Project team's ID : " PPTID-CDS-FEB-24-1839"

Project : PRCP-1016-HeartDieseasePred.

INTRODUCTOIN

Predicting and diagnosing heart disease is the biggest challenge in the medical industry and relies on factors such as the physical examination, symptoms and signs of the patient.

Factors that influence heart disease are body cholesterol levels, smoking habit and obesity, family history of illnesses, blood pressure, and work environment. Machine learning algorithms play an essential and precise role in the prediction of heart disease.

"Advances in technology allow machine language to combine with Big Data tools to manage unstructured and exponentially growing data. Heart disease is seen as the world's deadliest disease of human life. In particular, in this type of disease, the heart is not able to push the required amount of blood to the remaining organs of the human body to perform regular functions."

Heart disease can be predicted based on various symptoms such as age, gender, heart rate, etc. and reduces the death rate of heart patients.

Due to the increasing use of technology and data collection, we can now predict heart disease using machine learning algorithms.

Problem Statement

Task 1 :- Prepare a complete data analysis report on the given data.

Task 2 :- Create a model predicting potential Heart Diseases in people using Machine Learning algorithms.

Task3 :- Suggestions to the Hospital to awake the predictions of heart diseases prevent life threats.

Dataset

1. There are 14 columns in the dataset, where the patient_id column is a unique and random identifier. The remaining 13 features are described in the section below.
2. • slope_of_peak_exercise_st_segment (type: int): the slope of the peak exercise ST segment, an electrocardiography read out indicating quality of blood flow to the heart
3. • thal (type: categorical): results of thallium stress test measuring blood flow to the heart, with possible values normal, fixed_defect, reversible_defect
4. • resting_blood_pressure (type: int): resting blood pressure

5. • chest_pain_type (type: int): chest pain type (4 values)
6. • num_major_vessels (type: int): number of major vessels (0-3) colored by flourosopy
7. • fasting_blood_sugar_gt_120_mg_per_dl (type: binary): fasting blood sugar > 120 mg/dl
8. • resting_ekg_results (type: int): resting electrocardiographic results (values 0,1,2)
9. • serum_cholesterol_mg_per_dl (type: int): serum cholestoral in mg/dl
10. • oldpeak_eq_st_depression (type: float): oldpeak = ST depression induced by exercise relative to rest, a measure of abnormality in electrocardiograms
11. • sex (type: binary): 0: female, 1: male
12. • age (type: int): age in years
13. • max_heart_rate_achieved (type: int): maximum heart rate achieved (beats per minute)
14. • exercise_induced_angina (type: binary): exercise-induced chest pain (0: False, 1: True) Mod

Importing some necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

```

```
data1=pd.read_csv('/content/values.csv')
data2=pd.read_csv('/content/labels.csv')

print(data1.columns)

print(data2.columns)
```

```
Index(['patient_id', 'slope_of_peak_exercise_st_segment', 'thal',
       'resting_blood_pressure', 'chest_pain_type', 'num_major_vessels',
       'fasting_blood_sugar_gt_120_mg_per_dl', 'resting_ekg_results',
       'serum_cholesterol_mg_per_dl', 'oldpeak_eq_st_depression', 'sex', 'age',
       'max_heart_rate_achieved', 'exercise_induced_angina'],
      dtype='object')
Index(['patient_id', 'heart_disease_present'], dtype='object')
```

Merging 2 Data Sets Using 'patient_id'

```
data=pd.merge(data1,data2,on='patient_id' ,how='inner')
data
```

	patient_id	slope_of_peak_exercise_st_segment	thal	resting_blood_press
0	0z64un	1	normal	
1	ryoo3j	2	normal	
2	yt1s1x	1	normal	
3	l2xjde	1	reversible_defect	
4	oyt4ek	3	reversible_defect	
...	...	...	...	
175	5qfar3	2	reversible_defect	
176	2s2b1f	2	normal	
177	nsd00i	2	reversible_defect	
178	0xw93k	1	normal	
179	2nx10r	1	normal	

180 rows × 15 columns

Next steps: Generate code with data View recommended plots

Basic Checks

```
1 data.columns
```

```
Index(['patient_id', 'slope_of_peak_exercise_st_segment', 'thal',
       'resting_blood_pressure', 'chest_pain_type', 'num_major_vessels',
       'fasting_blood_sugar_gt_120_mg_per_dl', 'resting_ekg_results',
       'serum_cholesterol_mg_per_dl', 'oldpeak_eq_st_depression', 'sex', 'age',
       'max_heart_rate_achieved', 'exercise_induced_angina',
       'heart_disease_present'],
      dtype='object')
```

```
1 data.drop('patient_id',axis=1,inplace=True)
```

Double-click (or enter) to edit

```
1 data.head()
```

	slope_of_peak_exercise_st_segment	thal	resting_blood_pressure	chest_pai
0	1	normal	128	
1	2	normal	110	
2	1	normal	125	
3	1	reversible_defect	152	
4	3	reversible_defect	178	

Next steps: Generate code with data View recommended plots

```
1 data.shape
```

```
(180, 14)
```

Here data set have 180 rows and 14 columns

```
1 data.tail()
```

	slope_of_peak_exercise_st_segment	thal	resting_blood_pressure	chest_p
175	2	reversible_defect	125	
176	2	normal	180	
177	2	reversible_defect	125	
178	1	normal	124	
179	1	normal	160	

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 180 entries, 0 to 179
Data columns (total 14 columns):
 #   Column                               Non-Null Count  Dtype  
---  ------                               --------------  -----  
 0   slope_of_peak_exercise_st_segment    180 non-null    int64  
 1   thal                                 180 non-null    object 
 2   resting_blood_pressure               180 non-null    int64  
 3   chest_pain_type                      180 non-null    int64  
 4   num_major_vessels                    180 non-null    int64  
 5   fasting_blood_sugar_gt_120_mg_per_dl 180 non-null    int64  
 6   resting_ekg_results                  180 non-null    int64  
 7   serum_cholesterol_mg_per_dl          180 non-null    int64  
 8   oldpeak_eq_st_depression             180 non-null    float64
 9   sex                                  180 non-null    int64  
 10  age                                  180 non-null    int64  
 11  max_heart_rate_achieved              180 non-null    int64  
 12  exercise_induced_angina              180 non-null    int64  
 13  heart_disease_present                180 non-null    int64  
dtypes: float64(1), int64(12), object(1)
memory usage: 21.1+ KB
```

1. slope_of_peak_exercise_st_segment (int64): The slope of the peak exercise ST segment in the electrocardiogram.
2. thal (object): A categorical variable representing a type of thalassemia.
3. resting_blood_pressure (int64): The resting blood pressure of the individual.
4. chest_pain_type (int64): The type of chest pain experienced by the individual.
5. num_major_vessels (int64): The number of major vessels colored by fluoroscopy.
6. fasting_blood_sugar_gt_120_mg_per_dl (int64): Presence of fasting blood sugar greater than 120 mg/dl.
7. resting_ekg_results (int64): The result of the resting electrocardiographic measurement.
8. serum_cholesterol_mg_per_dl (int64): The serum cholesterol level in mg/dl.
9. oldpeak_eq_st_depression (float64): ST depression induced by exercise relative to rest.
10. sex (int64): Gender of the individual (assuming 0 or 1 for male and female).
11. age (int64): Age of the individual.
12. max_heart_rate_achieved (int64): Maximum heart rate achieved during exercise.
13. exercise_induced_angina (int64): Presence of exercise-induced angina.

14. heart_disease_present (int64): The target variable indicating the presence or absence of heart disease.

The data seems to be related to heart health, with features that could be used to predict the presence of heart disease. The next steps would typically involve exploratory data analysis, data preprocessing, and then applying machine learning models to predict or analyze heart disease based on these features.

```
1 data.duplicated().sum()
```

```
0
```

'0' duplicate rows present in a DataFrame or their is no duplicates found in dataset

```
1 data.isnull().sum()
```

```
slope_of_peak_exercise_st_segment       0
thal                                    0
resting_blood_pressure                  0
chest_pain_type                         0
num_major_vessels                       0
fasting_blood_sugar_gt_120_mg_per_dl    0
resting_ekg_results                     0
serum_cholesterol_mg_per_dl             0
oldpeak_eq_st_depression                0
sex                                     0
age                                     0
max_heart_rate_achieved                 0
exercise_induced_angina                 0
heart_disease_present                   0
dtype: int64
```

So,we have no missing values

```
1 data.describe().T
```

	count	mean	std	min	25%	50%	
slope_of_peak_exercise_st_segment	180.0	1.550000	0.618838	1.0	1.00	1.0	
resting_blood_pressure	180.0	131.311111	17.010443	94.0	120.00	130.0	1
chest_pain_type	180.0	3.155556	0.938454	1.0	3.00	3.0	
num_major_vessels	180.0	0.694444	0.969347	0.0	0.00	0.0	
fasting_blood_sugar_gt_120_mg_per_dl	180.0	0.161111	0.368659	0.0	0.00	0.0	
resting_ekg_results	180.0	1.050000	0.998742	0.0	0.00	2.0	
serum_cholesterol_mg_per_dl	180.0	249.211111	52.717969	126.0	213.75	245.5	2
oldpeak_eq_st_depression	180.0	1.010000	1.121357	0.0	0.00	0.8	
sex	180.0	0.688889	0.464239	0.0	0.00	1.0	
age	180.0	54.811111	9.334737	29.0	48.00	55.0	
max_heart_rate_achieved	180.0	149.483333	22.063513	96.0	132.00	152.0	1
exercise_induced_angina	180.0	0.316667	0.466474	0.0	0.00	0.0	
heart_disease_present	180.0	0.444444	0.498290	0.0	0.00	0.0	

```
1 data.columns
```

```
Index(['slope_of_peak_exercise_st_segment', 'thal', 'resting_blood_pressure',
       'chest_pain_type', 'num_major_vessels',
```

```
       'fasting_blood_sugar_gt_120_mg_per_dl', 'resting_ekg_results',
       'serum_cholesterol_mg_per_dl', 'oldpeak_eq_st_depression', 'sex', 'age',
       'max_heart_rate_achieved', 'exercise_induced_angina',
       'heart_disease_present'],
      dtype='object')
```

```
data_desc=data[['slope_of_peak_exercise_st_segment','thal','chest_pain_type',
                'num_major_vessels','fasting_blood_sugar_gt_120_mg_per_dl',
                'resting_ekg_results','sex']]
```

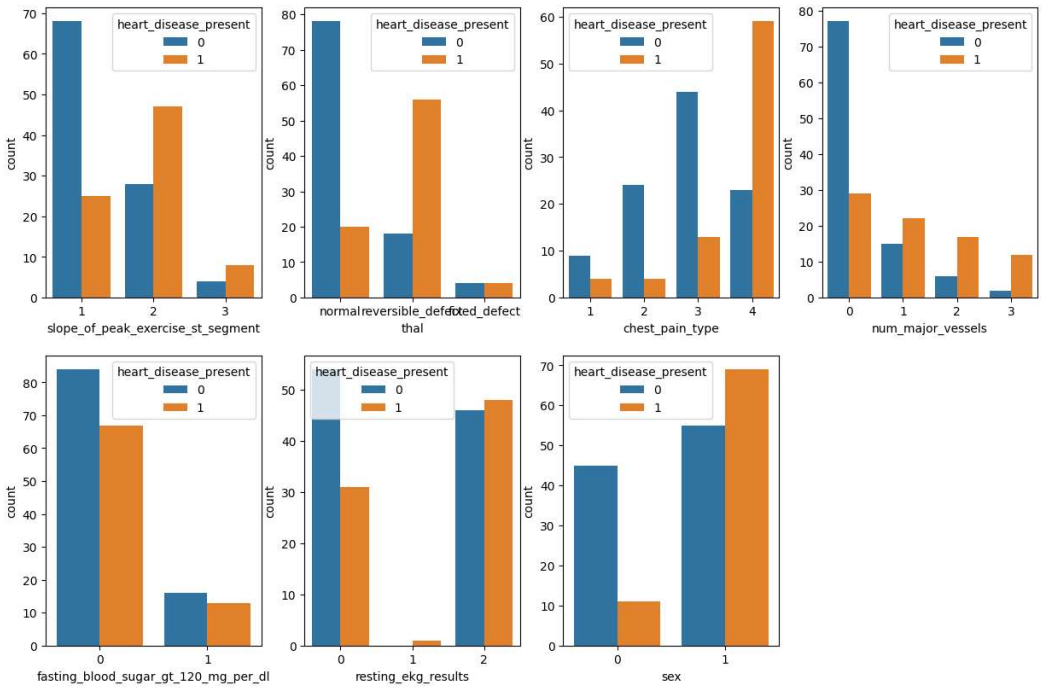
Exploratory Data Analysis(EDA)

EDA stands for Exploratory Data Analysis. It's a crucial first step in data science projects that involves investigating and understanding a dataset before jumping into formal analysis or modeling.

EDA uses statistical and visualization techniques to reveal patterns and trends within the data.

```
y = data["heart_disease_present"]
```

```
plt.figure(figsize=(15,15))
p=1

for i in data_desc:
  if p<=12:
    ax=plt.subplot(3,4,p)
    sns.countplot(x=data[i],hue=data.heart_disease_present)
  p+=1
```

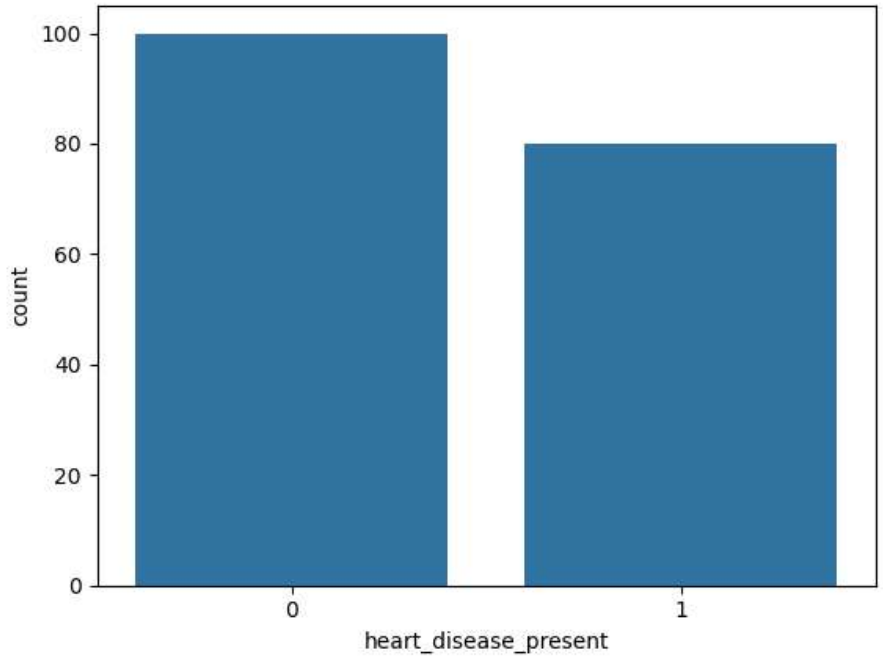


The count plots for each categorical variable (data_desc) give insights into the distribution of different categories within each variable. This can help understand the prevalence of certain characteristics within the dataset.

By using the hue parameter to differentiate count plots based on the heart_disease_present variable, the visualizations allow for understanding how each categorical variable relates to the presence or absence of heart disease.

```
sns.countplot(x='heart_disease_present', data=data)
target_temp = data.heart_disease_present.value_counts()
print(target_temp)
plt.show()
```

```
0    100
1     80
Name: heart_disease_present, dtype: int64
```

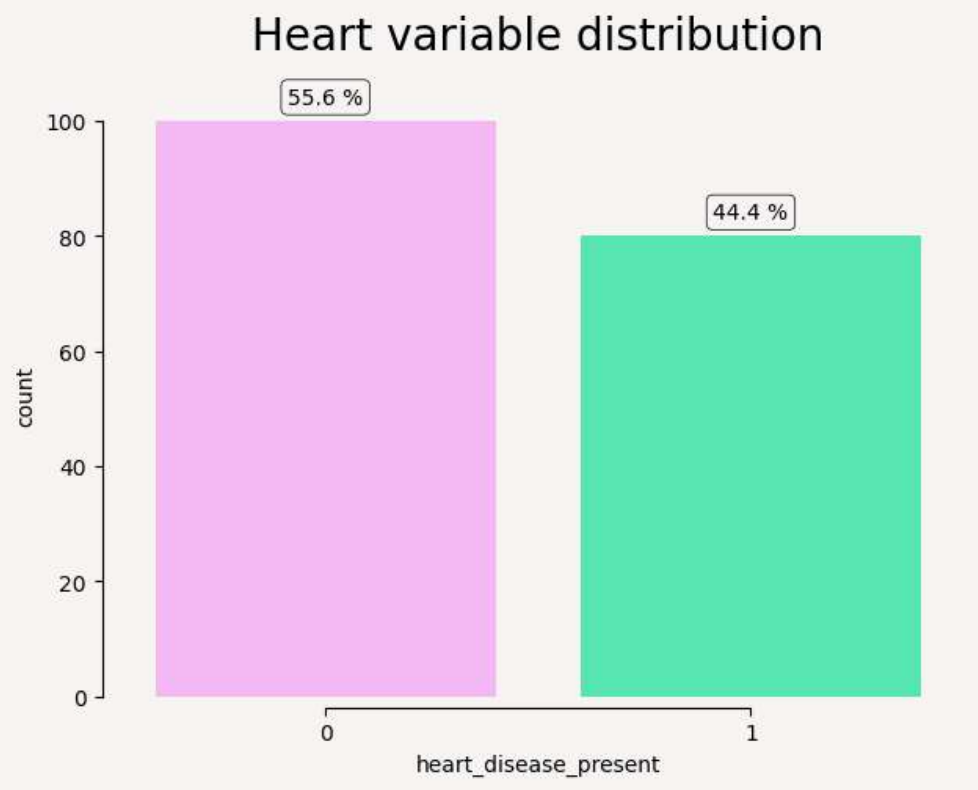


Percentage of patient with or without heart problems in the given dataset

```
print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/180,2)))
print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/180,2)))
```

```
Percentage of patience without heart problems: 55.56
Percentage of patience with heart problems: 44.44
```

```
mypal= ['#FC05FB', '#FEAEFE', '#FCD2FC','#F3FEFA', '#B4FFE4','#3FFEBA']

plt.figure(figsize=(7, 5),facecolor='#F6F5F4')
total = float(len(data))
ax = sns.countplot(x=data['heart_disease_present'], palette=mypal[1::4])
ax.set_facecolor('#F6F5F4')

for p in ax.patches:

    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2.,height + 3,'{:1.1f} %'.format((height/total)*100), ha="center",
           bbox=dict(facecolor='none', edgecolor='black', boxstyle='round', linewidth=0.5))

ax.set_title('Heart variable distribution', fontsize=20, y=1.05)
sns.despine(right=True)
sns.despine(offset=5, trim=True)
```

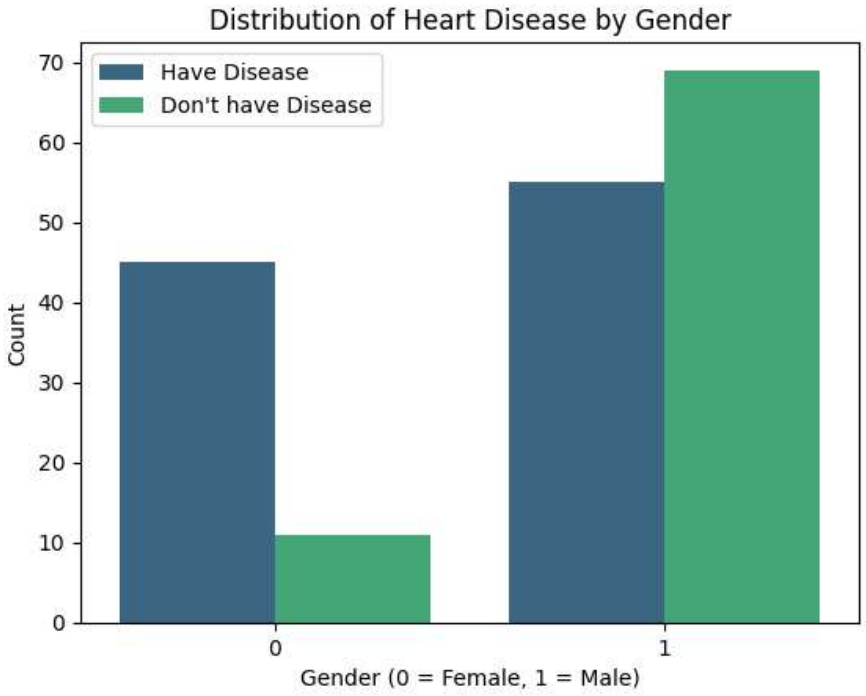


Heart Disease frequency for sex (where 0 is female and 1 is male) and "blue" is have heart disease and "green" is don't have heart disease

```
1 data["sex"].unique()
```

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

```
1 sns.countplot(x='sex', data=data, hue='heart_disease_present', palette='viridis')
2 plt.title('Distribution of Heart Disease by Gender')
3 plt.ylabel('Count')
4 plt.xticks(rotation=0)
5 plt.legend(["Have Disease","Don't have Disease"])
6
7 plt.xlabel('Gender (0 = Female, 1 = Male)')
8 plt.show()
```



Insights

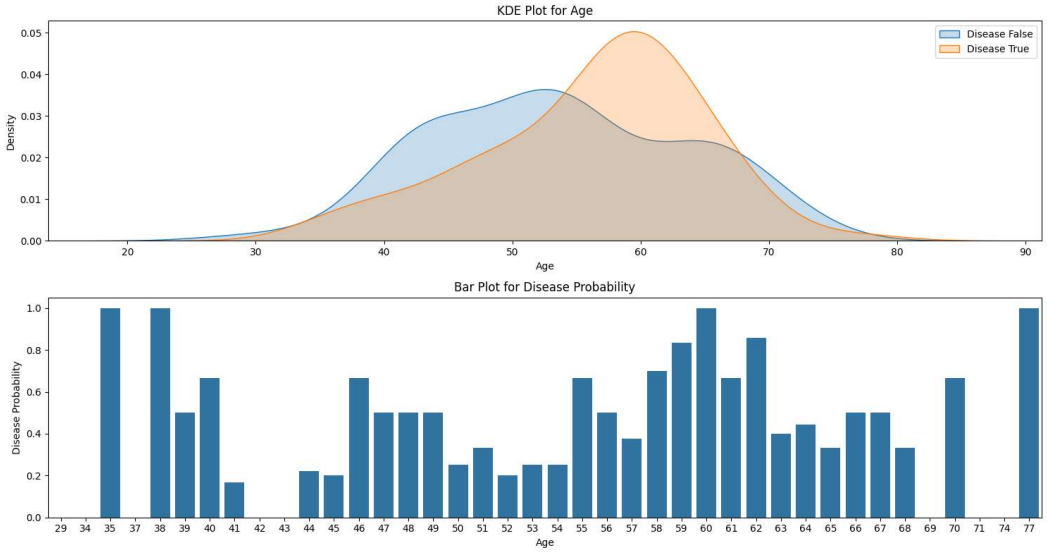
It show that heart disease is more prevalent in males than in females.

Male: 70 individuals have heart disease and 52 don't have it. Female: 45 individuals have heart disease and 10 don't have it.

Double-click (or enter) to edit

Plot accodring to AGE group

```
def plotAge(data):
    # Create subplots
    fig, axes = plt.subplots(2, 1, figsize=(15, 8))

    # KDE plot for 'age' variable
    sns.kdeplot(data[data['heart_disease_present'] == 0]['age'], label='Disease False', shade=True, ax=axes[0])
    sns.kdeplot(data[data['heart_disease_present'] == 1]['age'], label='Disease True', shade=True, ax=axes[0])
    axes[0].set(xlabel='Age', ylabel='Density')
    axes[0].set_title('KDE Plot for Age')
    axes[0].legend()

    # Bar plot for disease probability
    avg = data.groupby('age')['heart_disease_present'].mean().reset_index()
    sns.barplot(x='age', y='heart_disease_present', data=avg, ax=axes[1])
    axes[1].set(xlabel='Age', ylabel='Disease Probability')
    axes[1].set_title('Bar Plot for Disease Probability')

    # Adjust layout
    plt.tight_layout()

    # Show the plot
    plt.show()



# Call the function with your dataset
plotAge(data)

```



According to the data, the probability of heart disease is higher at ages 35 and 38 as well as ages 60 , 62 and 77.

Percentage of female and male patients

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

```
Percentage of Female Patients:31.11%
Percentage of Male Patients:68.89%
```

Separating categorical and continuous variables

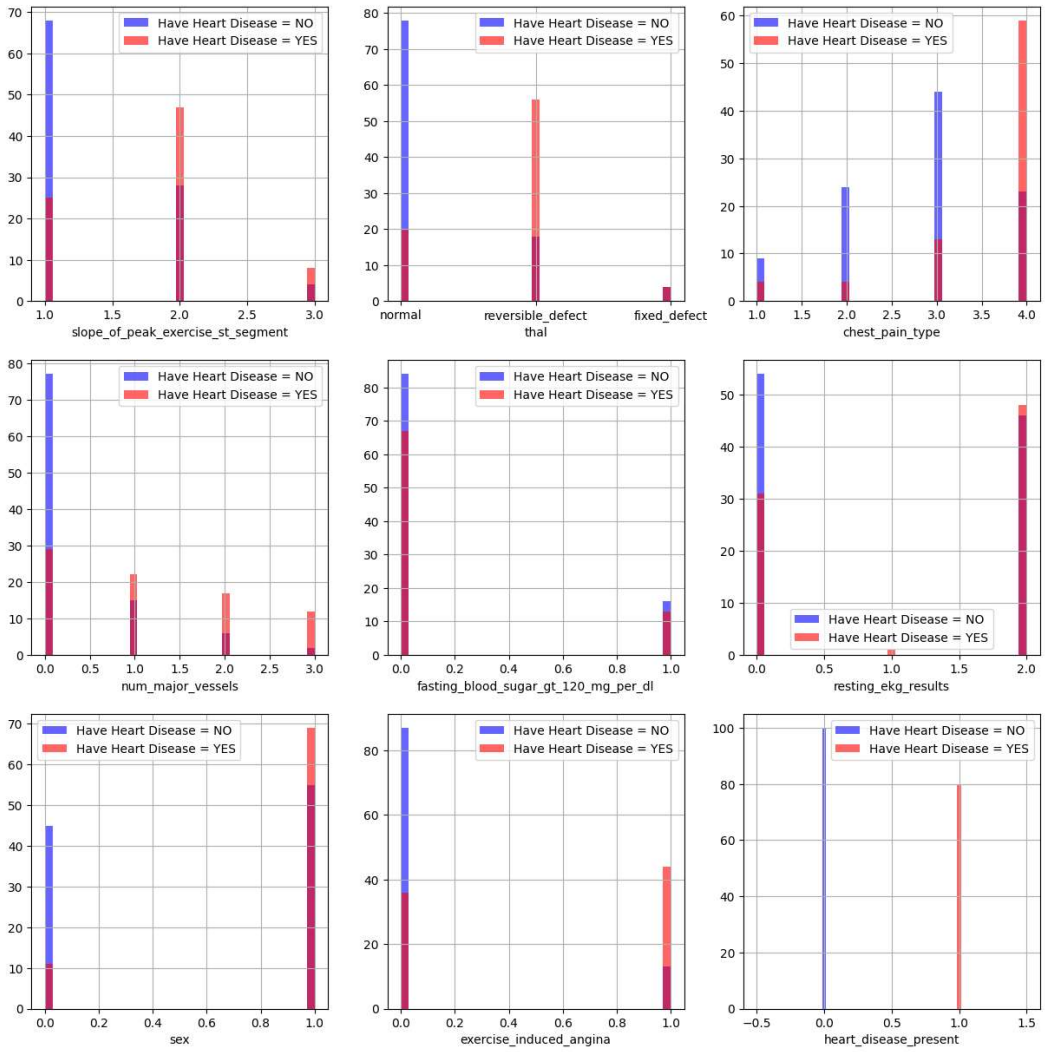
```
categorical_val = []
continous_val = []
for column in data.columns:
    print('==============================')
    print(f"{column} : {data[column].unique()}")
    if len(data[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
==============================
slope_of_peak_exercise_st_segment : [1 2 3]
==============================
thal : ['normal' 'reversible_defect' 'fixed_defect']
==============================
resting_blood_pressure : [128 110 125 152 178 130 150 170 120 140 138 144 136 160 108 106 156 180
 112 122 124 135 105 115 126 172 145 118 134 100 155 132 102  94 117 142]
==============================
chest_pain_type : [2 3 4 1]
==============================
```

```
num_major_vessels : [0 3 2 1]
==============================
fasting_blood_sugar_gt_120_mg_per_dl : [0 1]
==============================
resting_ekg_results : [2 0 1]
==============================
serum_cholesterol_mg_per_dl : [308 214 304 223 270 180 258 276 326 219 302 226 335 236 231 200 234 253
 204 319 233 228 245 211 303 205 185 175 225 203 325 230 222 126 209 269
 255 243 252 265 417 267 261 149 281 311 315 330 256 239 295 197 564 305
 283 160 254 282 322 250 188 220 199 215 218 196 266 229 259 268 177 168
 262 271 299 293 141 277 212 321 294 313 232 289 213 274 263 244 298 172
 353 210 192 246 286 360 174 227 248 224 300 235 217 193 167 216 195 309
 273 198 290 275 206 164 207 249 327 201]
==============================
oldpeak_eq_st_depression : [0.  1.6 4.2 2.6 0.6 3.4 0.4 0.2 3.8 0.9 1.4 0.1 0.3 2.3 1.5 3.  2.  1.
 3.1 2.5 0.8 2.4 1.8 1.9 2.8 1.2 0.5 2.2 1.3 1.1 0.7 3.2 5.6 6.2]
==============================
sex : [1 0]
==============================
age : [45 54 77 40 59 42 60 57 50 66 64 38 29 58 71 52 67 70 68 51 41 65 53 48
 62 74 61 63 46 43 56 44 35 55 49 47 37 69 39 34]
==============================
max_heart_rate_achieved : [170 158 162 181 145 150 157 112 140 151 178 152 182 126 175 144 202 147
 142 138 143 115 159 184 155 123 168 114 154 165 186 173 163 121 161 137
 172 130 167 141 166 125 103 120 132 169 179  99 177 160 156 109 139 134
 113 149 174 131 148 153 133 122 105 106 192 108  96 171 180 188 111  97
 117]
==============================
exercise_induced_angina : [0 1]
==============================
heart_disease_present : [0 1]
```

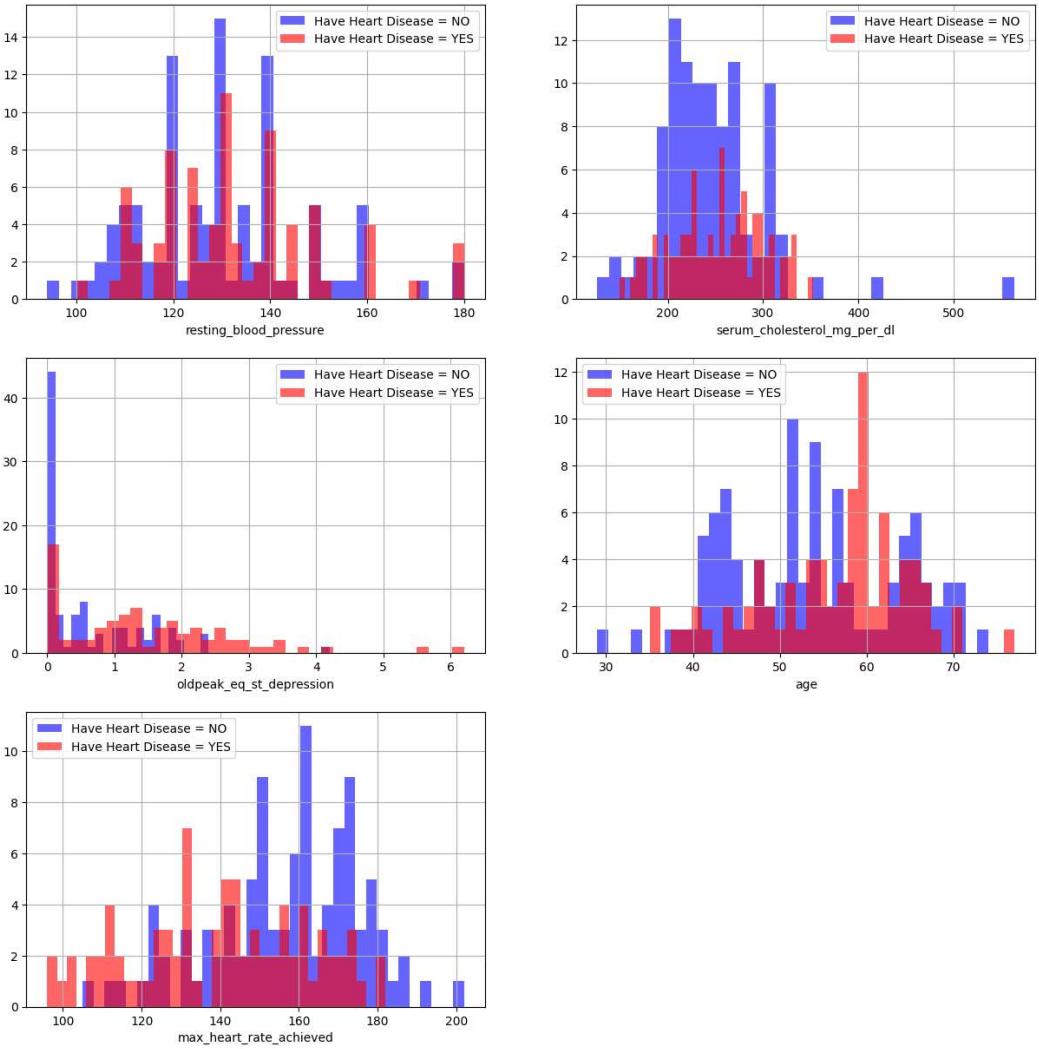
Overall, the code snippet is creating a visualization that compares the distribution of each categorical feature in the heart disease dataset between patients with and without heart disease.

```
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    data[data["heart_disease_present"] == 0][column].hist(bins=35, color='blue', label='Have Heart Disease = NO', alpha=0.6)
    data[data["heart_disease_present"] == 1][column].hist(bins=35, color='red', label='Have Heart Disease = YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```



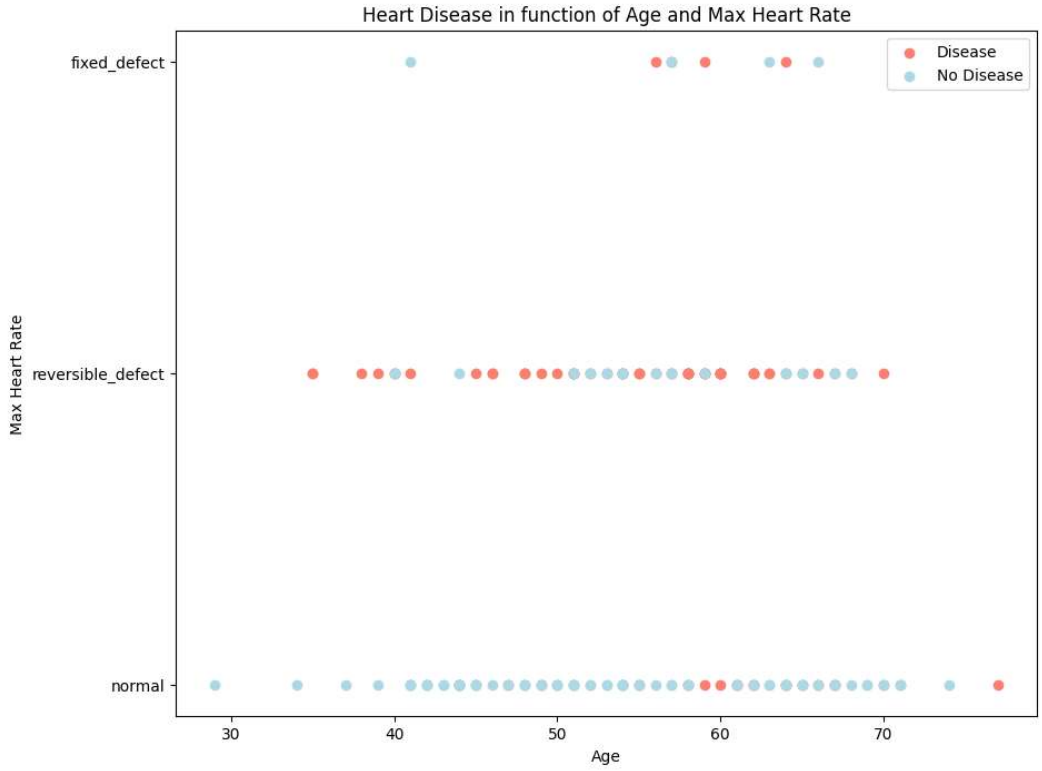
Distribution of each countinous feature in the heart disease dataset between patients with and without heart disease

```
1
2 plt.figure(figsize=(15, 15))
3
4 for i, column in enumerate(continous_val, 1):
5     plt.subplot(3, 2, i)
6     data[data["heart_disease_present"] == 0][column].hist(bins=35, color='blue', label='Have Heart Disease = NO', alpha=0.6)
7     data[data["heart_disease_present"] == 1][column].hist(bins=35, color='red', label='Have Heart Disease = YES', alpha=0.6)
8     plt.legend()
9     plt.xlabel(column)
```



A scatter plot to visualize the relationship between age and max heart rate for people with and without heart disease.

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(data.age[data.heart_disease_present==1],
            data.thal[data.heart_disease_present==1],
            c="salmon")

# Scatter with negative examples
plt.scatter(data.age[data.heart_disease_present==0],
            data.thal[data.heart_disease_present==0],
            c="lightblue")

# Add some helpful info
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);
```

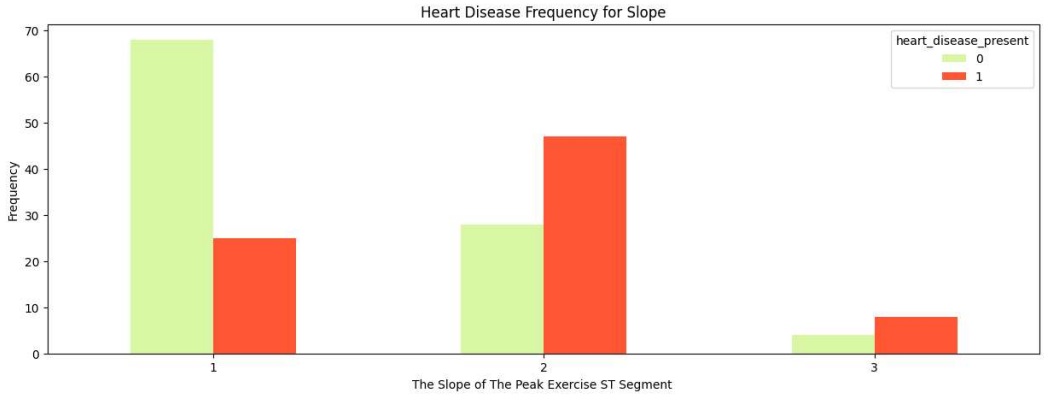


Heart Disease Frequency for Slope_of_peak_exercise

```
data["slope_of_peak_exercise_st_segment"].unique()
```

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

```
pd.crosstab(data.slope_of_peak_exercise_st_segment,data.heart_disease_present).plot(kind="bar",figsize=(15,5),color=['#DAF7A6','#FF5733' ])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



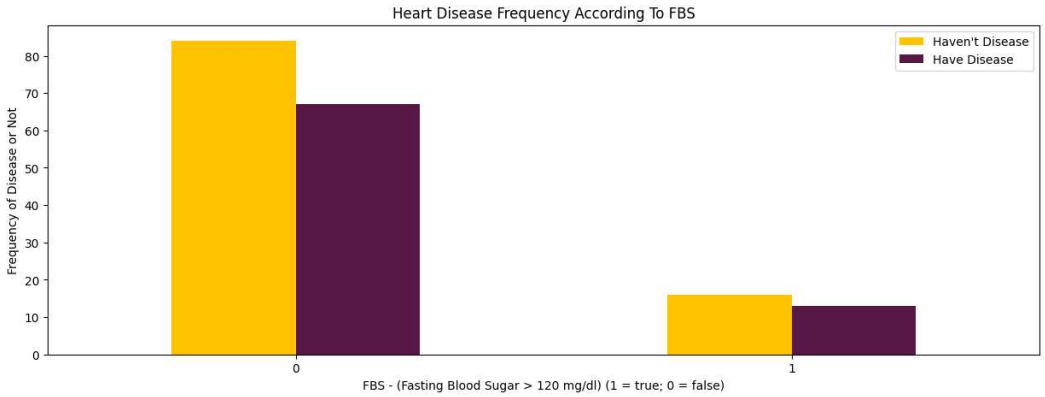
#Slope '1' causes heart pain much more than Slope '2' and '3'

Heart disease Frequency according to Fasting Blood sugar

```
data["fasting_blood_sugar_gt_120_mg_per_dl"].unique()
```

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

```
pd.crosstab(data.fasting_blood_sugar_gt_120_mg_per_dl,data.heart_disease_present).plot(kind="bar",figsize=(15,5),color=['#FFC300','#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



Heart Disease Frequency According To Chest Pain Type

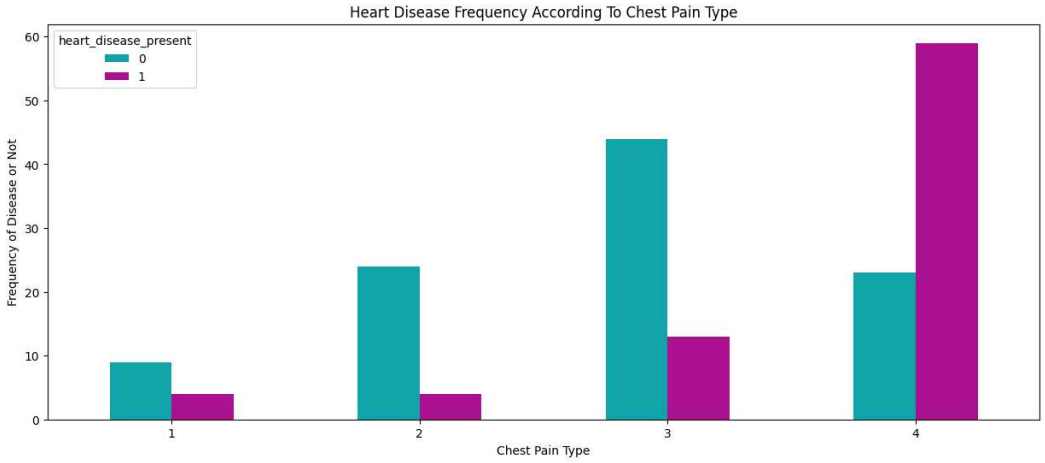
Analysing the chest pain (4 types of chest pain)

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

```
1 data["chest_pain_type"].unique()
```

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

```
1 pd.crosstab(data.chest_pain_type,data.heart_disease_present).plot(kind="bar",figsize=(15,6),color=['#11A5AA','#AA1190' ])
2 plt.title('Heart Disease Frequency According To Chest Pain Type')
3 plt.xlabel('Chest Pain Type')
4 plt.xticks(rotation = 0)
5 plt.ylabel('Frequency of Disease or Not')
6 plt.show()
```

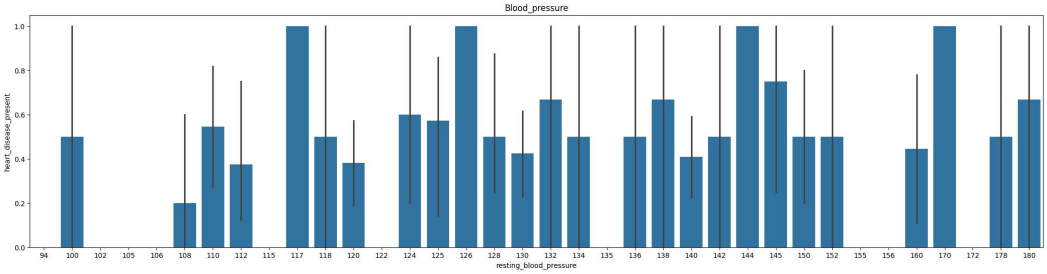


Analysing The person's resting blood pressure (mm Hg on admission to the hospital)

```
1
2 data["resting_blood_pressure"].unique()
```

```
array([128, 110, 125, 152, 178, 130, 150, 170, 120, 140, 138, 144, 136,
       160, 108, 106, 156, 180, 112, 122, 124, 135, 105, 115, 126, 172,
       145, 118, 134, 100, 155, 132, 102,  94, 117, 142])
```

```
plt.figure(figsize=(26, 6))
sns.barplot(x=data["resting_blood_pressure"], y="heart_disease_present", data=data)
plt.title('Blood_pressure')
plt.xlabel('resting_blood_pressure')
plt.xticks(rotation = 0)
plt.ylabel('heart_disease_present')
plt.show()
```

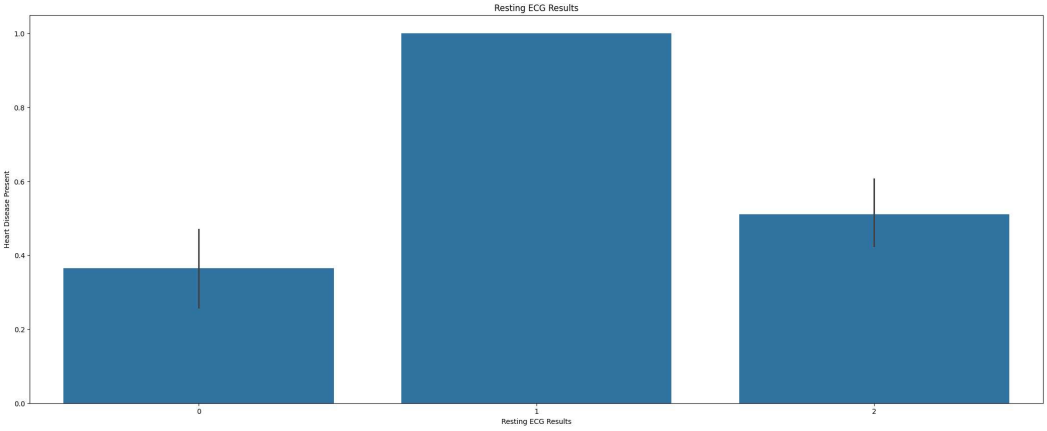


Analysing the Resting electrocardiographic measurement (0 = normal, 1 = having ST-T wave abnormality, 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)

```
data["resting_ekg_results"].unique()
```

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

```
plt.figure(figsize=(26, 10))
sns.barplot(x=data["resting_ekg_results"], y="heart_disease_present", data=data)
plt.title('Resting ECG Results')
plt.xlabel('Resting ECG Results')
plt.ylabel('Heart Disease Present')
plt.xticks(rotation=0)
plt.show()
```



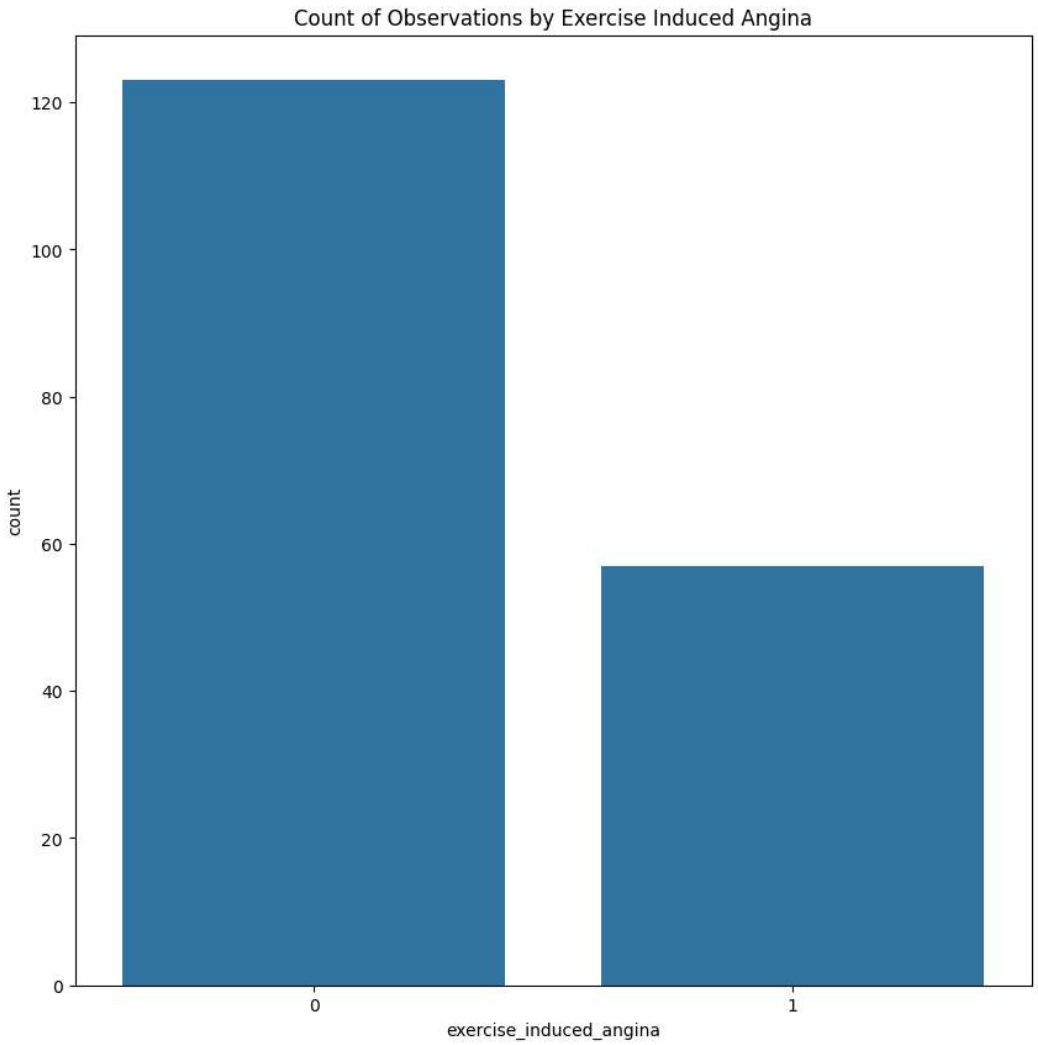
people with restecg '1' and '2' are much more likely to have a heart disease than with restecg '0'

Analysing Exercise induced angina (1 = yes; 0 = no)

```
1 data["exercise_induced_angina"].unique()
```

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

```
1 plt.figure(figsize=(10, 10))
2 sns.countplot(x="exercise_induced_angina", data=data)
3 plt.title('Count of Observations by Exercise Induced Angina')
4 plt.show()
```



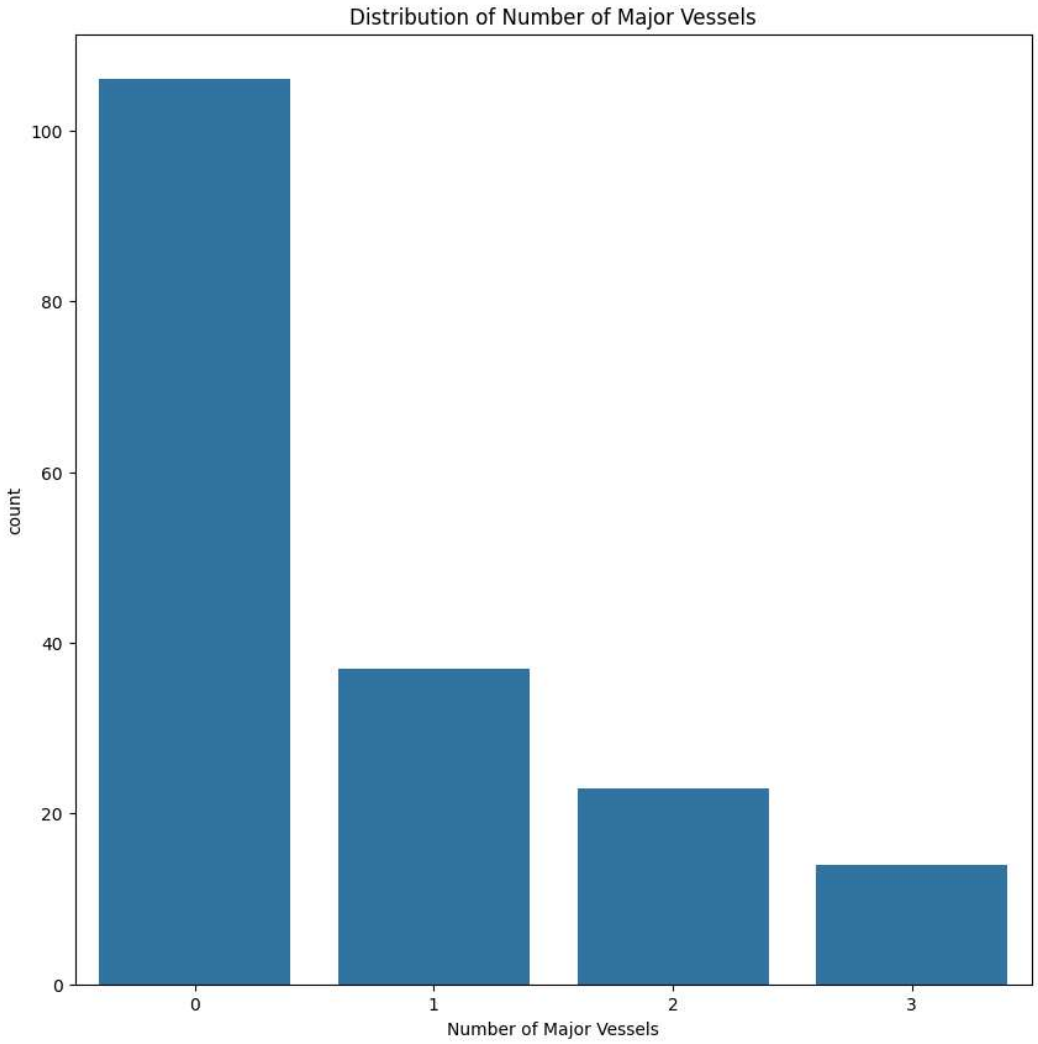
#People with exercise_induced_angina=1 are much less likely to have heart problems.

Analysing number of major vessels (0-3) colored by flourosopy

```
1 data["num_major_vessels"].unique()
```

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

```
1 plt.figure(figsize=(10, 10))
2 sns.countplot(data=data, x='num_major_vessels')
3 plt.title('Distribution of Number of Major Vessels')
4 plt.xlabel('Number of Major Vessels')
5 plt.show()
```



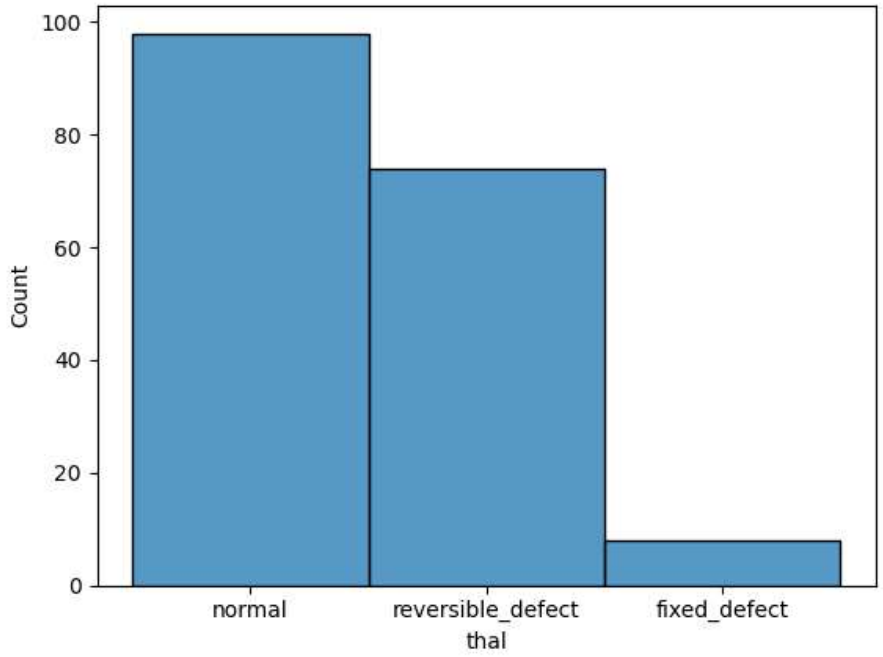
Analysing A blood disorder called thalassemia (normal; fixed defect;reversable defect)

```
1 data["thal"].unique()
```

```
array(['normal', 'reversible_defect', 'fixed_defect'], dtype=object)
```

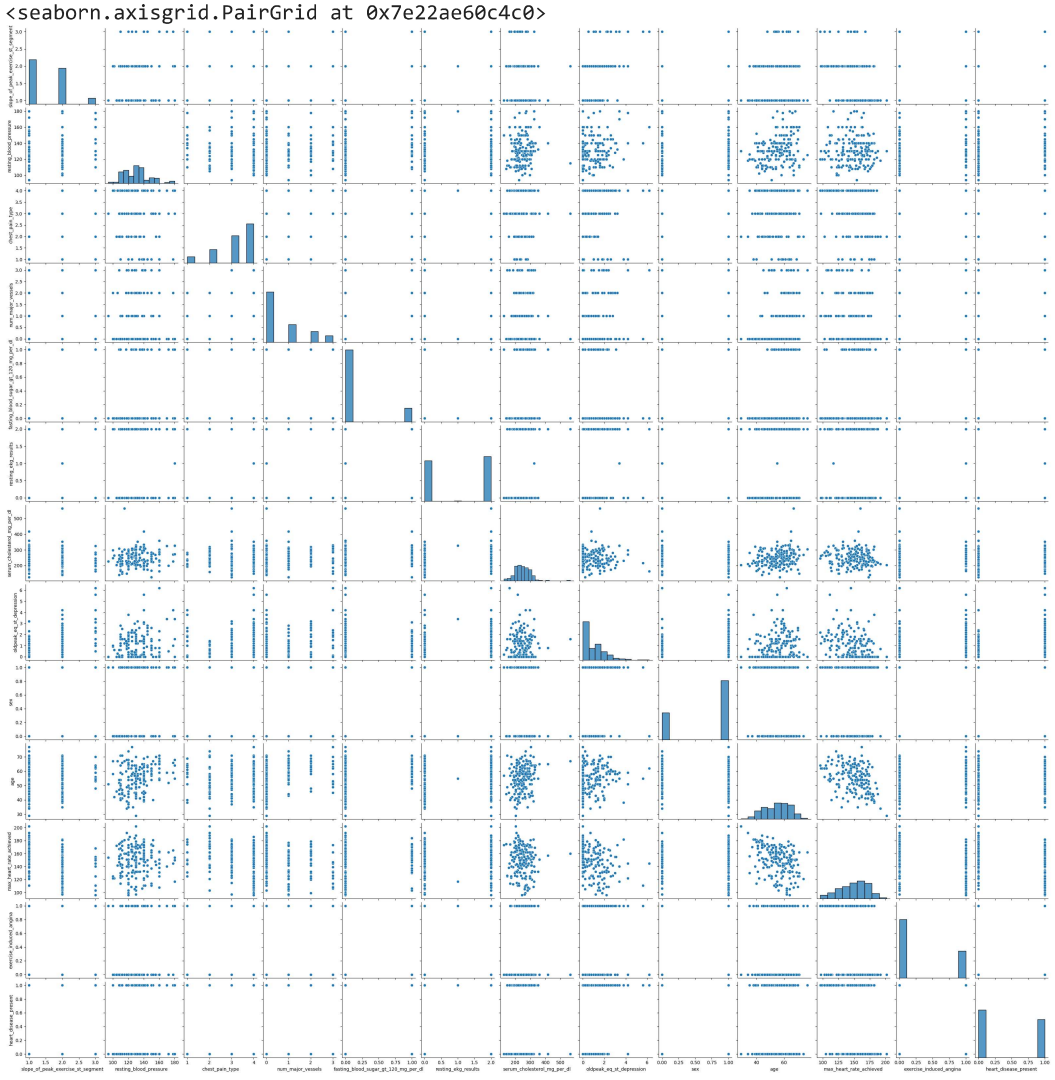
```
1 sns.histplot(data["thal"])
2
```

```
<Axes: xlabel='thal', ylabel='Count'>
```

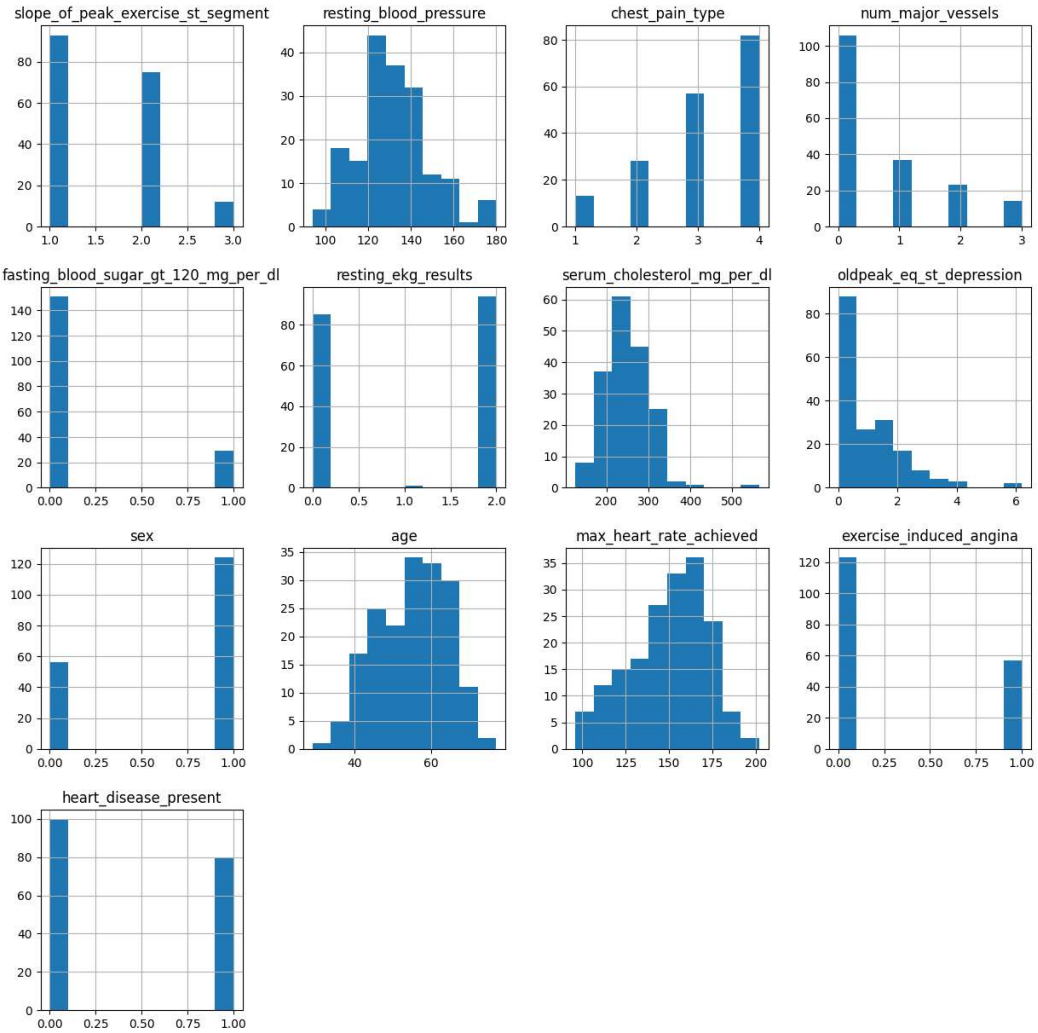



```
1 sns.pairplot(data=data)
```

```
<seaborn.axisgrid.PairGrid at 0x7e22ae60c4c0>
```



```
fig, ax = plt.subplots(figsize=(15, 15))

# Plot histograms for each column of the dataframe
data.hist(ax=ax)

plt.show()
```

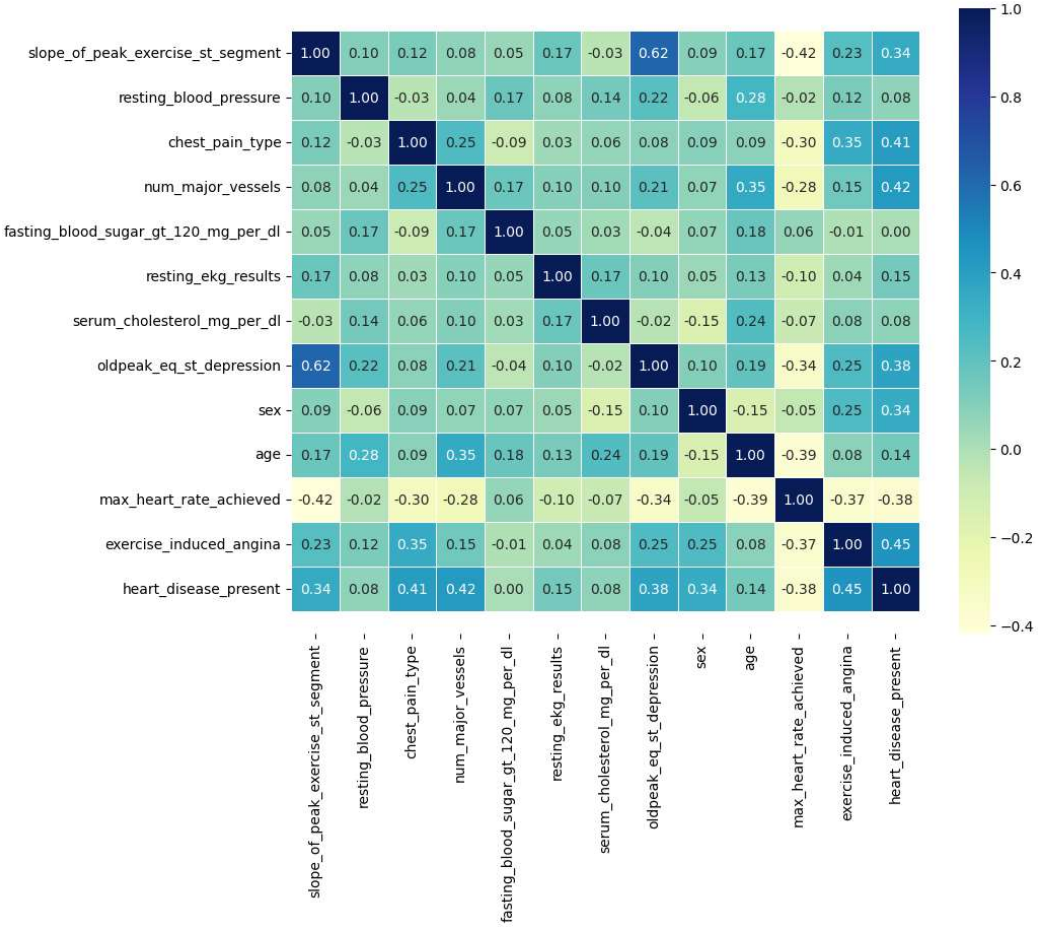


Correlation Matrix

Correlation analysis is a method of statistical evaluation used to study the strength of a relationship between two, numerically measured, continuous variables.

```
# Let's make our correlation matrix a little prettier
corr_matrix = data.corr()
fig, ax = plt.subplots(figsize=(10, 8))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

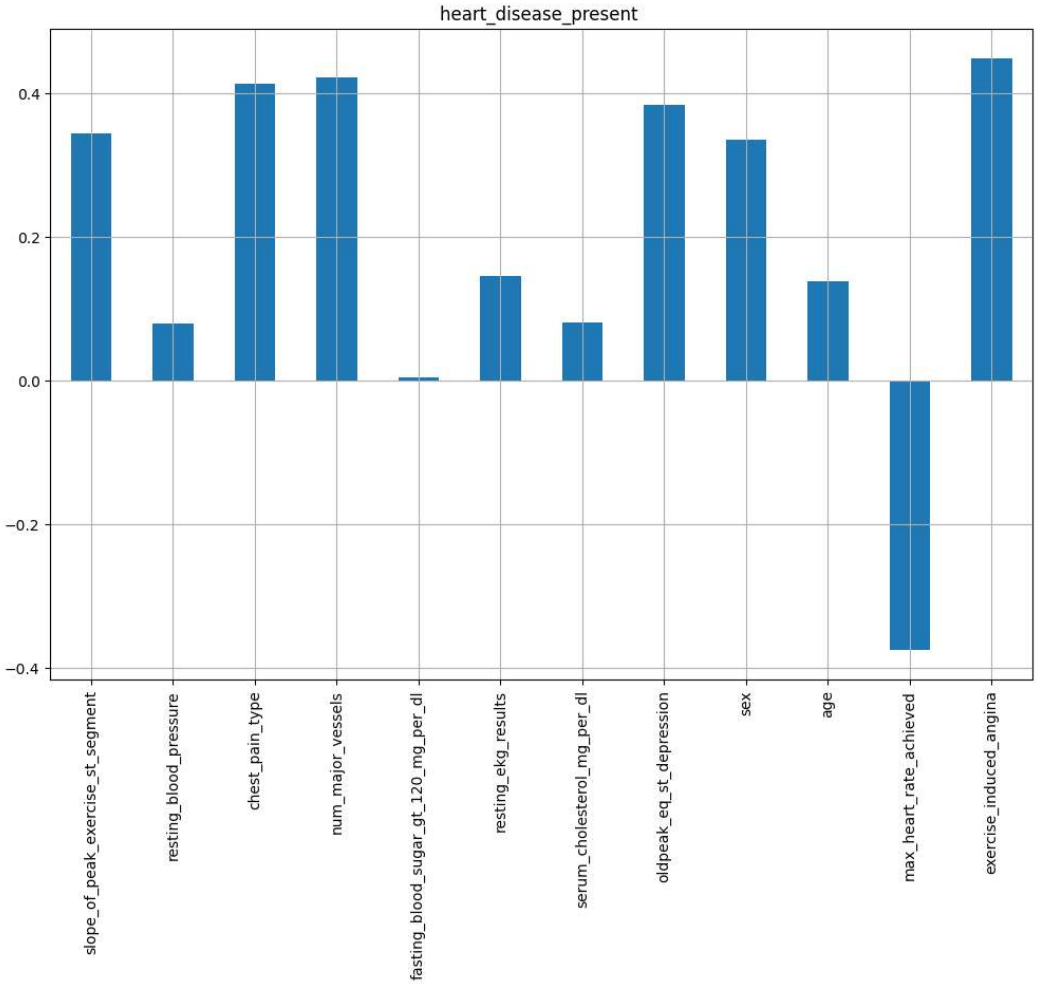
```
(13.5, -0.5)
```



Their is no relationaship between the columns,hence no need to do drop the column

```
data.drop('heart_disease_present', axis=1).corrwith(data.heart_disease_present).plot(kind='bar', grid=True, figsize=(12, 8),
                                                  title="heart_disease_present ")
```

```
<Axes: title={'center': 'heart_disease_present '}>
```




```
1 print(data.corr()["heart_disease_present"].abs().sort_values(ascending=False))
```

```
heart_disease_present                 1.000000
exercise_induced_angina               0.448647
num_major_vessels                     0.421519
chest_pain_type                       0.412829
oldpeak_eq_st_depression              0.382930
max_heart_rate_achieved               0.375352
slope_of_peak_exercise_st_segment     0.344224
sex                                   0.335421
resting_ekg_results                   0.145933
age                                   0.138255
serum_cholesterol_mg_per_dl           0.079775
resting_blood_pressure                0.078506
fasting_blood_sugar_gt_120_mg_per_dl  0.003379
Name: heart_disease_present, dtype: float64
```

Creating Dummy Variables

Data Processing

After exploring the dataset, we can observe that we need to convert some variables to dummy variables and scale all values before training the machine learning models. Since 'cp', 'thal' and 'slope' are categorical variables we'll turn them into dummy variables.

```
a = pd.get_dummies(data['chest_pain_type'], prefix = "cp")
b = pd.get_dummies(data['thal'], prefix = "thal")
c = pd.get_dummies(data['slope_of_peak_exercise_st_segment'], prefix = "slope")
```

```
frames = [data, a, b, c]
data = pd.concat(frames, axis = 1)
data.head()
```

	slope_of_peak_exercise_st_segment	thal	resting_blood_pressure	chest_pai
0	1	normal	128	
1	2	normal	110	
2	1	normal	125	
3	1	reversible_defect	152	
4	3	reversible_defect	178	

5 rows × 24 columns

```
data = data.drop(columns = ['chest_pain_type', 'thal', 'slope_of_peak_exercise_st_segment'])
data.head()
```

	resting_blood_pressure	num_major_vessels	fasting_blood_sugar_gt_120_mg_per_dl	res
0	128	0	0	
1	110	0	0	
2	125	3	0	
3	152	0	0	
4	178	0	0	

5 rows × 21 columns

```
from sklearn.preprocessing import StandardScaler
s_sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'serum_cholesterol_mg_per_dl', 'thal', 'oldpeak_eq_st_depression']
```

Applying Logistic Regression

Now, I will train a machine learning model for the task of heart disease prediction. I will use the logistic regression algorithm as I mentioned at the beginning of the article.

But before training the model I will first define a helper function for printing the classification report of the performance of the machine learning model:

```
# Split the dataset into features (y) and target (x)
X = data.drop('heart_disease_present', axis=1)
y = data.heart_disease_present
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
# Train Accuracy LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)

y_pred=lr_clf.predict(X_test)

accuracy_score(y_test,y_pred)
print("Train Accuracy : {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
Train Accuracy : 84.44%
```

```
# Test Accuracy LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

lr_clf = LogisticRegression()
lr_clf.fit(X_test, y_test)

y_pred=lr_clf.predict(X_test)

accuracy_score(y_test,y_pred)
print("Test Accuracy : {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
Test Accuracy : 86.67%
```

SMOTE LOGISTIC REGRESSION

```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
x_smote,y_smote=sm.fit_resample(X_train,y_train)

y_smote.value_counts()
```

```
0    81
1    81
Name: heart_disease_present, dtype: int64
```

LOGISTIC REGRESSION AFTER SMOTE

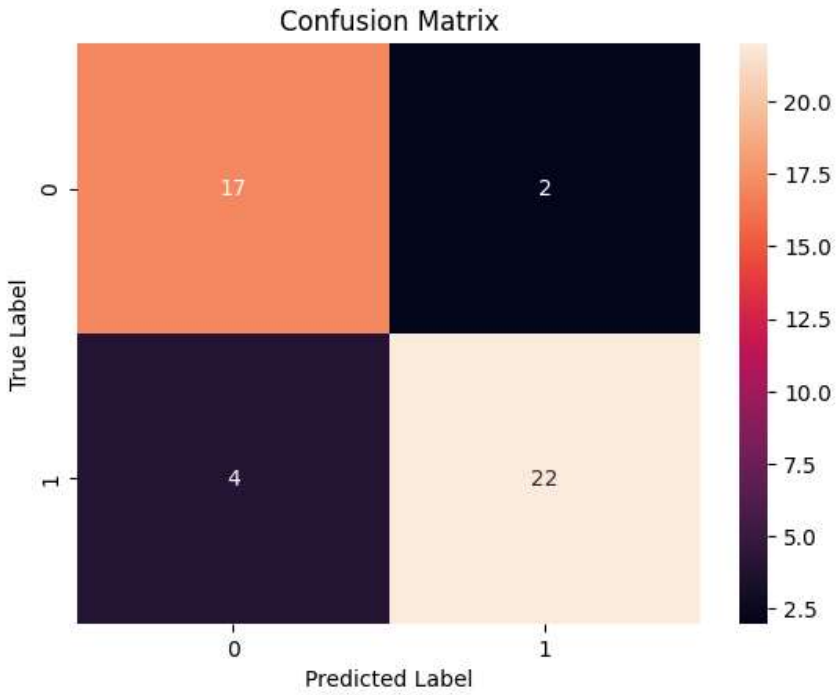
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

lr= LogisticRegression()
lr.fit(x_smote, y_smote)

y_pred_smote=lr.predict(X_test)

accuracy_score(y_test,y_pred_smote)
print("Smote Accuracy : {:.2f}%".format(accuracy_score(y_test,y_pred_smote)*100))

```

```
Smote Accuracy : 80.00%
```

Confusion Matrix LogisticRegression

The rows in a confusion matrix represent the actual labels of the data points, and the columns represent the labels that the algorithm predicted. The diagonal cells show the number of correct predictions, while the off-diagonal cells show the number of incorrect predictions.

```
from sklearn.metrics import confusion_matrix
matrix= confusion_matrix(y_test, y_pred)
sns.heatmap(matrix,annot = True, fmt = "d")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Top Left (17): This cell shows that the model correctly predicted 17 data points that actually belong to class "0".

Top Right (2): This cell shows that the model incorrectly predicted class "1" for 2 data points that actually belong to class "0". This is also known as a false positive.

Bottom Left (7.5): This cell shows that the model incorrectly predicted class "0" for 7.5 data points that actually belong to class "1". This is also known as a false negative.

Bottom Right (22): This cell shows that the model correctly predicted 22 data points that actually belong to class "1"

Precision Score and Classification report

```
from sklearn.metrics import precision_score,classification_report
precision = precision_score(y_test, y_pred)
print("Precision: ",precision)
classification_report=classification_report(y_test, y_pred)
print(classification_report)
```

```
Precision:  0.9166666666666666
              precision    recall  f1-score   support

           0       0.81      0.89      0.85        19
           1       0.92      0.85      0.88        26

    accuracy                           0.87        45
   macro avg       0.86      0.87      0.86        45
```

```
weighted avg       0.87      0.87      0.87        45
```

KNN Model

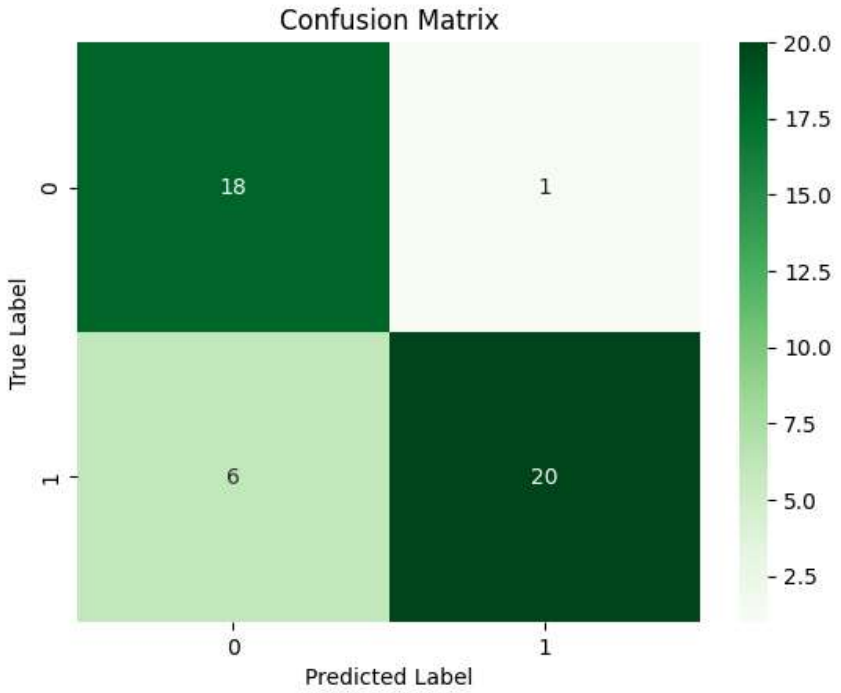
KNN works based on the idea that similar data points tend to have similar characteristics. When making a prediction about a new data point, it considers the labels of the k closest data points in the training data set.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier

# Normalize the data
scaler = MinMaxScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)

# KNN Model
knn = KNeighborsClassifier(n_neighbors = 8)
knn.fit(X_train_norm, y_train)
y_pred_knn = knn.predict(X_test_norm)

print("KNN : {:.2f}%".format(knn.score(X_test_norm, y_test)*100))
```

```
KNN : 84.44%
```

Confusion matrix of KNN

```
matrix = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(matrix, annot=True, fmt="d",cmap="Greens")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import precision_score,classification_report
precision = precision_score(y_test, y_pred_knn)
print("Precision: ",precision)
classification_report=classification_report(y_test, y_pred_knn)
print(classification_report)
```

```
Precision:  0.9523809523809523
              precision    recall  f1-score   support

           0       0.75      0.95      0.84        19
           1       0.95      0.77      0.85        26

    accuracy                           0.84        45
   macro avg       0.85      0.86      0.84        45
weighted avg       0.87      0.84      0.85        45
```

In this code, we first normalize the data using MinMaxScaler. Then, we create a KNN model with 8 neighbors and fit it to the normalized training data. We use the normalized test data to make predictions and calculate the accuracy score.

Support Vector Machine (SVM) Algorithm

SVM is the Supervised Machine Learning algorithm used for both classification, regression. But mostly preferred for classification.

Given a dataset, the algorithm tries to divide the data using hyperplanes and then makes the predictions. SVM is a non-probabilistic linear classifier. While other classifiers, when classifying, predict the probability of a data point to belong to one group or the another, SVM directly says to which group the datapoint belongs to without using any probability calculation.

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

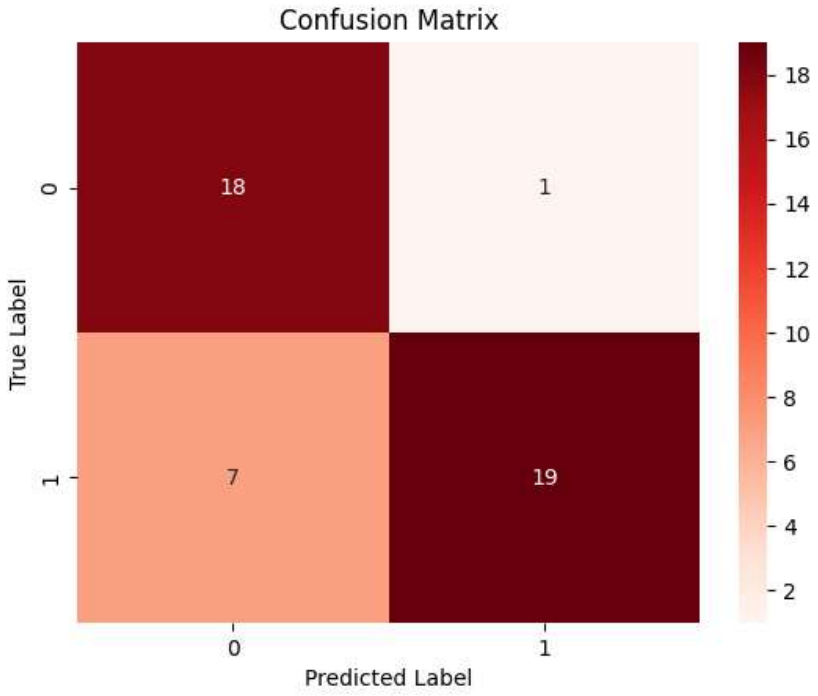
```
svm = SVC(kernel='rbf', C=10, gamma=0.01)
svm.fit(X_train_scaled, y_train)
prediction = svm.predict(X_test_scaled)

```

```
y_pred_scm = svm.predict(X_test_scaled)
print("SVM Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred_scm)*100))
```

```
SVM Accuracy: 82.22%
```

Confusion matrix of SVM

```
matrix = confusion_matrix(y_test, y_pred_scm)
sns.heatmap(matrix, annot=True, fmt="d", cmap="Reds")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Precision score and classification report

```
1 from sklearn.metrics import precision_score,classification_report
2 precision = precision_score(y_test, y_pred_scm)
3 print("Precision: ",precision)
4 classification_report=classification_report(y_test, y_pred_scm)
5 print(classification_report)
```

```
Precision:  0.95
              precision    recall  f1-score   support

           0       0.72      0.95      0.82        19
           1       0.95      0.73      0.83        26

    accuracy                           0.82        45
   macro avg       0.83      0.84      0.82        45
weighted avg       0.85      0.82      0.82        45
```

We first normalize the data using StandardScaler. Then, we create an SVM model with a radial basis function kernel (RBF), regularization parameter C=10, and kernel coefficient gamma=0.01. The SVM model is then fit to the scaled training data and used to make predictions on the scaled test data. The accuracy score is calculated using the accuracy_score function from scikit-learn.

"Note > that normalization is often recommended for SVM models, as they can be sensitive to the scale of the features. StandardScaler is a common choice for normalization, as it scales each feature to have zero mean and unit variance. However, other normalization methods like MinMaxScaler can also be used."

Naive Bayes

Naive Bayes is a popular algorithm in machine learning used for classification tasks. It's based on Bayes' theorem and makes predictions based on probabilities.

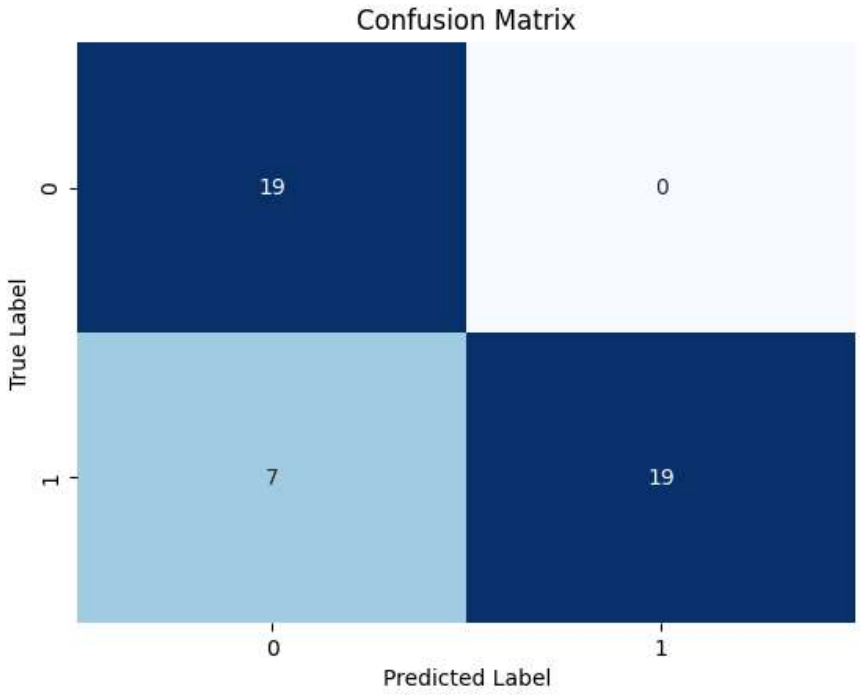
Classifies new data points into predefined categories. Imagine sorting emails into spam or inbox, classifying documents by topic, or predicting if a patient has a certain disease.

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
acc = nb.score(X_test,y_test)*100

print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
```

```
Accuracy of Naive Bayes: 84.44%
```

Confusion matrix of Naive Bayes

```
matrix = confusion_matrix(y_test, y_pred_nb)
sns.heatmap(matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



Precision score and classification report

```
from sklearn.metrics import precision_score,classification_report
precision = precision_score(y_test, y_pred_nb)
print("Precision: ",precision)
classification_report=classification_report(y_test, y_pred_nb)
print(classification_report)
```

```
Precision:  1.0
              precision    recall  f1-score   support
```

```
              0       0.73      1.00      0.84        19
              1       1.00      0.73      0.84        26

       accuracy                           0.84        45
      macro avg       0.87      0.87      0.84        45
   weighted avg       0.89      0.84      0.84        45
```

Random Forest

Random forest is a powerful and versatile machine learning algorithm that excels in both classification and regression tasks. It belongs to the category of ensemble learning algorithms, which means it combines the predictions of multiple models to generate a more robust final prediction.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(X_train ,y_train)
y_pred_rf = rf.predict(X_test)
acc = rf.score(X_test,y_test)*100
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
```

```
Random Forest Algorithm Accuracy Score : 82.22%
```

Confusion matrix of Random Forest

```
matrix = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(matrix, annot=True, fmt="d")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```