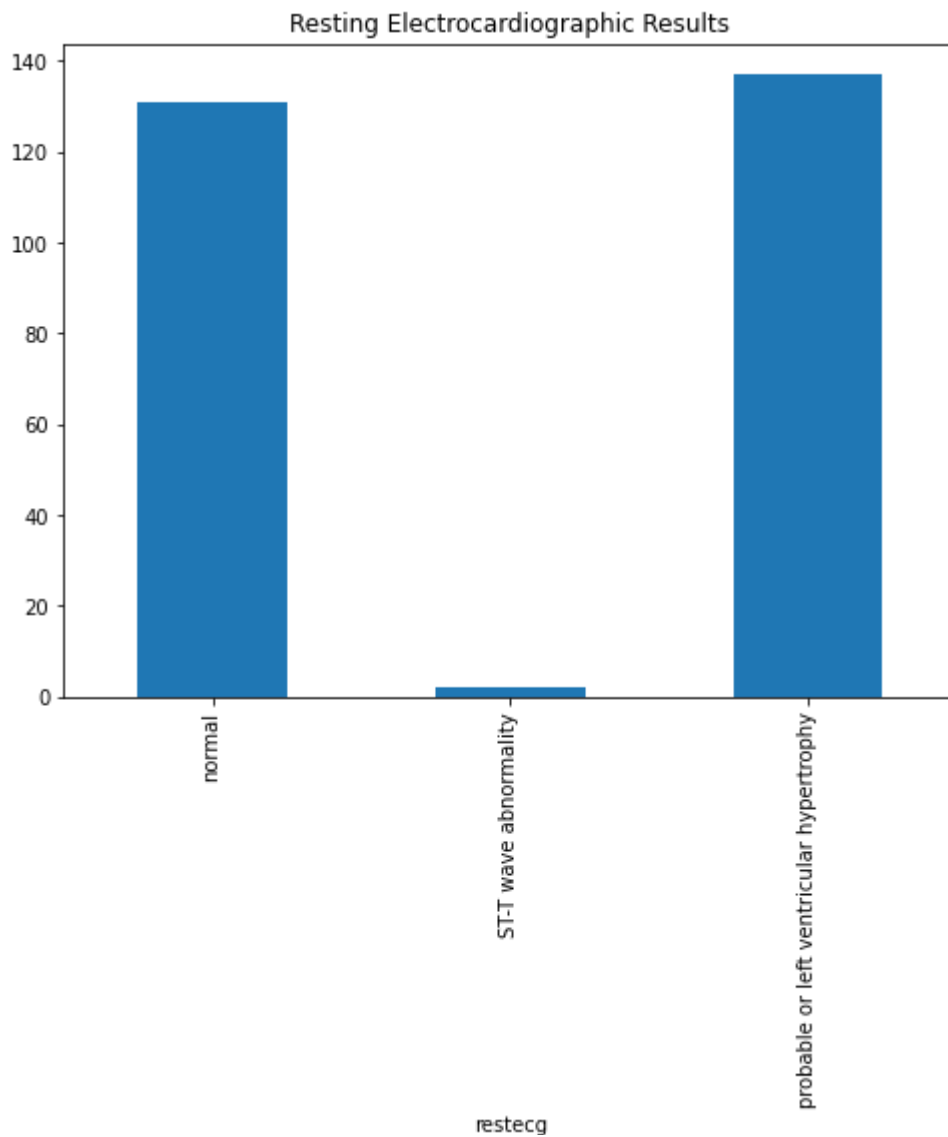
```
dataset["restecg"].unique()
```

```
array([2, 0, 1])
```

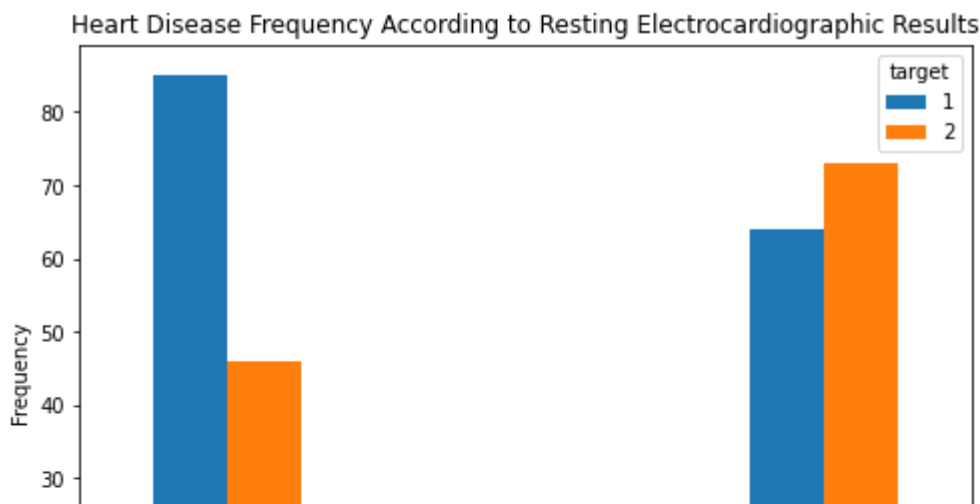
```
# Display electrocardiographic results in bar chart
dataset.groupby(dataset['restecg']).count()['target'].plot(kind='bar', title='Resting Electrocardiographic Measurement')
```



```
plt.xticks(np.arange(3), ('normal', 'ST-T wave abnormality', 'probable or left ventricular hypertrophy'))
plt.show()
```



```
# Display resting electrocardiographic results based on the target
pd.crosstab(dataset.restecg,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Resting Electrocardiographic Results')
plt.xticks(np.arange(3), ('normal', 'ST-T wave abnormality', 'probable or left ventricular hypertrophy'))
plt.xlabel('Resting Electrocardiographic Results')
plt.ylabel('Frequency')
plt.show()
```



Usually the people who do not have heart disease have normal electrocardiographic, whereas the people who have heart disease have probable or left ventricular hypertrophy.



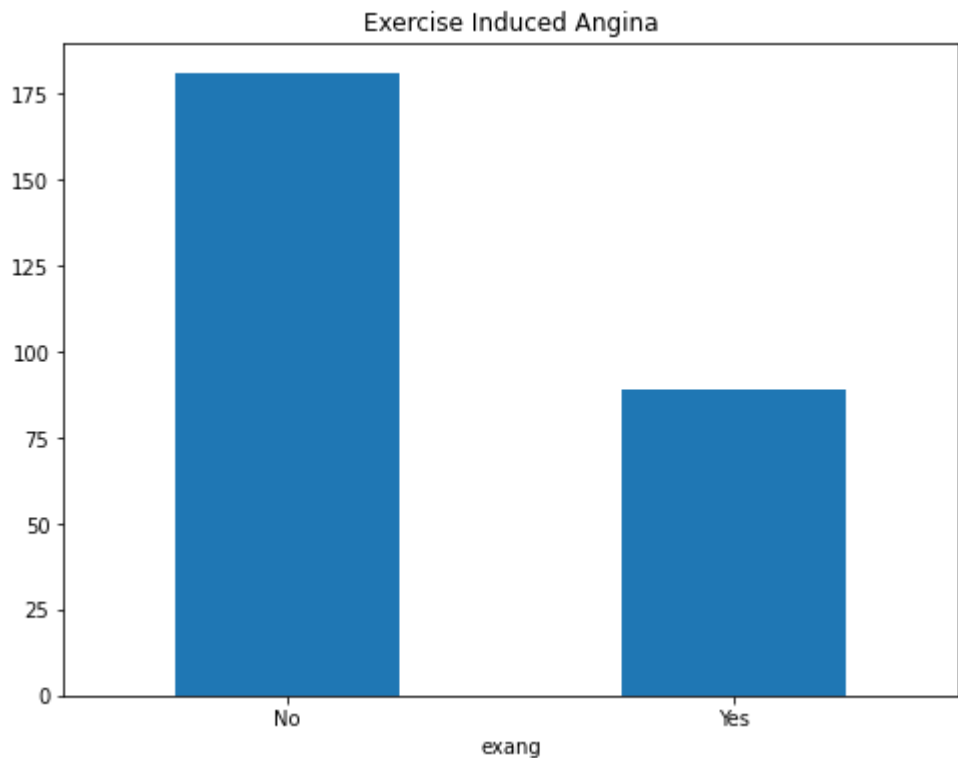
▼ Analysing Exercise Induced Angina [exang]

(1 = yes; 0 = no)

```
dataset["exang"].unique()

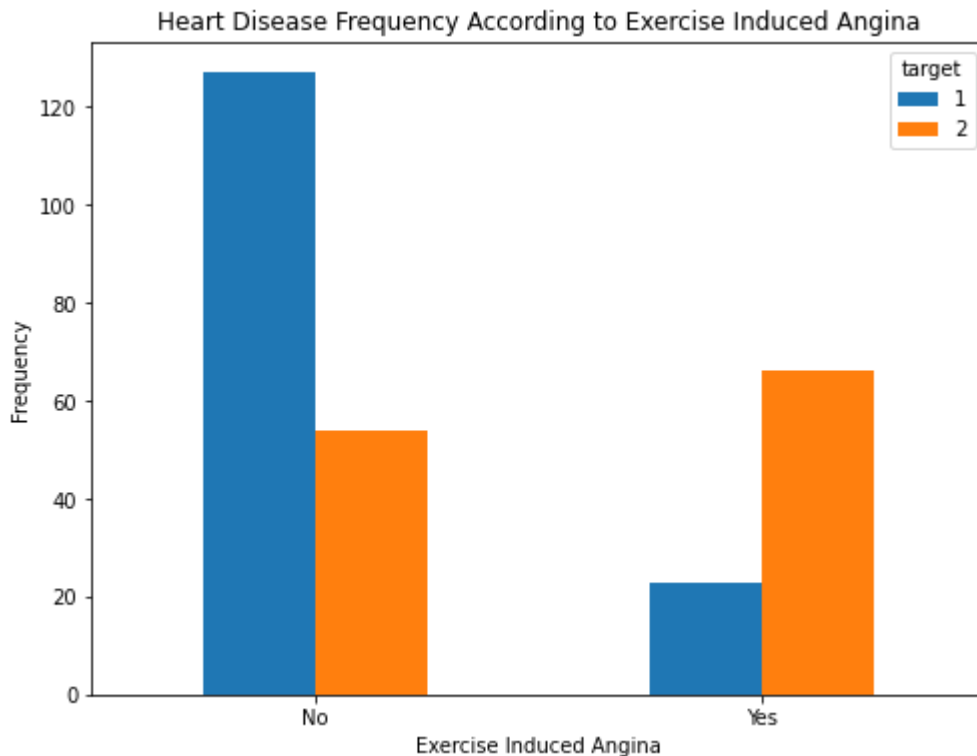
array([0, 1])
```

```
# Display exercise induced angina in bar chart
dataset.groupby(dataset['exang']).count()['target'].plot(kind = 'bar', title = 'Exercise Induced Angina')
plt.xticks(np.arange(2), ('No', 'Yes'), rotation = 0)
plt.show()
```



▼ Display exercise induced angina based on the target

```
pd.crosstab(dataset.exang,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Exercise Induced Angina')
plt.xlabel('Exercise Induced Angina')
plt.xticks(np.arange(2), ('No', 'Yes'), rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



The people who suffer from exercise induced angina are more likely to be infected with the heart disease.

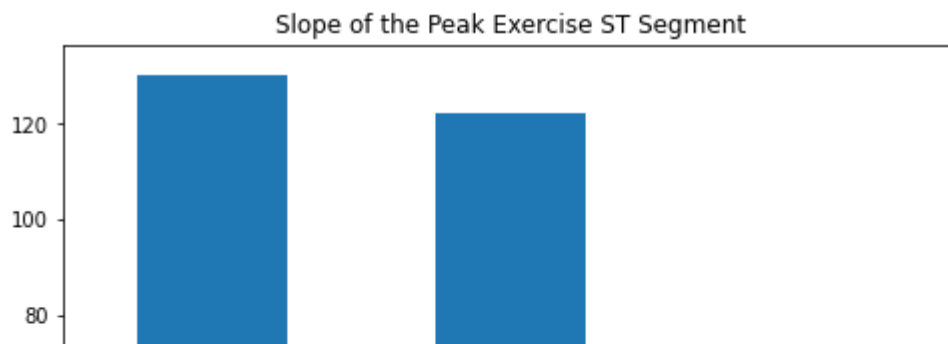
▼ Analysing the Slope of the peak exercise ST segment [slope]

(Value 1: upsloping, Value 2: flat, Value 3: downsloping)

```
dataset["slope"].unique()

array([2, 1, 3])
```

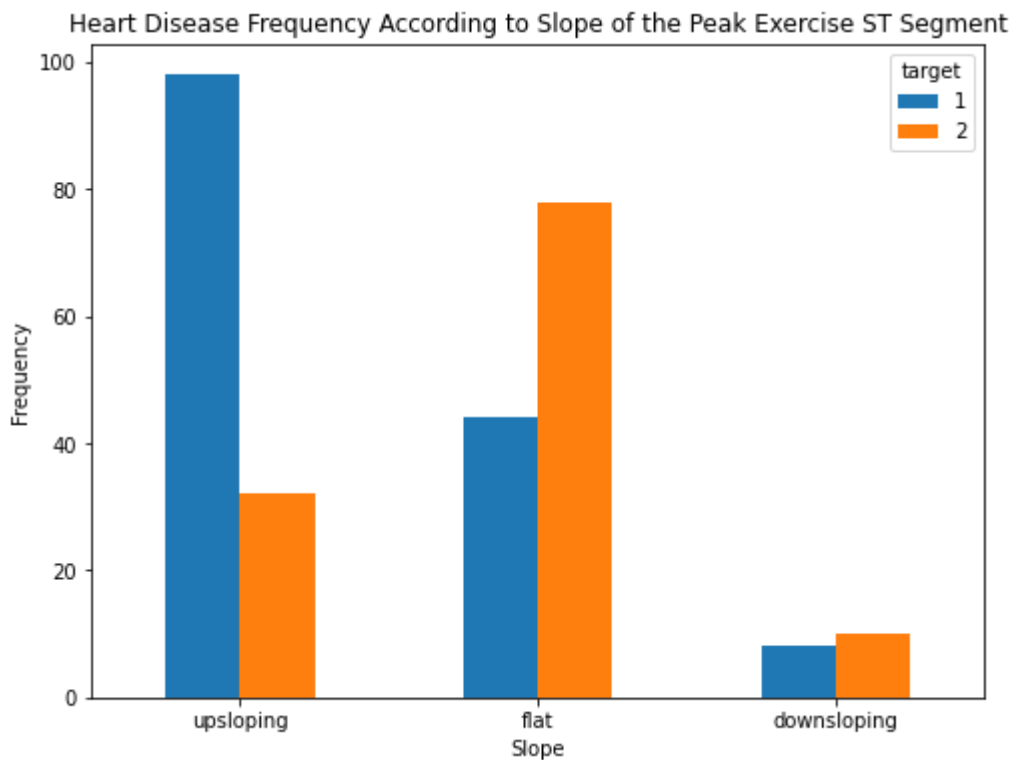
```
# Display slope of the peak exercise ST segment in bar chart
dataset.groupby(dataset['slope']).count()['target'].plot(kind = 'bar', title = 'Slope of the Peak Ex
plt.xticks(np.arange(3), ('upsloping', 'flat', 'downsloping'), rotation = 0)
plt.show()
```



▼ Display slope of the peak exercise ST segment based on the target



```
pd.crosstab(dataset.slope,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Slope of the Peak Exercise ST Segment')
plt.xlabel('Slope')
plt.xticks(np.arange(3), ('upsloping', 'flat', 'downsloping'), rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



As it is clear, the people with flatslope peak ST segment are likely to have heart disease and usually the people who do not have heart disease have upsloping peak ST segment.

▼ Analysing Number of Major Vessels (0-3) colored by flourosopy [ca]

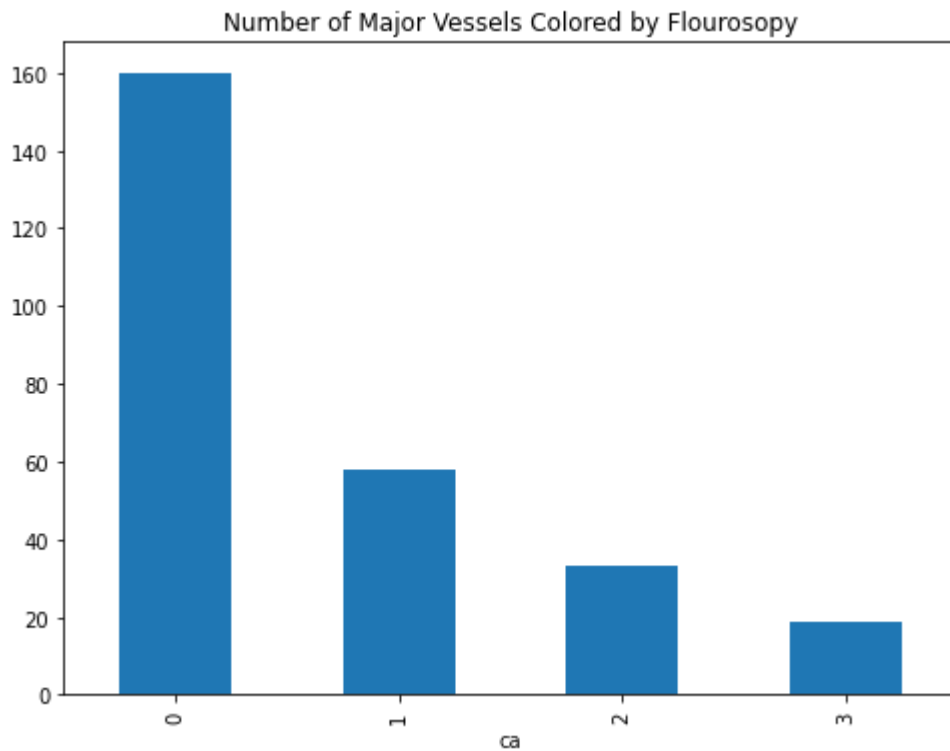
```
dataset["ca"].unique()

array([3, 0, 1, 2])
```

▼ Display number of major vessels in bar chart

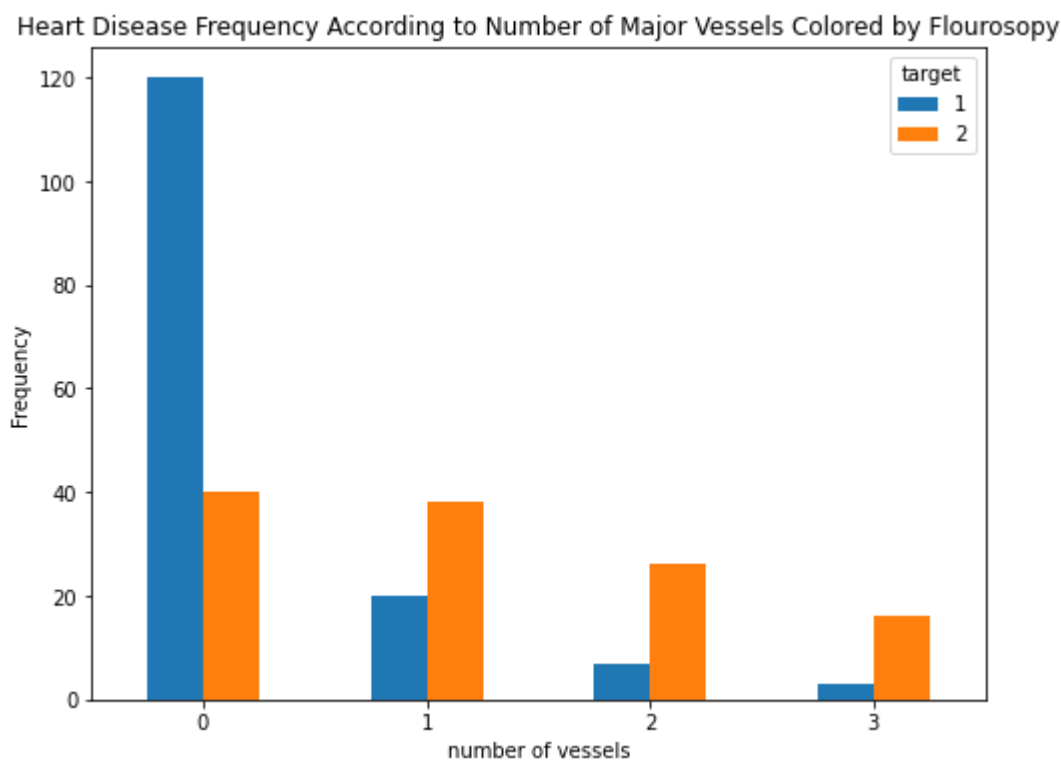
```
dataset.groupby(dataset['ca']).count()['target'].plot(kind = 'bar', title = 'Number of Major Vessels Colored by Flourosopy',
figsize = (8, 6))

plt.show()
```



▼ Display number of vessels based on the target

```
pd.crosstab(dataset.ca,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Number of Major Vessels Colored by Flourosopy')
plt.xlabel('number of vessels')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



As it is clear, the people who do not have heart disease usually do not have major vessels colored by flourosopy.

▼ Analysing a Blood Disorder called Thalassemia [thal]

(3 = normal ; 6 = fixed defect ; 7 = reversable defect)

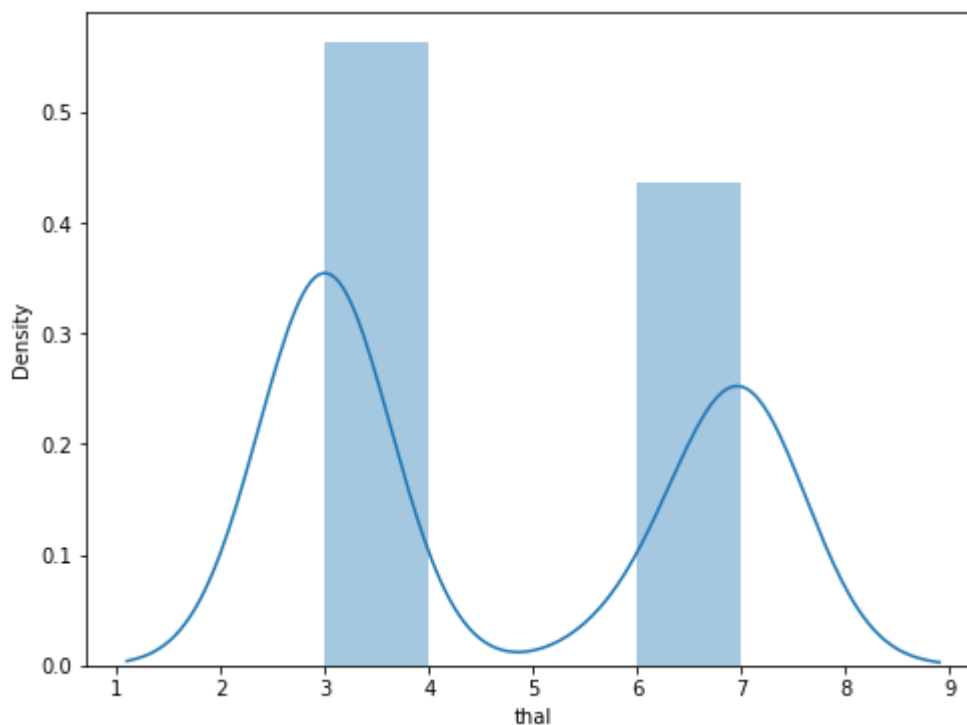
```
dataset["thal"].unique()
```

```
array([3, 7, 6])
```

▼ plotting the thalassemia distribution (3,6,7)

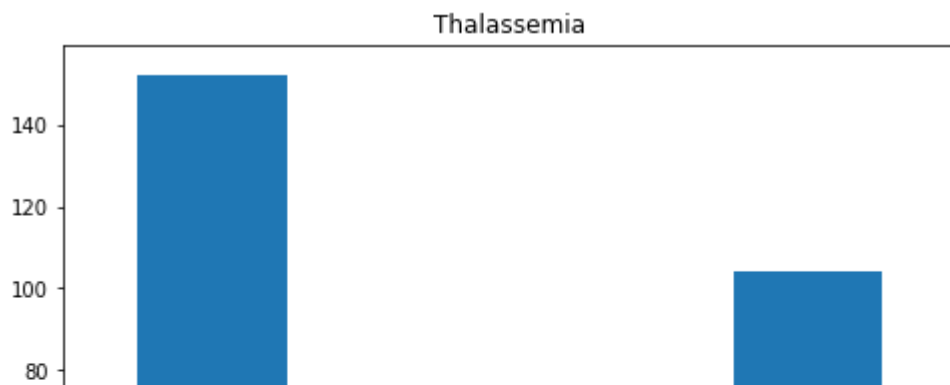
```
sns.distplot(dataset["thal"])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f80a3768e90>



▼ Display thalassemia in bar chart

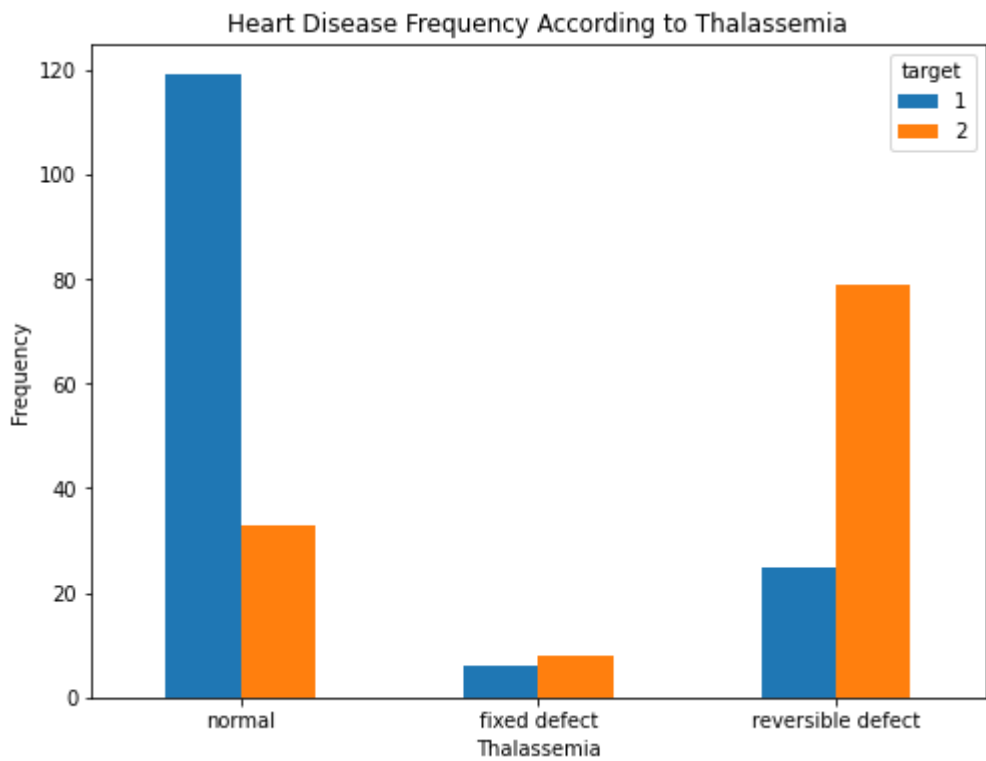
```
dataset.groupby(dataset['thal']).count()['target'].plot(kind = 'bar', title = 'Thalassemia')  
plt.xticks(np.arange(3), ('normal', 'fixed defect', 'reversible defect'), rotation = 0)  
plt.show()
```



Thalassemia compared with target



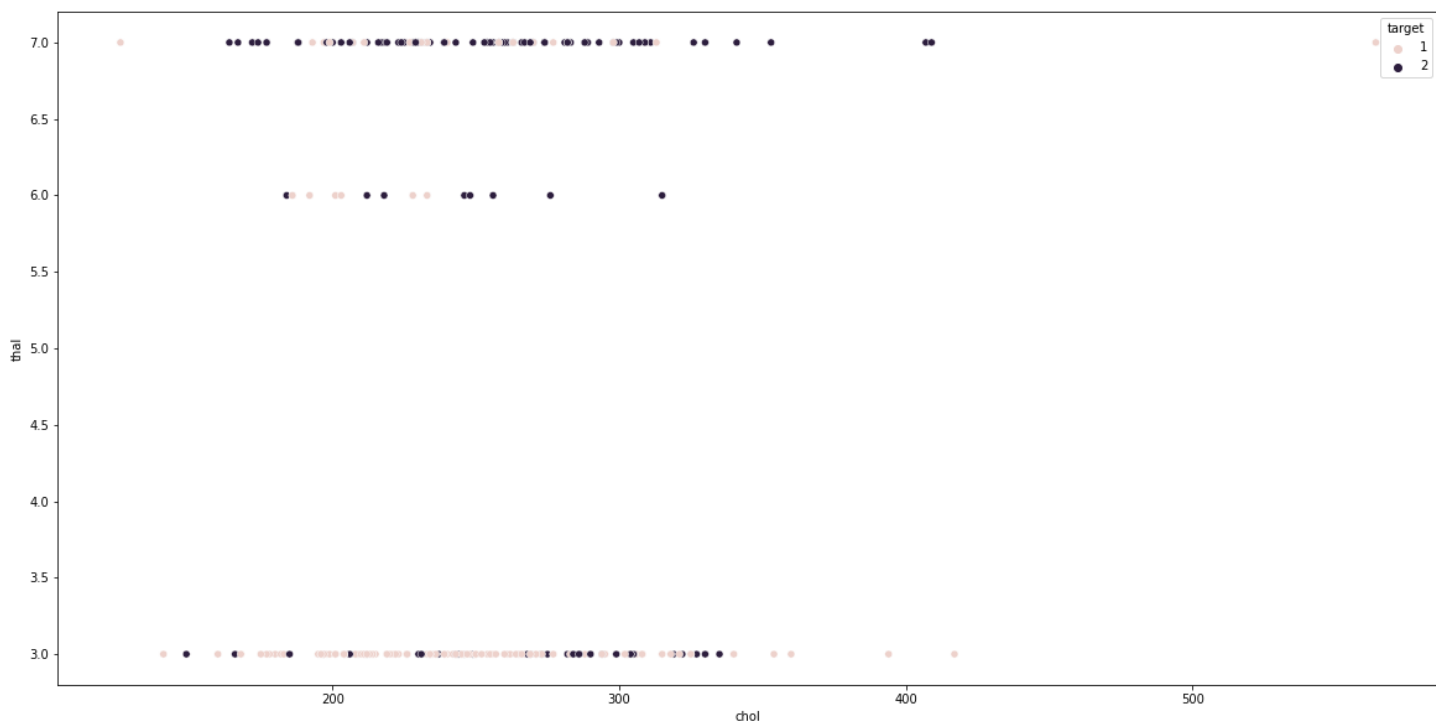
```
pd.crosstab(dataset.thal,dataset.target).plot(kind = "bar", figsize = (8, 6))
plt.title('Heart Disease Frequency According to Thalassemia')
plt.xlabel('Thalassemia')
plt.xticks(np.arange(3), ('normal', 'fixed defect', 'reversible defect'), rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



As it is clear, the people with reversible defect are likely to have heart disease.

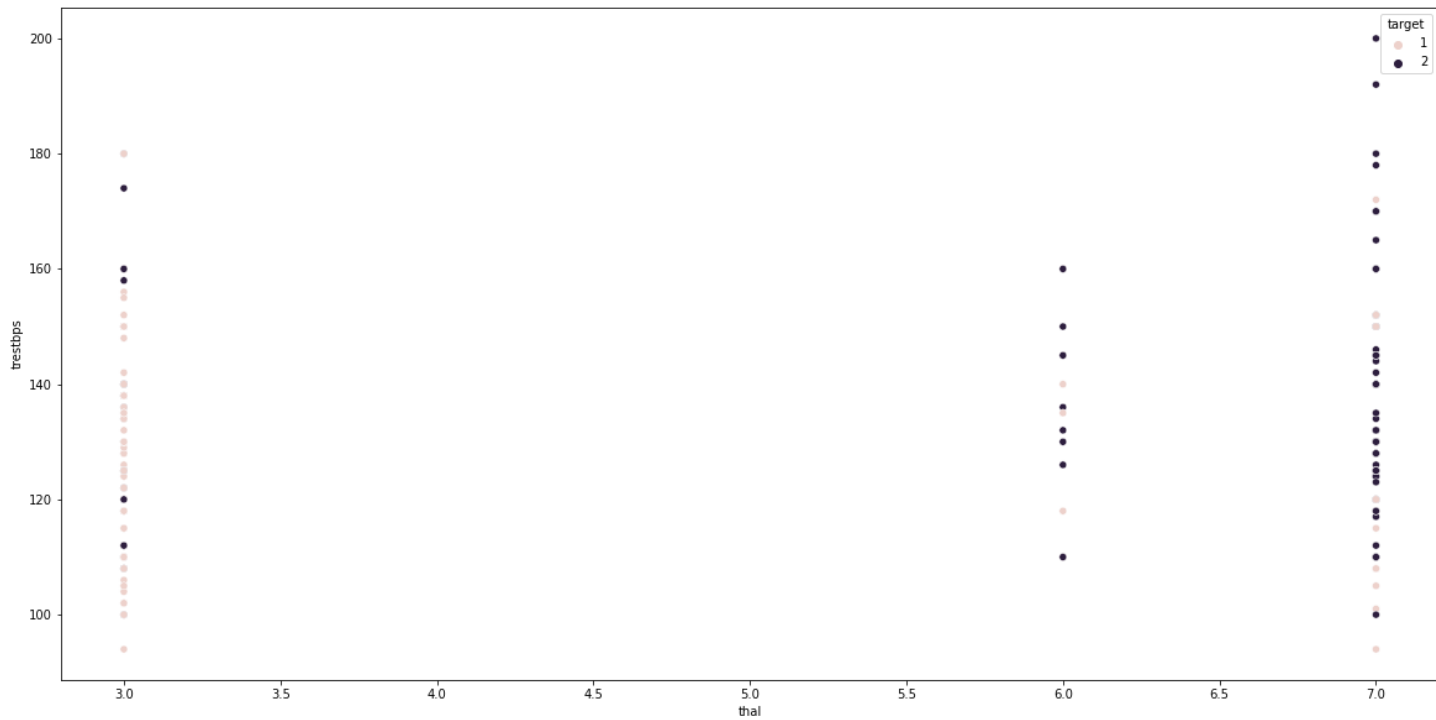
Thalassemia vs Cholesterol Scatterplot

```
plt.figure(figsize=(20,10))
sns.scatterplot(x='chol',y='thal',data=dataset,hue='target')
plt.show()
```



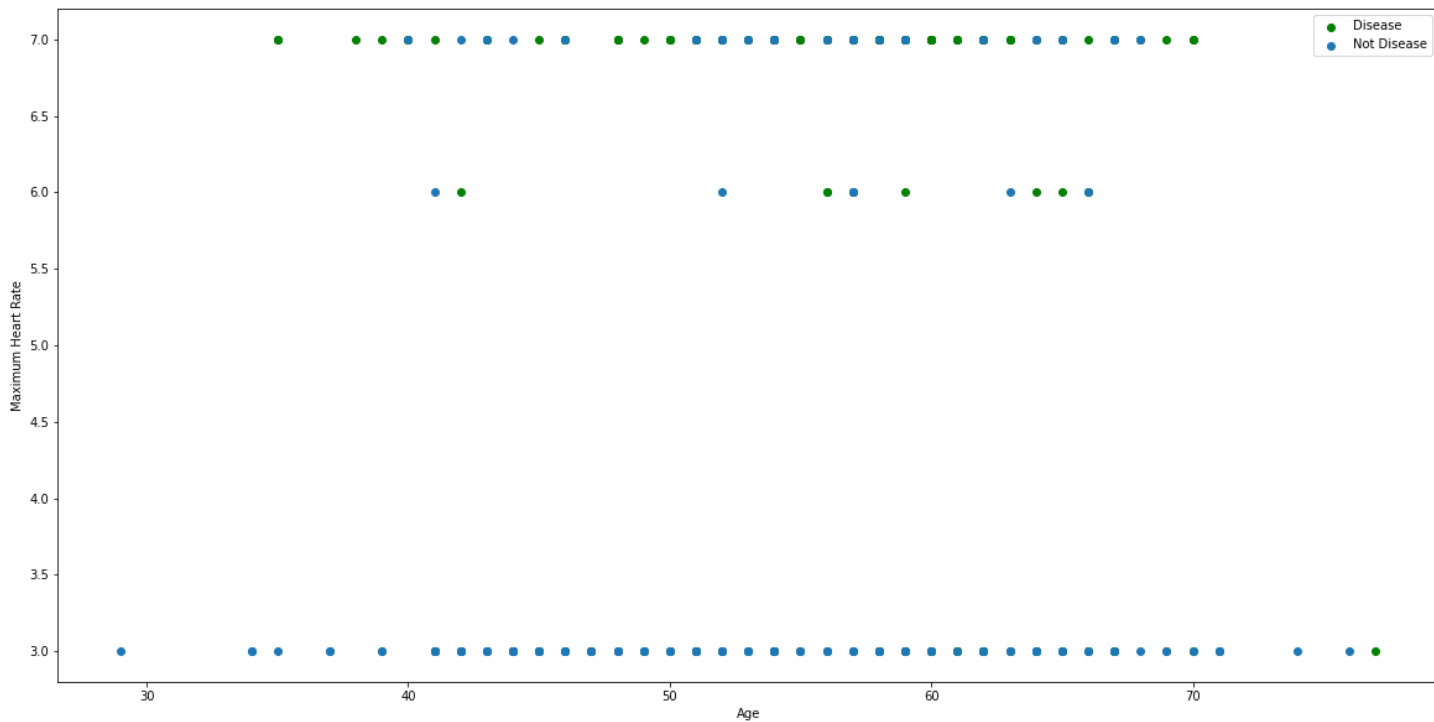
▼ Thalassemia vs Resting blood pressure Scatterplot

```
plt.figure(figsize=(20,10))
sns.scatterplot(x='thal',y='trestbps',data=dataset,hue='target')
plt.show()
```



▼ Thalassemia vs Age Scatterplot


```
plt.figure(figsize=(20, 10))
plt.scatter(x=dataset.age[dataset.target==2], y=dataset.thal[(dataset.target==2)], c="green")
plt.scatter(x=dataset.age[dataset.target==1], y=dataset.thal[(dataset.target==1)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



Maximum Heart Rate vs Oldpeak(Exercise induced ST-depression in comparison with the state of rest)

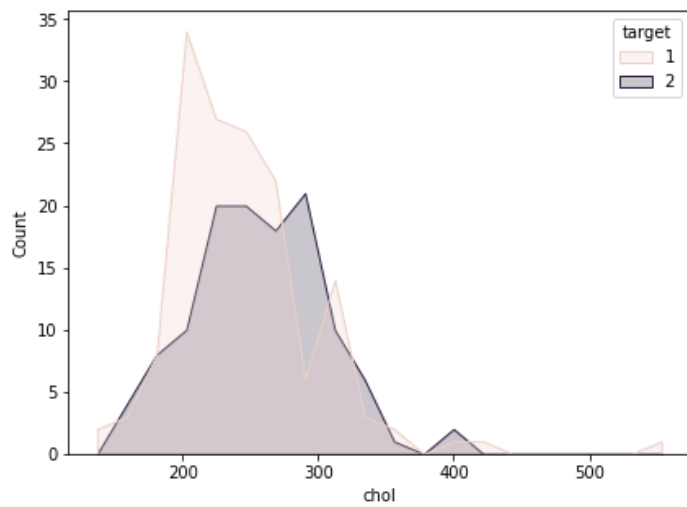
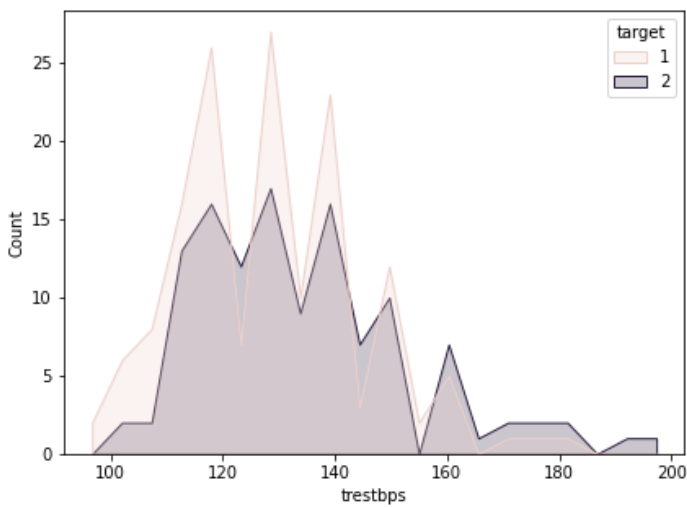
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(data=dataset,hue='target',x='thalach',bins=20,element='poly')
plt.subplot(1,2,2)
sns.histplot(data=dataset,hue='target',x='oldpeak',bins=20,element='poly')
plt.savefig("Thalach&oldpeak_Histplot.png")
```



Resting Blood Pressure vs Cholestrol



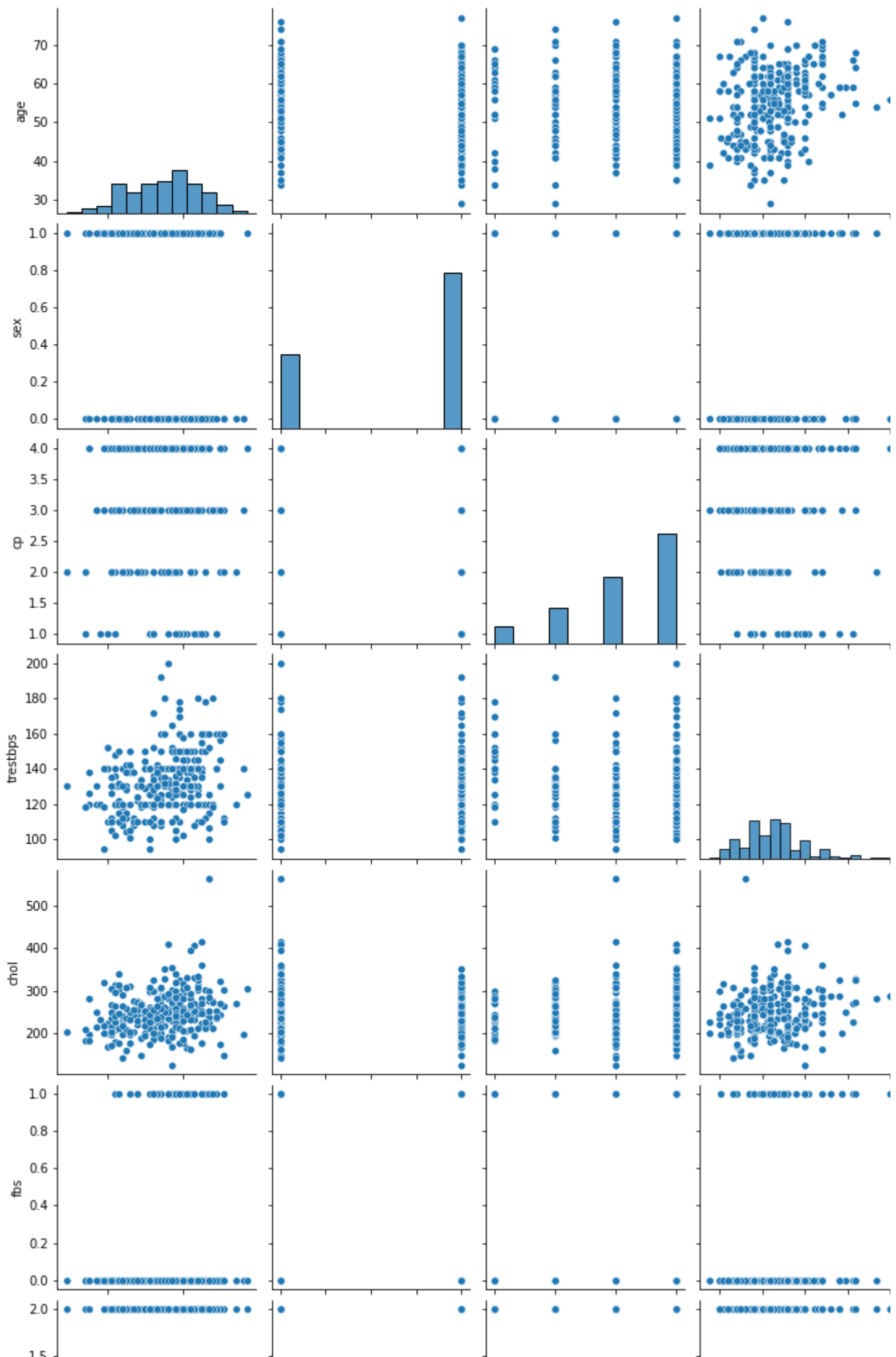
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
sns.histplot(data=dataset,hue='target',x='trestbps',bins=20,element='poly')
plt.subplot(1,2,2)
sns.histplot(data=dataset,hue='target',x='chol',bins=20,element='poly')
plt.savefig("Resting_blood_pressure&chol_Histplot.png")
```

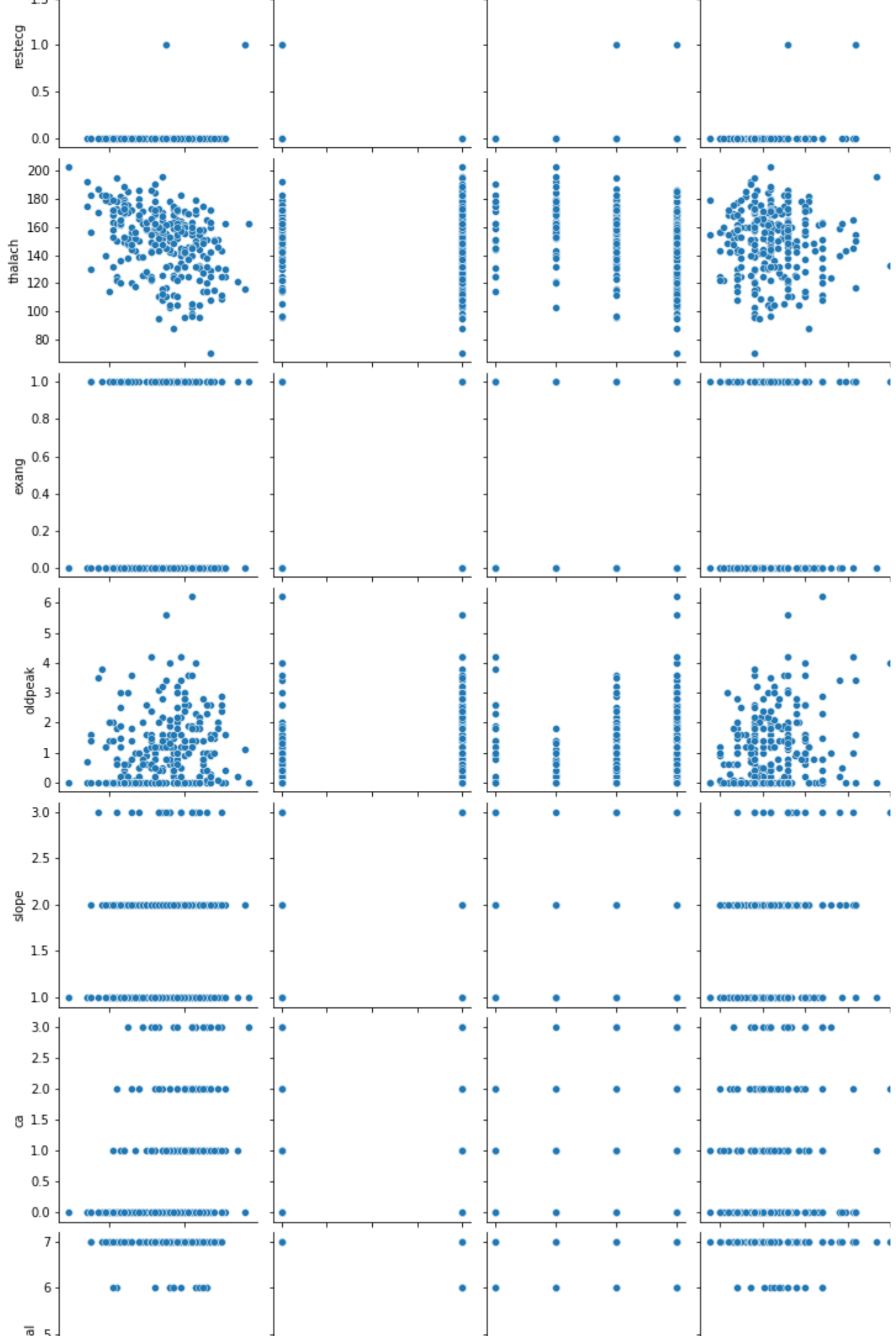


Pair Plots

```
sns.pairplot(data=dataset)
```

<seaborn.axisgrid.PairGrid at 0x7f80a3e1ab10>



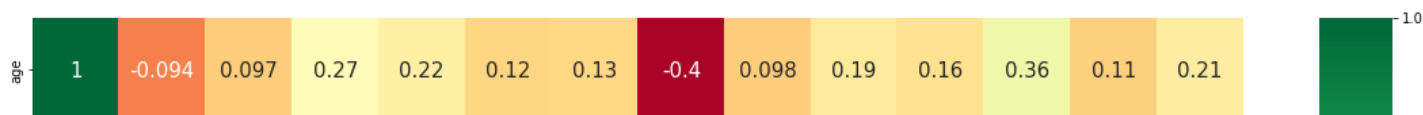


▼ Correlation Matrix

The best way to compare relationship between various features is to look at the correlation matrix between those features.

```
corr_matrix = dataset.corr()  
top_corr_feature = corr_matrix.index  
plt.figure(figsize=(20, 20))  
sns.heatmap(dataset[top_corr_feature].corr(), annot=True, cmap="RdYlGn", annot_kws={"size":15})
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f80a0b15650>



Taking a look at the correlation matrix above, it's easy to see that a few features have negative correlation with the target value while some have positive.



▼ Data Processing

After exploring the dataset, we observed that we need to convert some categorical variables into dummy variables and scale all the values before training the Machine Learning models. First, we'll use the `get_dummies` method to create dummy columns for categorical variables.



```
#dataset = pd.get_dummies(dataset, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca'],
```



Now, we will use the `StandardScaler` from `sklearn` to scale my dataset.



```
#standardScaler = StandardScaler()
#columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
#dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```



```
#dataset.head()
```



```
#dataset.describe()
```



▼ Train Test Split

We'll now import `train_test_split` to split our dataset into training and testing datasets. Then, we'll import all Machine Learning models we'll be using to train and test the data.



```
Y = dataset['target'].values
X = dataset.drop('target',axis=1).values
```

```
X.shape
```

```
(270, 13)
```

```
Y.shape
```

```
(270,)
```

```
# Split the dataset into training and testing.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
print("Training features have {0} records and Testing features have {1} records.".\
      format(X_train.shape[0], X_test.shape[0]))
```

Training features have 216 records and Testing features have 54 records.

Checking distribution of target variable in train test split

```
print('-----Training Set-----')
print(X_train.shape)
print(Y_train.shape)
print('-----Test Set-----')
print(X_test.shape)
print(Y_test.shape)
```

```
-----Training Set-----
(216, 13)
(216,)
-----Test Set-----
(54, 13)
(54,)
```

▼ Machine Learning Model

▼ 1. Random Forest Classifier

Now, we'll use the ensemble method, Random Forest Classifier, to create the model and vary the number of estimators to see their effect.

```
max_accuracy = 0
```

```
for x in range(500):
    rf_classifier = RandomForestClassifier(random_state=x)
    rf_classifier.fit(X_train,Y_train)
    Y_pred_rf = rf_classifier.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x
```

```
print(max_accuracy)
print(best_x)
```

```
85.19
135
```

```
rf_classifier = RandomForestClassifier(random_state=best_x)
rf_classifier.fit(X_train,Y_train)
Y_pred_rf = rf_classifier.predict(X_test)
Y_pred_rf.shape
```

```
(54,)
```

```
score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
score_rf
```

85.19

▼ Model Evaluation:

In this step we will first define which evaluation metrics we will use to evaluate our model. The most important evaluation metric for this problem domain is sensitivity, specificity, Precision, F1-measure, ROC and mathew correlation coefficient.

Mathew Correlation coefficient (MCC)

The Matthews correlation coefficient (MCC), instead, is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset.

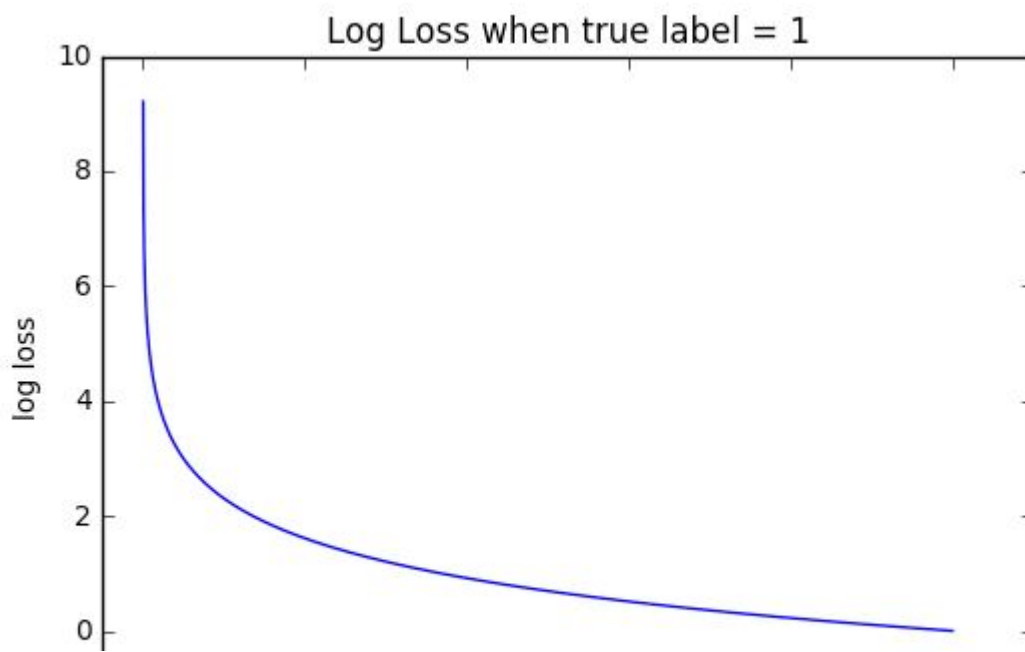
$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

(worst value: -1; best value: +1)

Log Loss

Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of our machine learning models is to minimize this value. A perfect model would have a log loss of 0. Log loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high log loss.

The graph below shows the range of possible log loss values given a true observation (isDog = 1). As the predicted probability approaches 1, log loss slowly decreases. As the predicted probability decreases, however, the log loss increases rapidly. Log loss penalizes both types of errors, but especially those predications that are confident and wrong!



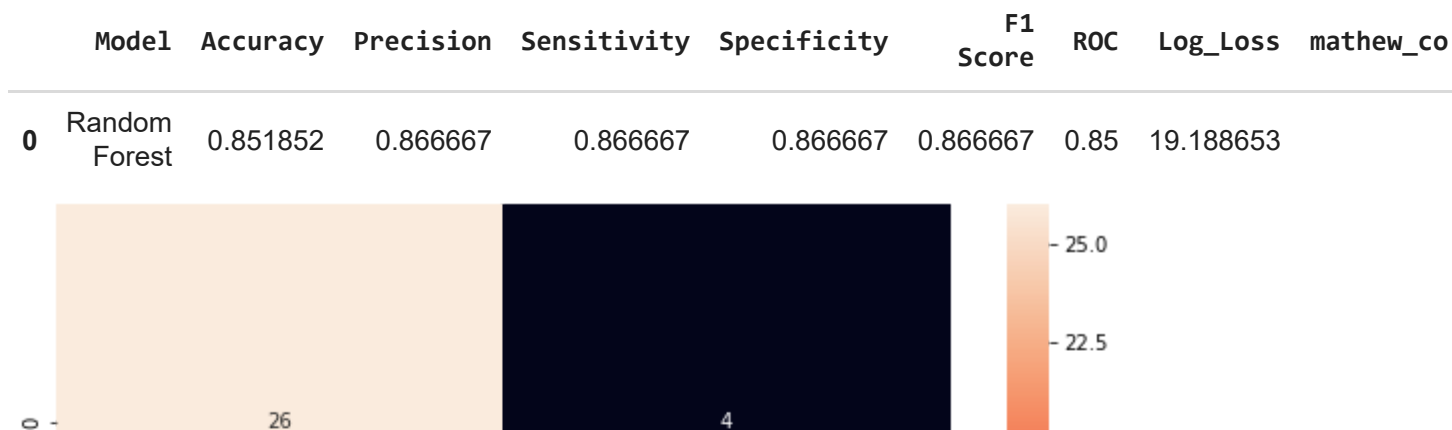
```
y_pred_rfe = rf_classifier.predict(X_test)
```

```
plt.figure(figsize=(10, 8))  
CM=confusion_matrix(Y_test,y_pred_rfe)  
sns.heatmap(CM, annot=True)
```

```
TN = CM[0][0]  
FN = CM[1][0]  
TP = CM[1][1]  
FP = CM[0][1]  
specificity = TN/(TN+FP)  
loss_log = log_loss(Y_test, y_pred_rfe)  
acc= accuracy_score(Y_test, y_pred_rfe)  
roc=roc_auc_score(Y_test, y_pred_rfe)  
prec = precision_score(Y_test, y_pred_rfe)  
rec = recall_score(Y_test, y_pred_rfe)  
f1 = f1_score(Y_test, y_pred_rfe)
```

```
mathew = matthews_corrcoef(Y_test, y_pred_rfe)  
model_results =pd.DataFrame([['Random Forest',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],  
                             columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score',
```

```
model_results
```



```
Y_pred_rf = np.around(Y_pred_rf)
print(metrics.classification_report(Y_test,Y_pred_rf))
```

```

              precision    recall  f1-score   support

     1         0.87         0.87         0.87         30
     2         0.83         0.83         0.83         24

   accuracy          0.85          0.85          0.85          54
  macro avg          0.85          0.85          0.85          54
 weighted avg          0.85          0.85          0.85          54

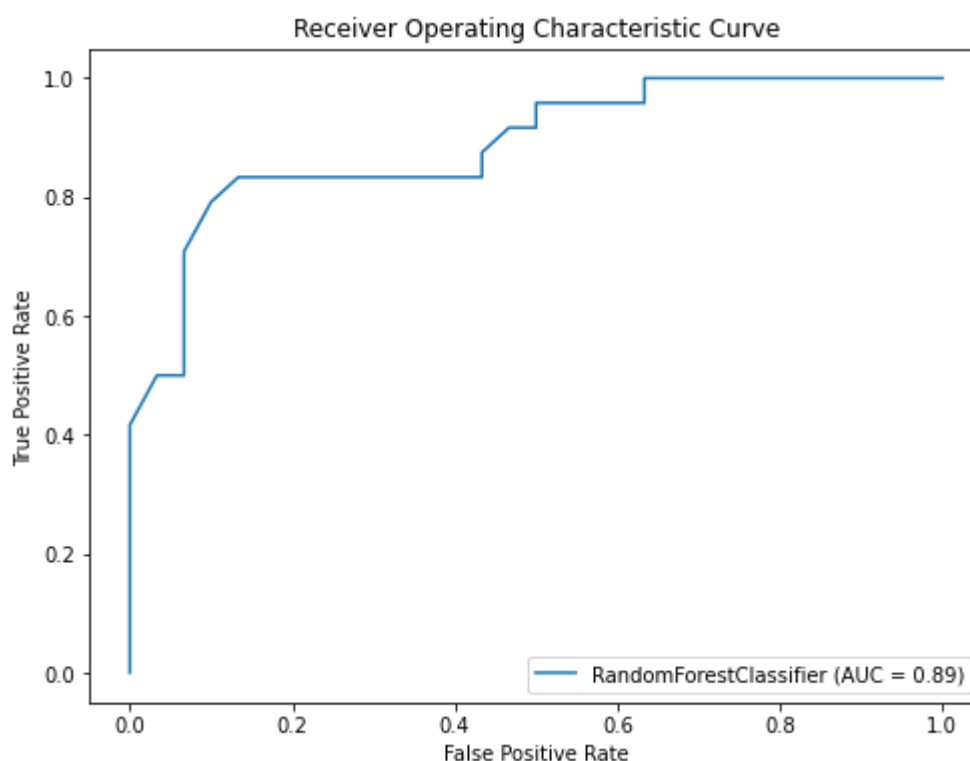
```



```

plot_roc_curve(rf_classifier,X_test,Y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
plt.savefig("RF.png")

```



▼ 2. K-Nearest Neighbors Classifier

The classification score varies based on different values of neighbors that we choose. Thus, we'll plot a score graph for different values of K (neighbors) and check when do we achieve the best score.

```
knn_classifier= KNeighborsClassifier(n_neighbors=31,leaf_size=30)
knn_classifier.fit(X_train,Y_train)
Y_pred_knn = knn_classifier.predict(X_test)
score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)
score_knn

64.81
```

▼ Model Evaluation:

```
y_pred_knne = knn_classifier.predict(X_test)

plt.figure(figsize=(10, 8))
CM=confusion_matrix(Y_test,y_pred_knne)
sns.heatmap(CM, annot=True)

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(Y_test, y_pred_knne)
acc= accuracy_score(Y_test, y_pred_knne)
roc=roc_auc_score(Y_test, y_pred_knne)
prec = precision_score(Y_test, y_pred_knne)
rec = recall_score(Y_test, y_pred_knne)
f1 = f1_score(Y_test, y_pred_knne)

mathew = matthews_corrcoef(Y_test, y_pred_knne)
model_results =pd.DataFrame([[ 'K-Nearest Neighbors ',acc, prec,rec,specificity, f1,roc, loss_log,mat
                                columns = [ 'Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score',

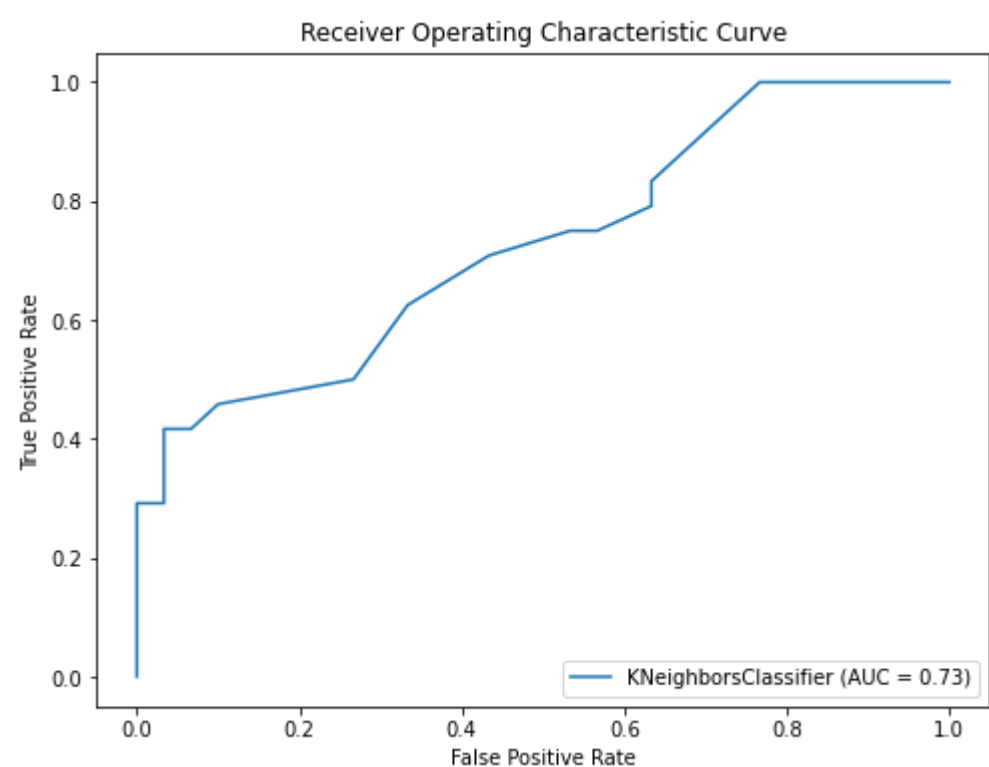
model_results
```

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	math
0	K-Nearest Neighbors	0.648148	0.689655	0.666667	0.666667	0.677966	0.645833	19.188653	

```
Y_pred_knn = np.around(Y_pred_knn)
print(metrics.classification_report(Y_test,Y_pred_knn))
```

	precision	recall	f1-score	support
1	0.69	0.67	0.68	30
2	0.60	0.62	0.61	24
accuracy			0.65	54
macro avg	0.64	0.65	0.65	54
weighted avg	0.65	0.65	0.65	54

```
plot_roc_curve(knn_classifier,X_test,Y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
plt.savefig("KNN.png")
```



▼ 3. Decision Tree Classifier

Here, we'll use the Decision Tree Classifier to model the problem at hand. We'll vary between a set of `max_features` and see which returns the best accuracy.

```
dt_classifier = DecisionTreeClassifier(
    max_depth=20,
    min_samples_split=2,
    min_samples_leaf=1,
```

```

min_weight_fraction_leaf=0.00001,
max_features='auto',
random_state=46)
dt_classifier.fit(X_train, Y_train)
Y_pred_dt=dt_classifier.predict(X_test)
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
score_dt

```

70.37

▼ Model Evaluation:

```

y_pred_dte = dt_classifier.predict(X_test)

```

```

plt.figure(figsize=(10, 8))
CM=confusion_matrix(Y_test,y_pred_dte)
sns.heatmap(CM, annot=True)

```

```

TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(Y_test, y_pred_dte)
acc= accuracy_score(Y_test, y_pred_dte)
roc=roc_auc_score(Y_test, y_pred_dte)
prec = precision_score(Y_test, y_pred_dte)
rec = recall_score(Y_test, y_pred_dte)
f1 = f1_score(Y_test, y_pred_dte)

```

```

mathew = matthews_corrcoef(Y_test, y_pred_dte)
model_results =pd.DataFrame([['Decision Tree',acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
                             columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score',


```

```

model_results

```

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_cor
0	Decision Tree	0.703704	0.733333	0.733333	0.733333	0.733333	0.7	19.188653	

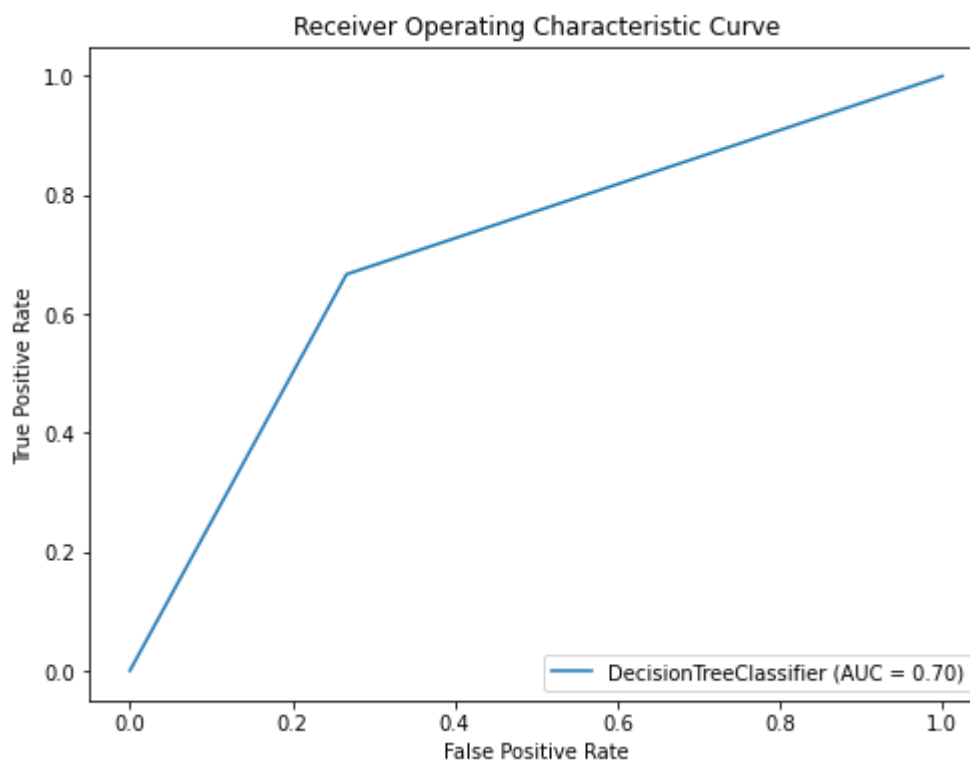


```
Y_pred_dt = np.around(Y_pred_dt)
print(metrics.classification_report(Y_test,Y_pred_dt))
```

	precision	recall	f1-score	support
1	0.73	0.73	0.73	30
2	0.67	0.67	0.67	24
accuracy			0.70	54
macro avg	0.70	0.70	0.70	54
weighted avg	0.70	0.70	0.70	54



```
plot_roc_curve(dt_classifier,X_test,Y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
```



4. Naive Bayes Classifier

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

```
nb_classifier = GaussianNB( var_smoothing=1e-50)
nb_classifier.fit(X_train,Y_train)
```

```
nb_classifier.predict(X_test)
Y_pred_nb = nb_classifier.predict(X_test)
score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)
score_nb
```

74.07

▼ Model Evaluation:

```
y_pred_nbe = nb_classifier.predict(X_test)
```

```
plt.figure(figsize=(10, 8))
CM=confusion_matrix(Y_test,y_pred_nbe)
sns.heatmap(CM, annot=True)
```

```
TN = CM[0][0]
FN = CM[1][0]
TP = CM[1][1]
FP = CM[0][1]
specificity = TN/(TN+FP)
loss_log = log_loss(Y_test, y_pred_nbe)
acc= accuracy_score(Y_test, y_pred_nbe)
roc=roc_auc_score(Y_test, y_pred_nbe)
prec = precision_score(Y_test, y_pred_nbe)
rec = recall_score(Y_test, y_pred_nbe)
f1 = f1_score(Y_test, y_pred_nbe)
```

```
mathew = matthews_corrcoef(Y_test, y_pred_nbe)
model_results =pd.DataFrame(['Naive Bayes ',acc, prec,rec,specificity, f1,roc, loss_log,mathew]),
                        columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score',

model_results
```

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_co
n	Naive	0.740741	0.766667	0.766667	0.766667	0.766667	0.7375	10.188653	

```

Y_pred_nb = np.around(Y_pred_nb)
print(metrics.classification_report(Y_test,Y_pred_nb))

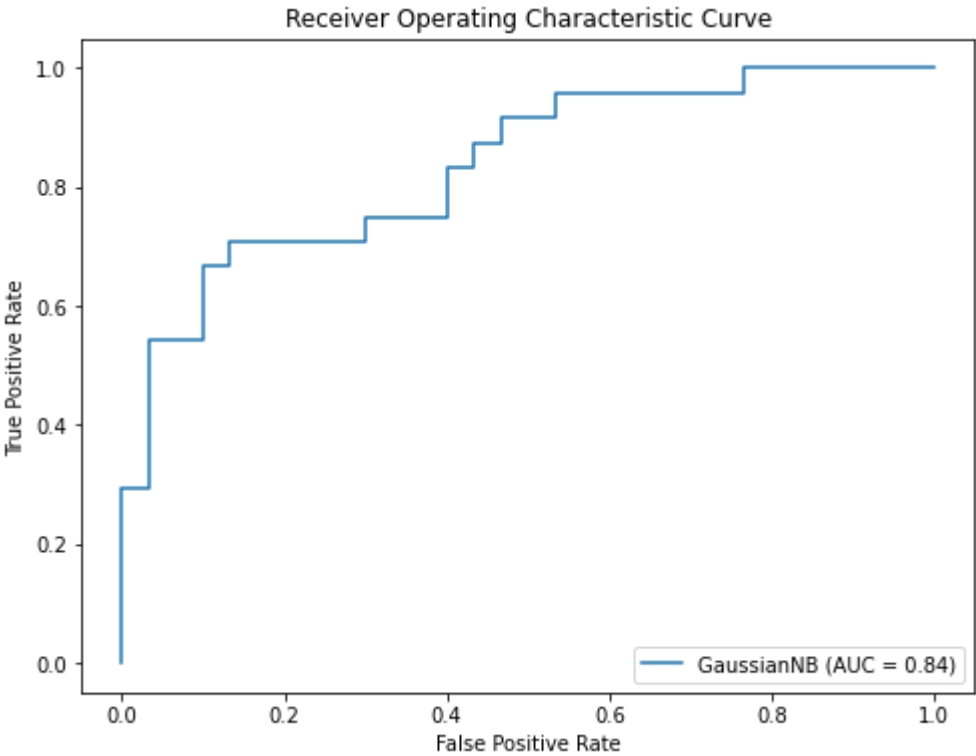
```

	precision	recall	f1-score	support
1	0.77	0.77	0.77	30
2	0.71	0.71	0.71	24
accuracy			0.74	54
macro avg	0.74	0.74	0.74	54
weighted avg	0.74	0.74	0.74	54

```

plot_roc_curve(nb_classifier,X_test,Y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
plt.savefig("GNB.png")

```



▼ Final Scores

```

scores = [score_rf,score_knn,score_nb,score_dt]
Models = ["Random Forest Classifier"," K-Nearest Neighbors Classifier","Navie Bayes Classifier","De

for i in range(len(Models)):
    print("The accuracy score achieved using "+Models[i]+" is: "+str(scores[i])+" %")

    The accuracy score achieved using Random Forest Classifier is: 85.19 %
    The accuracy score achieved using K-Nearest Neighbors Classifier is: 64.81 %

```