

# HEART DISEASE DETECTION

NAME: NITIN SINGH TATRARI

BATCH: 1825

### Problem statement:

**Heart disease** describes a range of conditions that affect your heart. The term “heart disease” is often used interchangeably with the term “cardiovascular disease”. Cardiovascular disease generally refers to conditions that involve narrowed or blocked blood vessels that can lead to a heart attack, chest pain (angina) or stroke. Other heart conditions, such as those that affect your heart’s muscle, valves or rhythm, also are considered forms of heart disease.

Heart disease is one of the biggest causes of morbidity and mortality among the population of the world. Prediction of cardiovascular disease is regarded as one of the most important subjects in the section of clinical data analysis. The amount of data in the healthcare industry is huge. Data mining turns the large collection of raw healthcare data into information that can help to make informed decisions and predictions. It is difficult to identify heart disease because of several contributory risk factors such as diabetes, high blood pressure, high cholesterol, abnormal pulse rate, and many other factors. Due to such constraints, scientists have turned towards modern approaches like Data Mining and Machine Learning for predicting the disease.

Machine learning (ML) proves to be effective in assisting in making decisions and predictions from the large quantity of data produced by the healthcare industry.

In this article, I’ll discuss a project where I worked on predicting potential Heart Diseases in people using Machine Learning algorithms. The algorithms included K Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier, and Logistic Regression & Random Forest Classifier.

## DATA ANALYSIS:

The data provided have 12 attributes and 1 target variable. All the variables are as follows:-

Sl. no.	Attribute	Description
0	Age	Age in years
1	sex	(1 = male; 0 = female)
2	cp	chest pain type - (1,2,3,4)
3	trestbps	resting blood pressure (in mm Hg on admission to the hospital)
4	chol	serum cholesterol in mg/dl
5	fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
6	restecg	resting electrocardiographic results (0 = normal; 1 = having ST-T; 2 = hypertrophy)
7	thalach	maximum heart rate achieved
8	exang	exercise induced angina (1 = yes; 0 = no)
9	oldpeak	ST depression induced by exercise relative to rest
10	slope	the slope of the peak exercise ST segment
11	ca	number of major vessels colored by fluoroscopy
12	thal	Thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect)
13	Target	1 or 0

### 1. Importing libraries and dataset

Firstly, we import libraries- pandas, numpy, seaborn & matplotlib. Through pandas we import data.

Through pandas we import data "heart\_disease.csv".

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import warnings
6 warnings.filterwarnings('ignore')
```

```
1 data=pd.read_csv('heart_disease.csv',header=None)
2 data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	63	1	4	140	260	0	1	112	1	3	2	?	?	2
1	44	1	4	130	209	0	1	127	0	0	?	?	?	0
2	60	1	4	132	218	0	1	140	1	1.5	3	?	?	2
3	55	1	4	142	228	0	1	149	1	2.5	1	?	?	1
4	66	1	3	110	213	1	2	99	1	1.3	2	?	?	0

## 2. Loading data into Data Frame and renaming columns' name.

```
1 df=pd.DataFrame(data)
```

```
1 colmn=['age','sex','cp','trestbps','chol','fbs','restecg','thalach','exang','oldpeak','slope','ca','thal','target']
2 df.columns=colmn
```

## 3. Understanding Data

```
1 for i in df.columns:
2     a=df[i].value_counts()
3     print('Column',i)
4     print(a)
```

Column age

```
63    15
62    15
55    14
60    12
58    12
61    12
57    10
64     9
56     9
69     8
59     8
54     7
51     7
65     6
74     5
67     5
53     5
68     4
48     4
72     3
66     3
75     3
40     2
46     2
50     2
49     2
41     2
52     2
71     2
42     1
37     1
38     1
77     1
43     1
44     1
45     1
76     1
70     1
35     1
```

Name: age, dtype: int64

Column sex

```
1    194
0      6
```

Name: sex, dtype: int64

Column chol

```
0      49
203     4
220     4
223     4
289     4
```

...

```
297     1
160     1
384     1
333     1
142     1
```

Name: chol, Length: 99, dtype: int64

Column fbs

```
0      129
1       71
```

Name: fbs, dtype: int64

Column restecg

```
1     93
0     80
2     27
```

Name: restecg, dtype: int64

Column cp

```
4      131
3       47
2       14
1        8
```

Name: cp, dtype: int64

Column trestbps

```
120    24
130    19
150    16
140    15
160    15
110    12
122    10
128     6
134     6
144     6
125     5
142     5
158     4
126     4
136     4
170     4
124     4
154     4
112     3
156     3
132     3
104     3
155     2
180     2
178     2
116     2
138     2
152     2
127     2
0        1
172     1
102     1
106     1
146     1
118     1
190     1
135     1
96       1
100      1
114      1
```

Name: trestbps, dtype: int64

Column exang

```
1      127
0       73
```

Name: exang, dtype: int64

Column oldpeak

```
0       62
2       35
1.5     31
1       21
3       18
2.5     12
0.5      8
4        6
1.3      2
1.6      1
3.5      1
0.8      1
-0.5     1
1.7      1
```

Name: oldpeak, dtype: int64

Column target

```
1      56
0      51
3      42
2      41
4      10
```

Name: target, dtype: int64

Column	thalach
120	112
140	127
145	140
146	149
147	112
148	127
149	140
150	149
151	112
152	127
153	140
154	149
155	112
156	127
157	140
158	149
159	112
160	127
161	140
162	149
163	112
164	127
165	140
166	149
167	112
168	127
169	140
170	149
171	112
172	127
173	140
174	149
175	112
176	127
177	140
178	149
179	112
180	127
181	140
182	149
183	112
184	127
185	140
186	149
187	112
188	127
189	140
190	149
191	112
192	127
193	140
194	149
195	112
196	127
197	140
198	149
199	112
200	127
201	140
202	149
203	112
204	127
205	140
206	149
207	112
208	127
209	140
210	149
211	112
212	127
213	140
214	149
215	112
216	127
217	140
218	149
219	112
220	127
221	140
222	149
223	112
224	127
225	140
226	149
227	112
228	127
229	140
230	149
231	112
232	127
233	140
234	149
235	112
236	127
237	140
238	149
239	112
240	127
241	140
242	149
243	112
244	127
245	140
246	149
247	112
248	127
249	140
250	149
251	112
252	127
253	140
254	149
255	112
256	127
257	140
258	149
259	112
260	127
261	140
262	149
263	112
264	127
265	140
266	149
267	112
268	127
269	140
270	149
271	112
272	127
273	140
274	149
275	112
276	127
277	140
278	149
279	112
280	127
281	140
282	149
283	112
284	127
285	140
286	149
287	112
288	127
289	140
290	149
291	112
292	127
293	140
294	149
295	112
296	127
297	140
298	149
299	112
300	127
301	140
302	149
303	112
304	127
305	140
306	149
307	112
308	127
309	140
310	149
311	112
312	127
313	140
314	149
315	112
316	127
317	140
318	149
319	112
320	127
321	140
322	149
323	112
324	127
325	140
326	149
327	112
328	127
329	140
330	149
331	112
332	127
333	140
334	149
335	112
336	127
337	140
338	149
339	112
340	127
341	140
342	149
343	112
344	127
345	140
346	149
347	112
348	127
349	140
350	149
351	112
352	127
353	140
354	149
355	112
356	127
357	140
358	149
359	112
360	127
361	140
362	149
363	112
364	127
365	140
366	149
367	112
368	127
369	140
370	149
371	112
372	127
373	140
374	149
375	112
376	127
377	140
378	149
379	112
380	127
381	140
382	149
383	112
384	127
385	140
386	149
387	112
388	127
389	140
390	149
391	112
392	127
393	140
394	149
395	112
396	127
397	140
398	149
399	112
400	127
401	140
402	149
403	112
404	127
405	140
406	149
407	112
408	127
409	140
410	149
411	112
412	127
413	140
414	149
415	112
416	127
417	140
418	149
419	112
420	127
421	140
422	149
423	112
424	127
425	140
426	149
427	112
428	127
429	140
430	149
431	112
432	127
433	140
434	149
435	112
436	127
437	140
438	149
439	112
440	127
441	140
442	149
443	112
444	127
445	140
446	149
447	112
448	127
449	140
450	149
451	112
452	127
453	140
454	149
455	112
456	127
457	140
458	149
459	112
460	127
461	140
462	149
463	112
464	127
465	140
466	149
467	112
468	127
469	140
470	149
471	112
472	127
473	140
474	149
475	112
476	127
477	140
478	149
479	112
480	127
481	140
482	149
483	112
484	127
485	140
486	149
487	112
488	127
489	140
490	149
491	112
492	127
493	140
494	149
495	112
496	127
497	140
498	149
499	112
500	127
501	140
502	149
503	112
504	127
505	140
506	149
507	112
508	127
509	140
510	149
511	112
512	127
513	140
514	149
515	112
516	127
517	140
518	149
519	112
520	127
521	140
522	149
523	112
524	127
525	140
526	149
527	112
528	127
529	140
530	149
531	112
532	127
533	140
534	149
535	112
536	127
537	140
538	149
539	112
540	127
541	140
542	149
543	112
544	127
545	140
546	149
547	112
548	127
549	140
550	149
551	112
552	127
553	140
554	149
555	112
556	127
557	140
558	149
559	112
560	127
561	140
562	149
563	112
564	127
565	140
566	149
567	112
568	127
569	140
570	149
571	112
572	127
573	140
574	149
575	112
576	127
577	140
578	149
579	112
580	127
581	140
582	149
583	112
584	127
585	140
586	149
587	112
588	127
589	140
590	149
591	112
592	127
593	140
594	149
595	112
596	127
597	140
598	149
599	112
600	127
601	140
602	149
603	112
604	127
605	140
606	149
607	112
608	127
609	140
610	149
611	112
612	127
613	140
614	149
615	112
616	127
617	140
618	149
619	112
620	127
621	140
622	149
623	112
624	127
625	140
626	149
627	112
628	127
629	140
630	149
631	112
632	127
633	140
634	149
635	112
636	127
637	140
638	149
639	112
640	127
641	140
642	149
643	112
644	127
645	140
646	149
647	112
648	127
649	140
650	149
651	112
652	127
653	140
654	149
655	112
656	127
657	140
658	149
659	112
660	127
661	140
662	149
663	112
664	127
665	140
666	149
667	112
668	127
669	140
670	149
671	112
672	127
673	140
674	149
675	112
676	127
677	140
678	149
679	112
680	127
681	140
682	149
683	112
684	127
685	140
686	149
687	112
688	127
689	140
690	149
691	112
692	127
693	140
694	149
695	112
696	127
697	140
698	149
699	112
700	127
701	140
702	149
703	112
704	127
705	140
706	149
707	112
708	127
709	140
710	149
711	112
712	127
713	140
714	149
715	112
716	127
717	140
718	149
719	112
720	127
721	140
722	149
723	112
724	127
725	140
726	149
727	112
728	127
729	140
730</	

## B) Treating '?'

```
1 series=['trestbps','chol','fbs','thalach','exang','oldpeak']
2 for i in series:
3     df[i]=df[i].replace('?',method='ffill')
```

We have replace '?' in the columns- 'trestbps', 'chol', 'fbs', 'thalach', 'exang' & 'oldpeak' with forward fill method

## C) Treating Data type

There are 5 integer type and 6 string type columns in the data frame.

Since all columns have numerical values, we have to change the data type to float.

```
1 df.dtypes
age          int64
sex          int64
cp           int64
trestbps     object
chol         object
fbs          object
restecg      int64
thalach      object
exang        object
oldpeak      object
target       int64
dtype: object

There 5 integer type columns and 6 string type columns

Since all columns have numerical value, we can change the data type to float

1 for i in df.columns:
2     df[i]=df[i].astype(float)

1 df.dtypes
age          float64
sex          float64
cp           float64
trestbps     float64
chol         float64
fbs          float64
restecg      float64
thalach      float64
exang        float64
oldpeak      float64
target       float64
dtype: object
```

## D) Treating 'Chol' column

We will replace value '0' in the column 'chol' with the mean of column 'Chol'.

```
1 df['chol'].mean()
181.545

1 df['chol']=df['chol'].replace(0,182)

1 df['chol'].value_counts()
0          49
203         4
220         4
223         4
289         4
..
297         1
160         1
384         1
333         1
142         1
Name: chol, Length: 99, dtype: int64
```

E) There are outliers in Age, trestbps, chol & thalach.

```
1 df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	target
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	59.350000	0.970000	3.505000	134.580000	226.135000	0.355000	0.735000	122.680000	0.635000	1.296000	1.520000
std	7.811697	0.171015	0.795701	20.44022	52.378568	0.479714	0.683455	21.749316	0.482638	1.12486	1.219441
min	35.000000	0.000000	1.000000	0.000000	100.000000	0.000000	0.000000	69.000000	0.000000	-0.500000	0.000000
25%	55.000000	1.000000	3.000000	120.000000	182.000000	0.000000	0.000000	108.000000	0.000000	0.000000	0.000000
50%	60.000000	1.000000	4.000000	130.000000	216.500000	0.000000	1.000000	120.000000	1.000000	1.500000	1.000000
75%	64.000000	1.000000	4.000000	150.000000	258.000000	1.000000	1.000000	140.000000	1.000000	2.000000	3.000000
max	77.000000	1.000000	4.000000	190.000000	458.000000	1.000000	2.000000	180.000000	1.000000	4.000000	4.000000

F) Treating outliers and removing skewness

```
1 from scipy.stats import zscore
```

```
1 z=np.abs(zscore(df))
2 df2=df[(z<3).all(axis=1)]
```

```
1 df2.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	target
0	63.0	1.0	4.0	140.0	260.0	0.0	1.0	112.0	1.0	3.0	2.0
1	44.0	1.0	4.0	130.0	209.0	0.0	1.0	127.0	0.0	0.0	0.0
2	60.0	1.0	4.0	132.0	218.0	0.0	1.0	140.0	1.0	1.5	2.0
3	55.0	1.0	4.0	142.0	228.0	0.0	1.0	149.0	1.0	2.5	1.0
4	66.0	1.0	3.0	110.0	213.0	1.0	2.0	99.0	1.0	1.3	0.0

```
1 df2.describe()
```

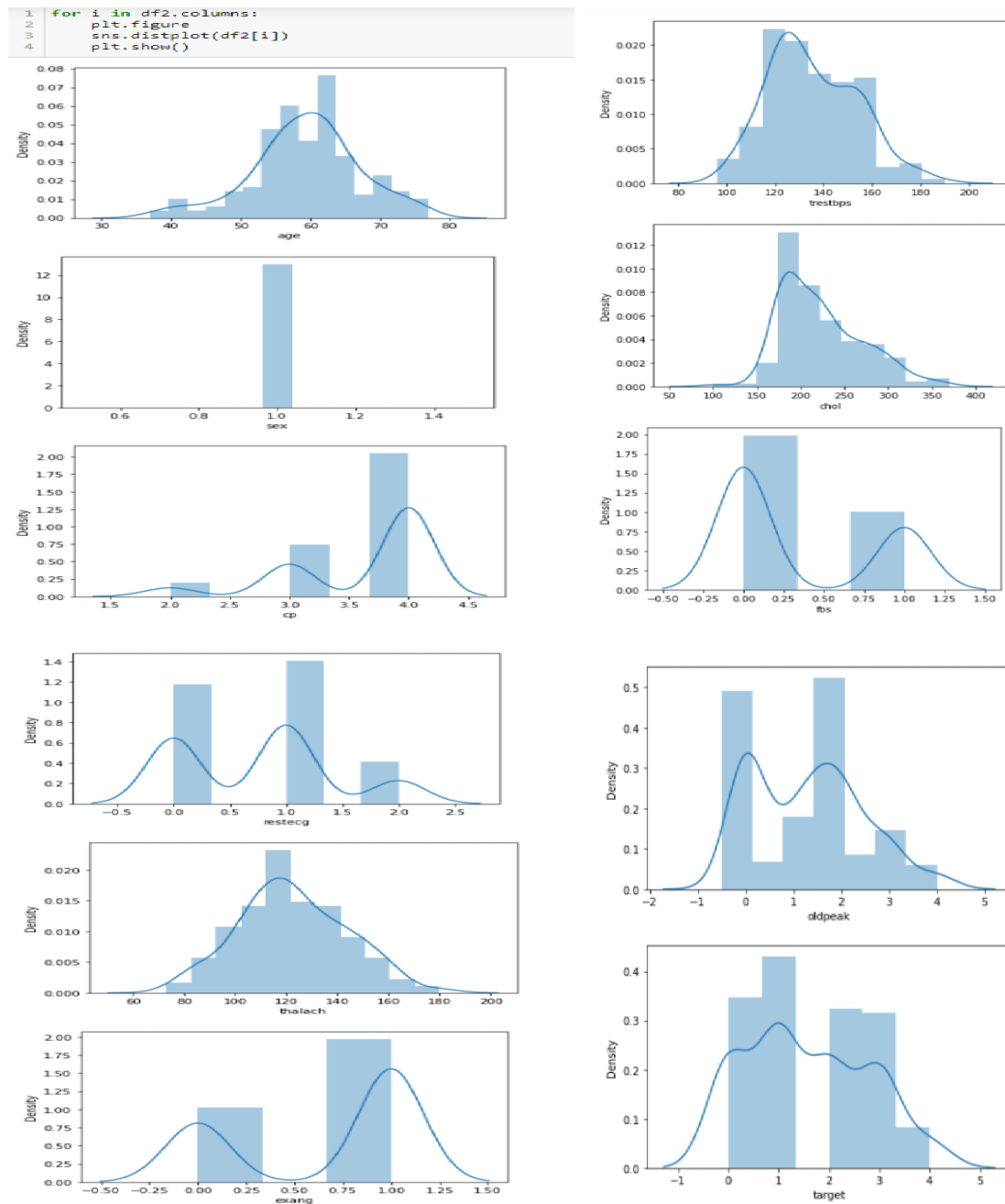
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	target
count	181.000000	181.0	181.000000	181.000000	181.000000	181.000000	181.000000	181.000000	181.000000	181.000000	181.000000
mean	59.348066	1.0	3.618785	135.558011	222.861878	0.337017	0.745856	122.707182	0.657459	1.304972	1.569061
std	7.746351	0.0	0.608715	18.433159	46.762494	0.474002	0.684387	21.248984	0.475876	1.140140	1.211947
min	37.000000	1.0	2.000000	96.000000	100.000000	0.000000	0.000000	73.000000	0.000000	-0.500000	0.000000
25%	55.000000	1.0	3.000000	122.000000	182.000000	0.000000	0.000000	110.000000	0.000000	0.000000	1.000000
50%	60.000000	1.0	4.000000	130.000000	216.000000	0.000000	1.000000	120.000000	1.000000	1.500000	1.000000
75%	63.000000	1.0	4.000000	150.000000	254.000000	1.000000	1.000000	140.000000	1.000000	2.000000	3.000000
max	77.000000	1.0	4.000000	190.000000	369.000000	1.000000	2.000000	180.000000	1.000000	4.000000	4.000000

We remove all the value less than or more than threshold =3. The shape of new Data Frame df2 is (181,11).

Outliers have reduced in Age, trestbps, chol & thalach.

#### 4. EDA

A) Now, we will graphically study the variables.



While removing outliers, value '0' having only 6 counts in column 'age' was removed. Thus it has only value '1'. Thus we can remove the column.

We will also convert replace values 2,3,4 with 1 in Target variable, as we have to find out whether the person is having heart diseases or not. We do not have to find the severity of the disease.



```
1 df2.drop('sex',axis=1,inplace=True)
```

```
1 df2.head()
```

	age	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	target
0	63.0	4.0	140.0	260.0	0.0	1.0	112.0	1.0	3.0	2.0
1	44.0	4.0	130.0	209.0	0.0	1.0	127.0	0.0	0.0	0.0
2	60.0	4.0	132.0	218.0	0.0	1.0	140.0	1.0	1.5	2.0
3	55.0	4.0	142.0	228.0	0.0	1.0	149.0	1.0	2.5	1.0
4	66.0	3.0	110.0	213.0	1.0	2.0	99.0	1.0	1.3	0.0

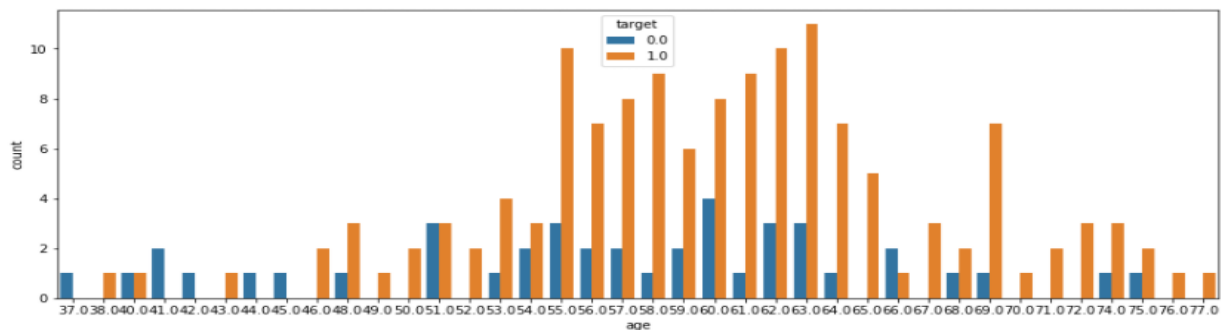
```
1 df2['target'].replace([2,3,4],1, inplace=True)
```

```
1 df2['target'].value_counts()
```

```
1.0    139
0.0     42
Name: target, dtype: int64
```

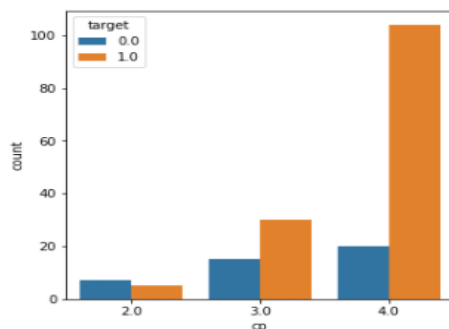
B) Graphically, we will try to interpret the relationship of target variable with other attributes.

```
1 plt.figure(figsize=(15,5))
2 sns.countplot(x='age',hue='target',data=df2)
3 plt.show()
```



he can interpret that at younger age changes for heart disease is low.

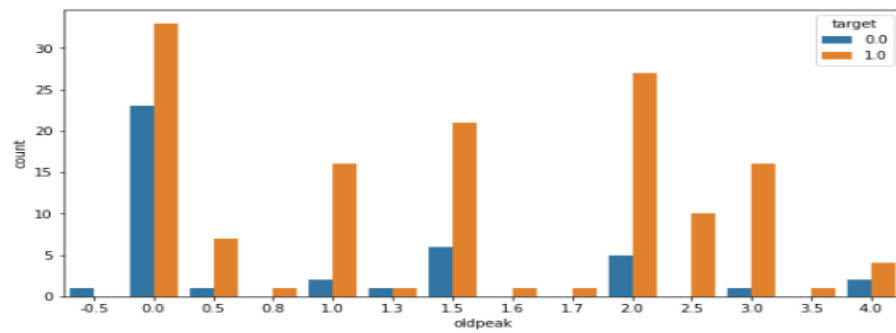
```
1 plt.figure(figsize=(5,5))
2 sns.countplot(x='cp',hue='target',data=df2)
3 plt.show()
```



```

1 plt.figure(figsize=(10,5))
2 sns.countplot(x='oldpeak',hue='target',data=df2)
3 plt.show()

```

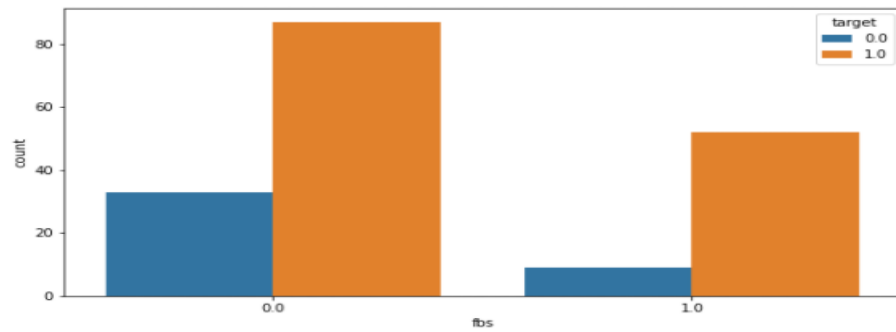


with the increase of oldpeak value , probability of heart diseases increases.

```

1 plt.figure(figsize=(10,5))
2 sns.countplot(x='fbs',hue='target',data=df2)
3 plt.show()

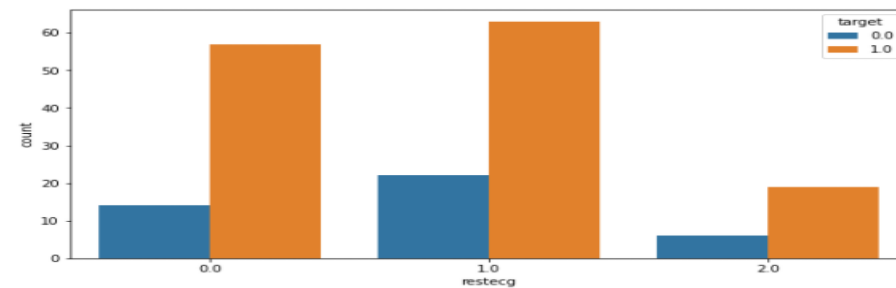
```



```

1 plt.figure(figsize=(10,5))
2 sns.countplot(x='restecg',hue='target',data=df2)
3 plt.show()

```

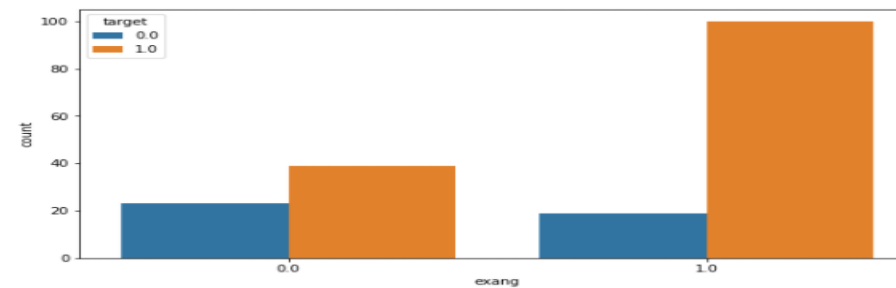


Patience having restecg value (1 = having ST-T) & (2 = hypertrophy), chances for heart diseases is less than 0 =normal

```

1 plt.figure(figsize=(10,5))
2 sns.countplot(x='exang',hue='target',data=df2)
3 plt.show()

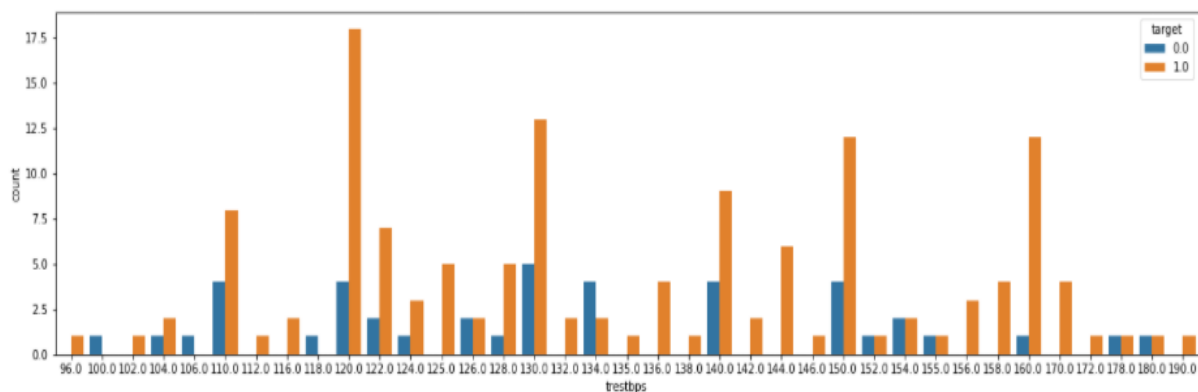
```



```

1 plt.figure(figsize=(20,5))
2 sns.countplot(x='trestbps',hue='target',data=df2)
3 plt.show()

```

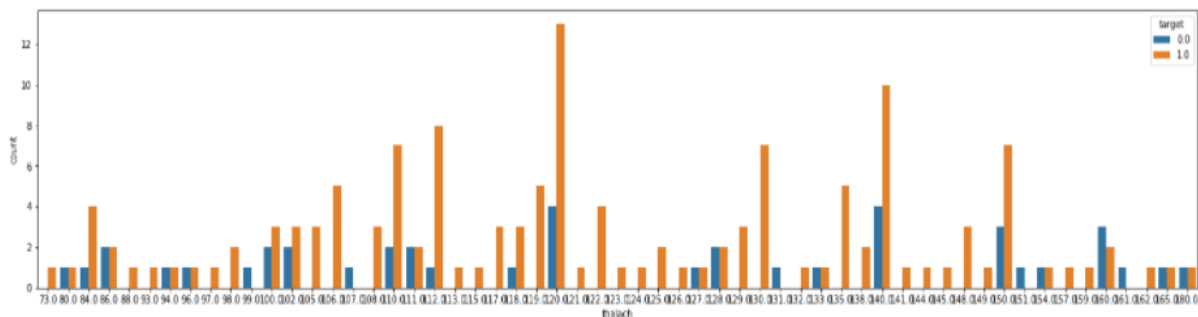


Patience having high trestbps(Resting blood pressure) tends to have more chances of Heart diseases

```

1 plt.figure(figsize=(25,5))
2 sns.countplot(x='thalach',hue='target',data=df2)
3 plt.show()

```

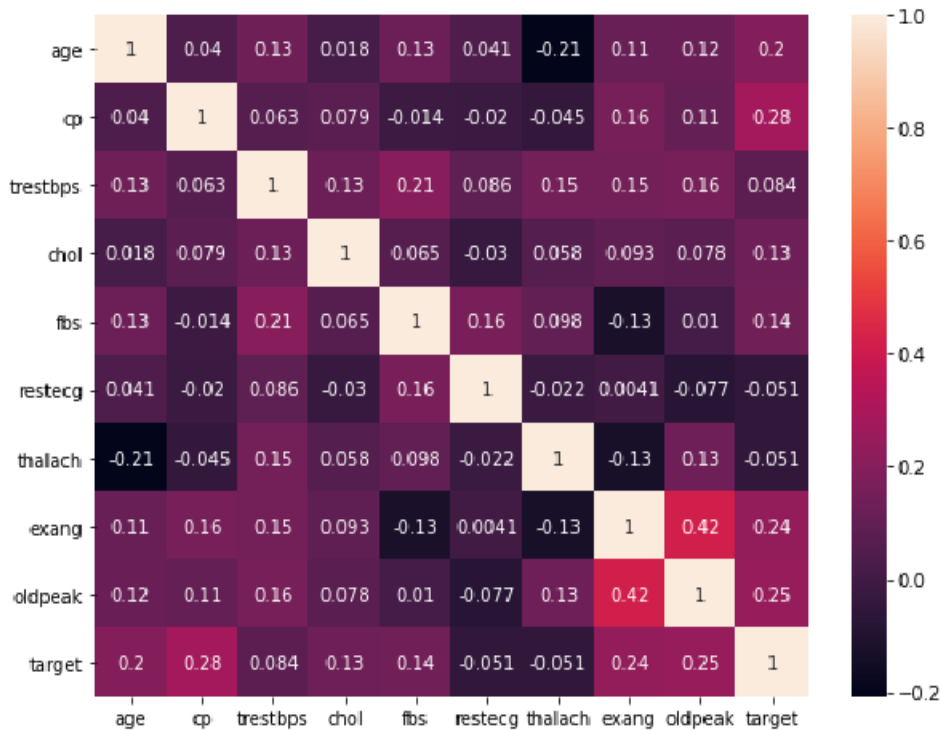


Following points observed from the graphical representation:-

- The chances of Heart diseases in younger age group are less.
- Patience have type 4 of CP have more chances to diagnosed with heart disease.
- Higher value of oldpeak value shows higher percentage of heart diseases.
- We have observed patience having higher fasting sugar level, tend to have more chances to diagnose with heart disease.
- Patience having restecg value (1 = having ST-T) & (2 = hypertrophy) have lesser ratio of diagnosed with heart diseases is less than 0 =normal.
- Exercise induced angina is seen more in patience having heart problem.
- Patience having high trestbps(Resting blood pressure) tends to have higher diagnosed ratio of Heart diseases
- Heart disease is high between 100-150 value of thalach(maximum heart rate achieved).

### C) Analyzing Correlation of target variable with other variables

```
1 plt.figure(figsize=(9,7))
2 sns.heatmap(df2.corr(), annot=True)
3 plt.show()
```



```
1 df2.drop({'thalach', 'restecg'}, axis=1, inplace=True)
2 df2.head()
```

	age	cp	trestbps	chol	fbs	exang	oldpeak	target
0	63.0	4.0	140.0	260.0	0.0	1.0	3.0	1.0
1	44.0	4.0	130.0	209.0	0.0	0.0	0.0	0.0
2	60.0	4.0	132.0	218.0	0.0	1.0	1.5	1.0
3	55.0	4.0	142.0	228.0	0.0	1.0	2.5	1.0
4	66.0	3.0	110.0	213.0	1.0	1.0	1.3	0.0

'age', 'cp', 'Chol', 'fbs', 'exang', 'oldpeak' shows correlation with target.

'restecg' and 'thalach' shows less than 6% correlation with target. Thus, we have drop them.

## 5. Model Development and Evaluation

We drop 'Target' column from df2 and store the remaining variables in array 'x'.

We store the target variable in array 'y'

```
1 x=df2.drop(['target'],axis=1)
2 x.head(5)
```

	age	cp	trestbps	chol	fbs	exang	oldpeak
0	63.0	4.0	140.0	260.0	0.0	1.0	3.0
1	44.0	4.0	130.0	209.0	0.0	0.0	0.0
2	60.0	4.0	132.0	218.0	0.0	1.0	1.5
3	55.0	4.0	142.0	228.0	0.0	1.0	2.5
4	66.0	3.0	110.0	213.0	1.0	1.0	1.3

```
1 x.shape
(181, 7)
```

```
1 y=df2['target']
2 y
```

```
0      1.0
1      0.0
2      1.0
3      1.0
4      0.0
...
192     1.0
193     1.0
194     1.0
197     1.0
199     1.0
Name: target, Length: 181, dtype: float64
```

```
1 y.shape
(181,)
```

Both array x and y have equal rows, which is 181.

By using StandardScaler from sklearn library, we standardizes the value in 'x'.

```
1 from sklearn.preprocessing import StandardScaler
2 sc=StandardScaler()
3 x=sc.fit_transform(x)
4 x
```

```
array([[ 0.47274696,  0.62799976,  0.24164663, ..., -0.71297499,
         0.72180912,  1.49080751],
       [-1.98682459,  0.62799976, -0.30235885, ..., -0.71297499,
        -1.38540782, -1.14774685],
       [ 0.08439356,  0.62799976, -0.19355775, ..., -0.71297499,
         0.72180912,  0.17153033],
       ...,
       [-1.72792232,  0.62799976, -0.08475665, ..., -0.71297499,
        -1.38540782, -1.14774685],
       [-0.56286211,  0.62799976, -0.73756323, ...,  1.40257375,
        -1.38540782, -1.14774685],
       [ 0.34329583, -2.66672362, -0.84636433, ..., -0.71297499,
         0.72180912, -1.14774685]])
```

## A) Importing algorithms and key metrics

The algorithms used for training and testing are `LogisticRegression()`, `DecisionTreeClassifier()`, `KNeighborsClassifier()` & `SpaceVectorClassifier()`.

The key metrics used are Accuracy score, Confusion matrix, `roc_score` and classification report.

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.neighbors import KNeighborsClassifier
4 from sklearn.svm import SVC
5 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, roc_auc_score
6 from sklearn.model_selection import train_test_split, cross_val_score
```

```
1 x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2, random_state=42)
```

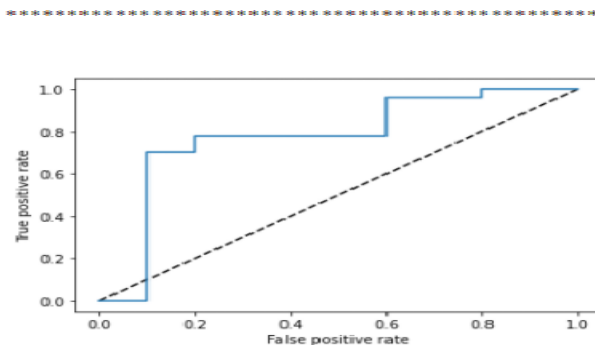
## B) Model Selection

```
1 model=[LogisticRegression(),DecisionTreeClassifier(), KNeighborsClassifier()]
2 for i in model:
3     print(i)
4     i.fit(x_train,y_train)
5     pred=i.predict(x_test)
6     y_pred_pro=i.predict_proba(x_test)[:,-1]
7     fpr,tpr,thresholds=roc_curve(y_test,y_pred_pro)
8     print('Accuracy score :', accuracy_score(y_test,pred))
9     print('Confusion matrix :\n', confusion_matrix(y_test,pred))
10    print('Classification report: \n ', classification_report(y_test,pred))
11    print('*****')
12    print('\n')
13    plt.plot([0,1],[0,1], 'k--')
14    plt.plot(fpr,tpr,label='KNeighborClassifier')
15    plt.xlabel('False positive rate')
16    plt.ylabel('True positive rate')
17    plt.show()
```

```
LogisticRegression()
Accuracy score : 0.7837837837837838
Confusion matrix :
[[ 4  6]
 [ 2 25]]
Classification report:
              precision    recall  f1-score   support

      0.0         0.67      0.40      0.50         10
      1.0         0.81      0.93      0.86         27

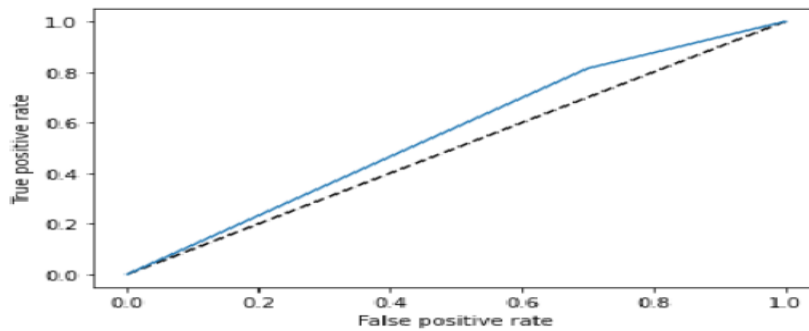
 accuracy         0.78         37
 macro avg        0.74         0.66      0.68         37
 weighted avg     0.77         0.78      0.76         37
```



```
DecisionTreeClassifier()
Accuracy score : 0.6756756756756757
Confusion matrix :
[[ 3  7]
 [ 5 22]]
Classification report:
```

	precision	recall	f1-score	support
0.0	0.38	0.30	0.33	10
1.0	0.76	0.81	0.79	27
accuracy			0.68	37
macro avg	0.57	0.56	0.56	37
weighted avg	0.65	0.68	0.66	37

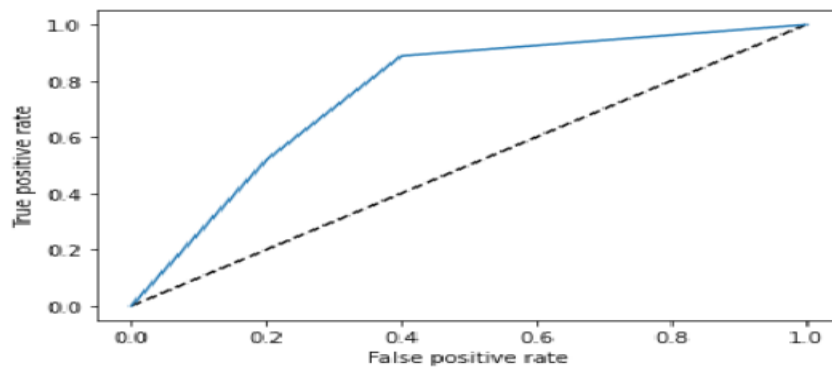
\*\*\*\*\*



```
KNeighborsClassifier()
Accuracy score : 0.7567567567567568
Confusion matrix :
[[ 2  8]
 [ 1 26]]
Classification report:
```

	precision	recall	f1-score	support
0.0	0.67	0.20	0.31	10
1.0	0.76	0.96	0.85	27
accuracy			0.76	37
macro avg	0.72	0.58	0.58	37
weighted avg	0.74	0.76	0.71	37

\*\*\*\*\*



```

1  svc=SVC(probability=True)
2  svc.fit(x_train,y_train)
3  pred=svc.predict(x_test)
4  y_pred_pro=svc.predict_proba(x_test)[:,:1]
5  fpr,tpr,thresholds=roc_curve(y_test,y_pred_pro)
6  print('Accuracy score :', accuracy_score(y_test,pred))
7  print('Confusion matrix :\n', confusion_matrix(y_test,pred))
8  print('Classification report: \n ', classification_report(y_test,pred))
9  plt.plot([0,1],[0,1], 'k--')
10 plt.plot(fpr,tpr,label='KNeighborClassifier')
11 plt.xlabel('False positive rate')
12 plt.ylabel('True positive rate')
13 plt.show()

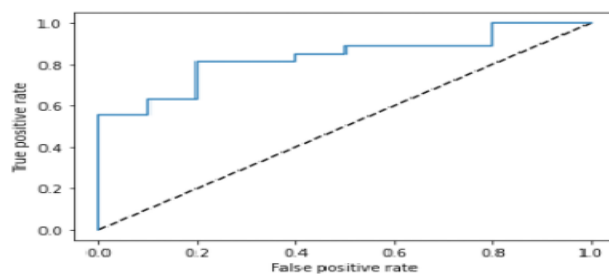
```

```

Accuracy score : 0.7567567567567568
Confusion matrix :
[[ 2  8]
 [ 1 26]]
Classification report:

```

	precision	recall	f1-score	support
0.0	0.67	0.20	0.31	10
1.0	0.76	0.96	0.85	27
accuracy			0.76	37
macro avg	0.72	0.58	0.58	37
weighted avg	0.74	0.76	0.71	37



LogisticRegression() algorithm is showing highest accuracy score of 78.37% among all other algorithms.

### C) Ensemble Technique

```

1  from sklearn.ensemble import RandomForestClassifier
2  rf=RandomForestClassifier()
3  rf.fit(x_train,y_train)
4  pred2=rf.predict(x_test)
5  print('Accuracy score :', accuracy_score(y_test,pred2))
6  print('Confusion matrix :\n', confusion_matrix(y_test,pred2))
7  print('Classification report: \n ', classification_report(y_test,pred2))

```

```

Accuracy score : 0.7567567567567568
Confusion matrix :
[[ 2  8]
 [ 1 26]]
Classification report:

```

	precision	recall	f1-score	support
0.0	0.67	0.20	0.31	10
1.0	0.76	0.96	0.85	27
accuracy			0.76	37
macro avg	0.72	0.58	0.58	37
weighted avg	0.74	0.76	0.71	37



## D) Hyperparameter Tuning

```
1 from sklearn.model_selection import GridSearchCV
2 parameters = {'C':[1,10], 'random_state': range(42,100)}
3 lg=LogisticRegression()
4 Grid=GridSearchCV(lg,parameters)
5 Grid.fit(x_train,y_train)
```

```
GridSearchCV(estimator=LogisticRegression(),
              param_grid={'C': [1, 10], 'random_state': range(42, 100)})
```

```
1 Grid.best_params_
```

```
{'C': 1, 'random_state': 42}
```

Since it is the default parameters, we will that only.

```
1 lg=LogisticRegression()
2 lg.fit(x_train,y_train)
3 pred3=lg.predict(x_test)
```

Since LogisticRegression() gives the best accuracy, we apply hyperparameter Tuning on it. The best result is shown in default parameter. They are random\_state: 42 and float: 1.

It gives an accuracy score of 78.38%.