Import the necessary libraries

In [1]:

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

Dataset to be imported which to be analysed.

In [5]:

```
df = pd.read_excel("C:\\Users\\VIMAL MADHAN\\Desktop\\pga15 python\\HeartDiseasePrediction.
```

Now the dataset will showing all details inform of table by rows and columns

In [6]:

```
df
```

Out[6]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	ex
0	63	1	3	145	233	1	0	150	
1	37	1	2	130	250	0	1	187	
2	41	0	1	130	204	0	0	172	
3	56	1	1	120	236	0	1	178	
4	57	0	0	120	354	0	1	163	
...	...	...	...	...	...	...	...	...	
298	57	0	0	140	241	0	1	123	
299	45	1	3	110	264	0	1	132	
300	68	1	0	144	193	1	1	141	
301	57	1	0	130	131	0	1	115	
302	57	0	1	130	236	0	0	174	

303 rows × 14 columns

now we using set_option in pandas to display max columns or rows from the table

In [10]:

```
pd.set_option('display.max_rows',500)
```

In [11]:

```
df
```

8	52	1	2	172	199	1	1	162	0
9	57	1	2	150	168	0	1	174	0
10	54	1	0	140	239	0	1	160	0
11	48	0	2	130	275	0	1	139	0
12	49	1	1	130	266	0	1	171	0
13	64	1	3	110	211	0	0	144	1
14	58	0	3	150	283	1	0	162	0
15	50	0	2	120	219	0	1	158	0
16	58	0	2	120	340	0	1	172	0
17	66	0	3	150	226	0	1	114	0
18	43	1	0	150	247	0	1	171	0
19	69	0	3	140	239	0	1	151	0

Now we are extracting the number of rows and columns in the dataset

In [12]:

```
df.shape
```

Out[12]:

```
(303, 14)
```

This shows that how much rows and columns are in the dataset

EXPLORATORY DATA ANALYSIS(EDA)

Before building the model,first we have to analyse whether there is any missing values in the dataset

In [13]:

```
df.isna().sum().sum()
```

Out[13]:

```
0
```

Now, it is confirmed that there is no missing or null values in the dataset and we can proceed for further process

In [ ]:

```

```

We have to get the basic informations of each and every attributes in our dataset

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  303 non-null    int64  
 1   gender               303 non-null    int64  
 2   chest_pain           303 non-null    int64  
 3   rest_bps             303 non-null    int64  
 4   cholestrol           303 non-null    int64  
 5   fasting_blood_sugar  303 non-null    int64  
 6   rest_ecg             303 non-null    int64  
 7   thalach              303 non-null    int64  
 8   exer_angina          303 non-null    int64  
 9   old_peak             303 non-null    float64
 10  slope                303 non-null    int64  
 11  ca                   303 non-null    int64  
 12  thalassemia          303 non-null    int64  
 13  target               303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [ ]:

```

```

Descriptive statistics for the given data

In [15]:

```
df.describe()
```

Out[15]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.52
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.52
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.00
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.00
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.00
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.00
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.00

CORRELATION comparision between each and every variables

Correlation > 0 = Positive correlation

correlation < 0 = Negative Correlation

correlation == 0 = No correlation

In [16]:

```
df_corr=df.corr()
```

In [17]:

```
df_corr
```

Out[17]:

age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer
000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0
447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0
653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0
351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0
678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0
308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0
211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0
522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0
801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1
013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0
814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0
326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0
001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0
439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0

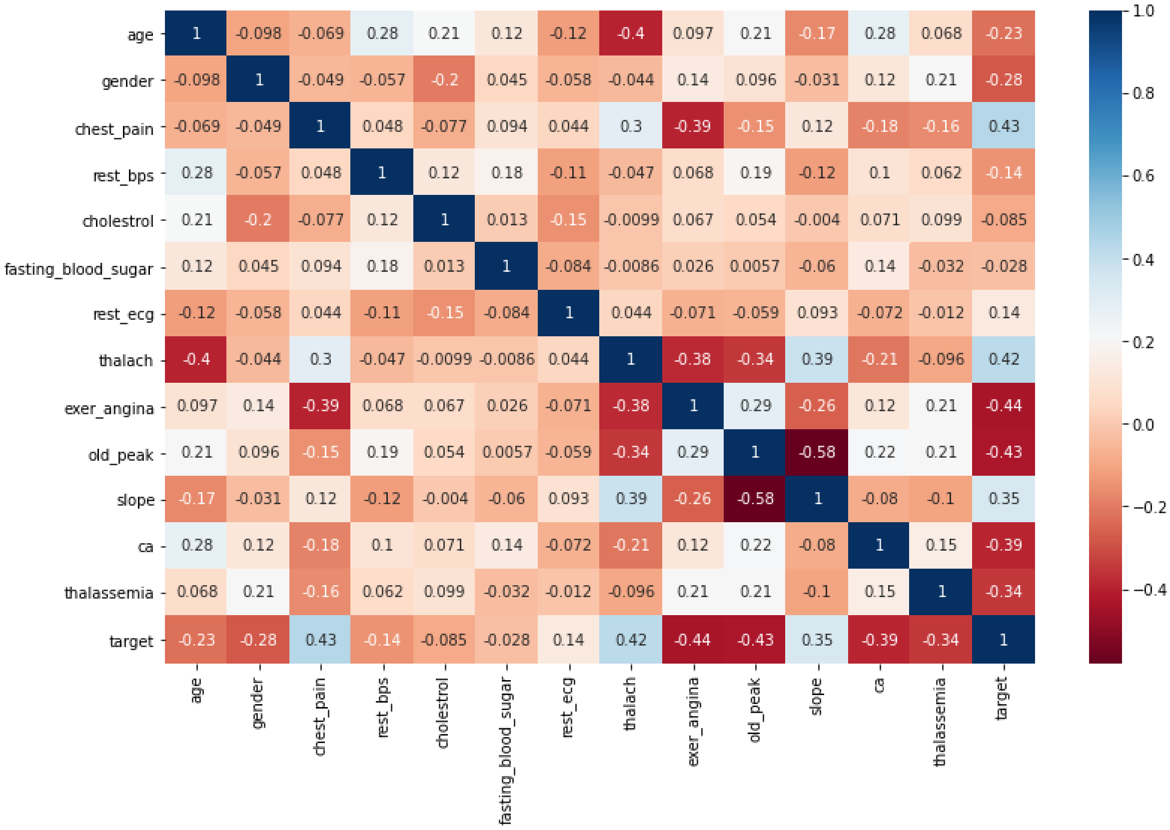
Since when there are lesser attributes it will be convinient for using Heatmap which gives the graphical correlation between each and every attributes

In [22]:

```
plt.figure(figsize=(13,8))
sns.heatmap(df_corr,annot = True,cmap = 'RdBu')
```

Out[22]:

```
<AxesSubplot:>
```



VISUALIZATION

In [24]:

```
df.head(15)
```

Out[24]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer
0	63	1	3	145	233	1	0	150	
1	37	1	2	130	250	0	1	187	
2	41	0	1	130	204	0	0	172	
3	56	1	1	120	236	0	1	178	
4	57	0	0	120	354	0	1	163	
5	57	1	0	140	192	0	1	148	
6	56	0	1	140	294	0	0	153	
7	44	1	1	120	263	0	1	173	
8	52	1	2	172	199	1	1	162	
9	57	1	2	150	168	0	1	174	
10	54	1	0	140	239	0	1	160	
11	48	0	2	130	275	0	1	139	
12	49	1	1	130	266	0	1	171	
13	64	1	3	110	211	0	0	144	
14	58	0	3	150	283	1	0	162	

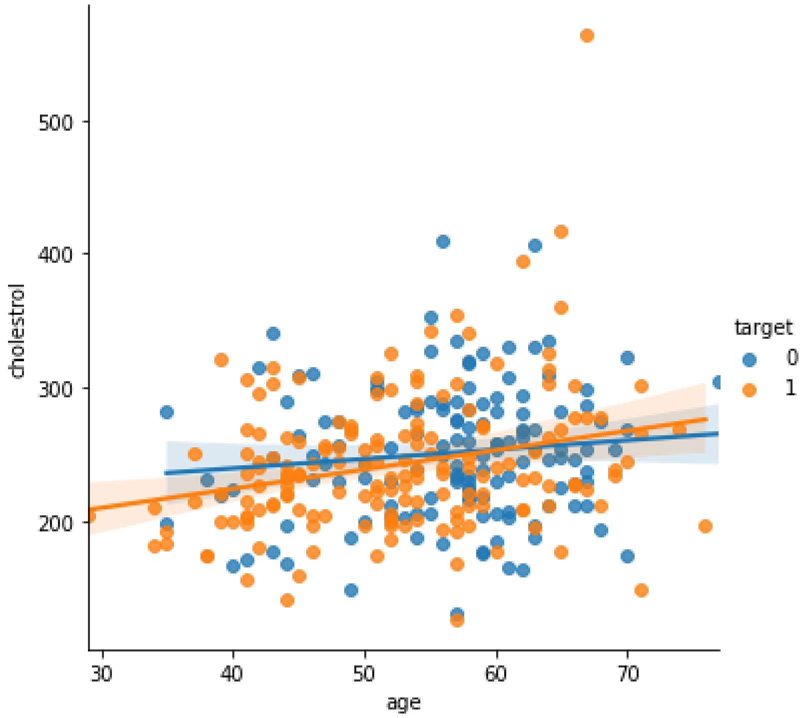
Relation between two variables

In [28]:

```
sns.lmplot(x='age',y='cholestrol',hue = 'target',data=df)
```

Out[28]:

```
<seaborn.axisgrid.FacetGrid at 0x239b43c6808>
```



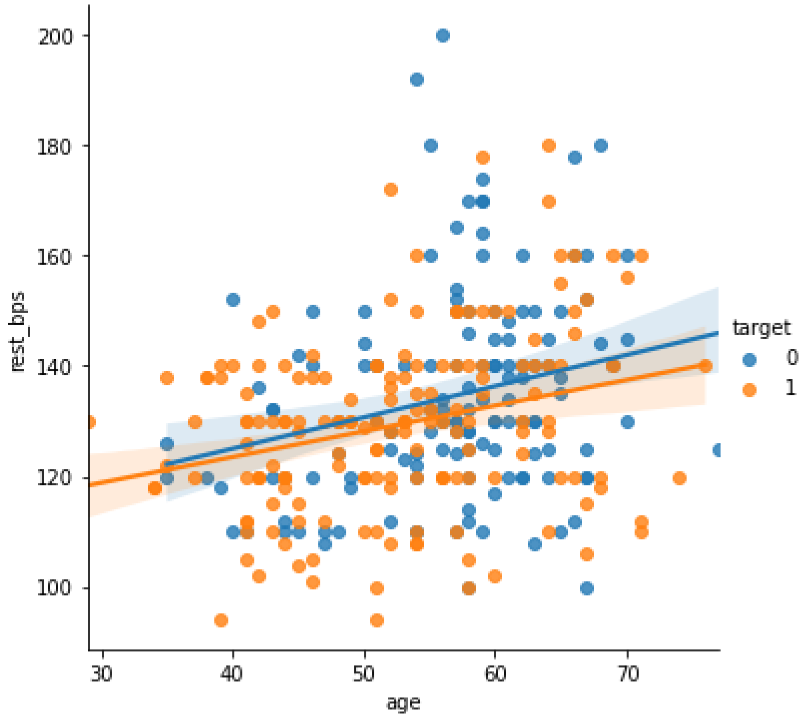
The above graph gives the relationship between age and cholestrol with respect to target.

In [29]:

```
sns.lmplot(x='age',y='rest_bps',hue = 'target',data=df)
```

Out[29]:

```
<seaborn.axisgrid.FacetGrid at 0x239b3d81ac8>
```



The above graph gives the relationship between age and rest_bps with respect to target.

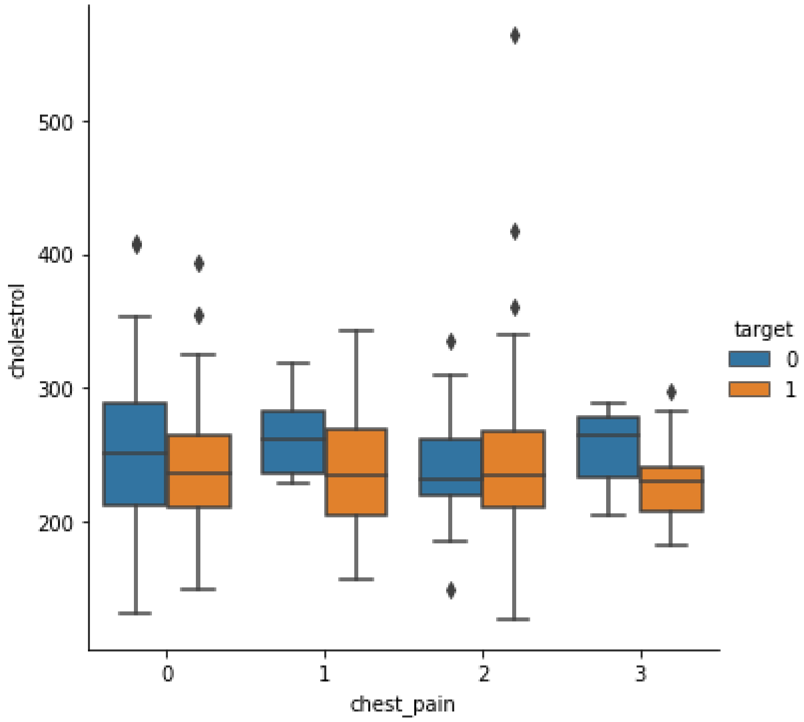
Outliers detection

In [31]:

```
sns.factorplot(x='chest_pain',y='cholestrol',hue='target',data=df,kind='box')
```

Out[31]:

```
<seaborn.axisgrid.FacetGrid at 0x239b4487c08>
```

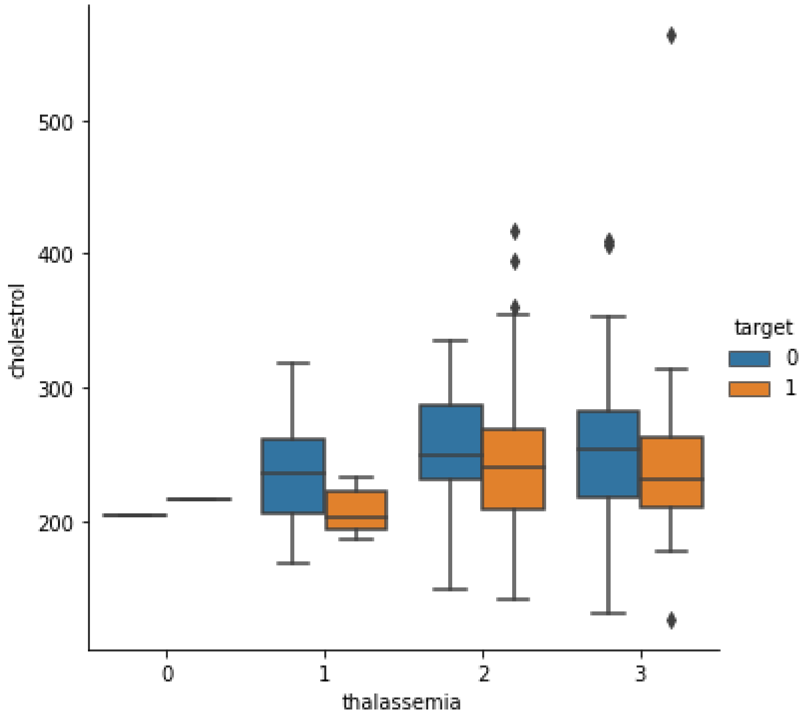


In [33]:

```
sns.factorplot(x='thalassemia',y='cholestrol',hue='target',data=df,kind='box')
```

Out[33]:

<seaborn.axisgrid.FacetGrid at 0x239b58e54c8>



Counting variables

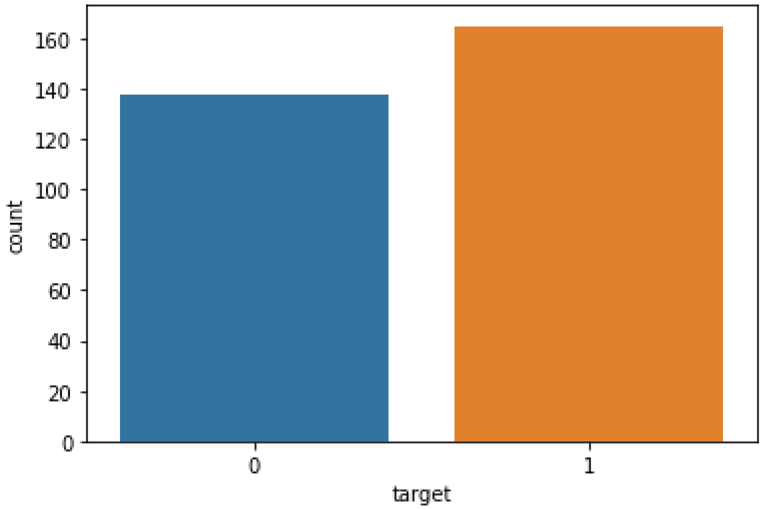
In [34]:

```
sns.countplot(df['target'])
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  FutureWarning
```

Out[34]:

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



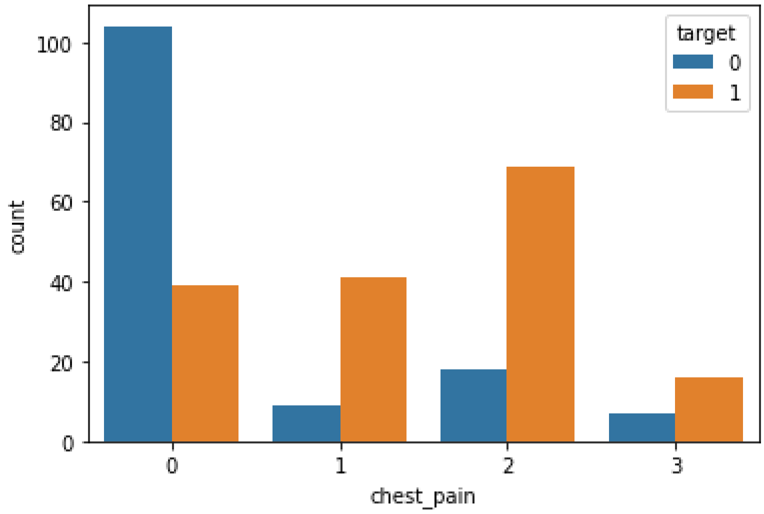
In [36]:

```
sns.countplot(df['chest_pain'],hue=df['target'])
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\_decorators.py:43: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data
`, and passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  FutureWarning
```

Out[36]:

```
<AxesSubplot:xlabel='chest_pain', ylabel='count'>
```



Analysing Distribution

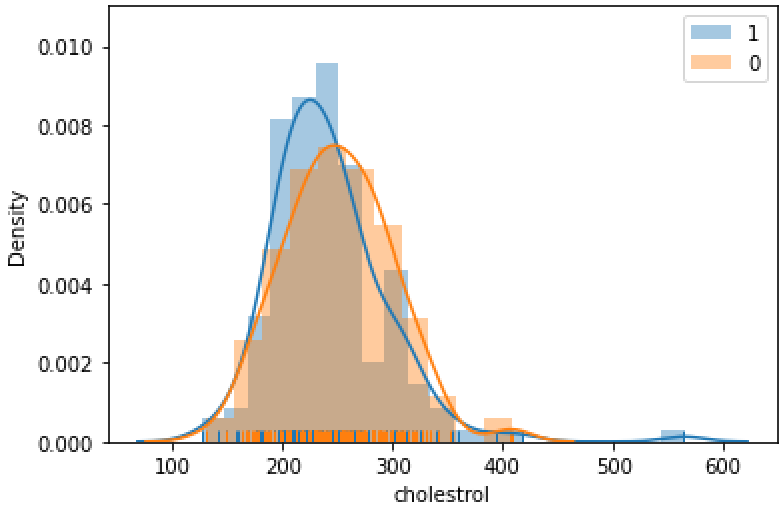
In [97]:

```
x=df['cholestrol'][df['target']==1]
y=df['cholestrol'][df['target']==0]
sns.distplot(x,rug=True,label=1);
sns.distplot(y,rug=True,label=0);
plt.legend()
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2551: FutureWarning: `distplot` is a deprecated function a
nd will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot`
(an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2055: FutureWarning: The `axis` variable is no longer used
and will be removed. Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2551: FutureWarning: `distplot` is a deprecated function a
nd will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot`
(an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2055: FutureWarning: The `axis` variable is no longer used
and will be removed. Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```

Out[97]:

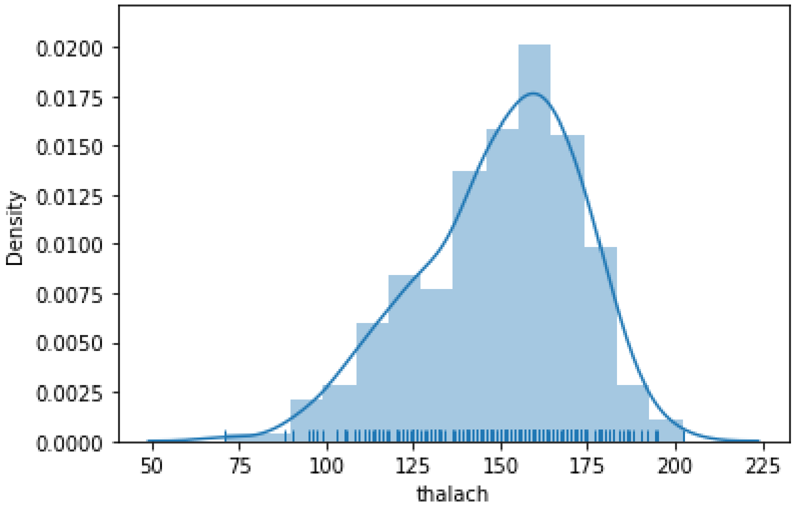
```
<matplotlib.legend.Legend at 0x239b9422208>
```



In [60]:

```
sns.distplot(df['thalach'],rug=True);
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2551: FutureWarning: `distplot` is a deprecated function a
nd will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot`
(an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\seaborn
\distributions.py:2055: FutureWarning: The `axis` variable is no longer used
and will be removed. Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```



Multi-dimensional comparision

In [61]:

```
df.head()
```

Out[61]:

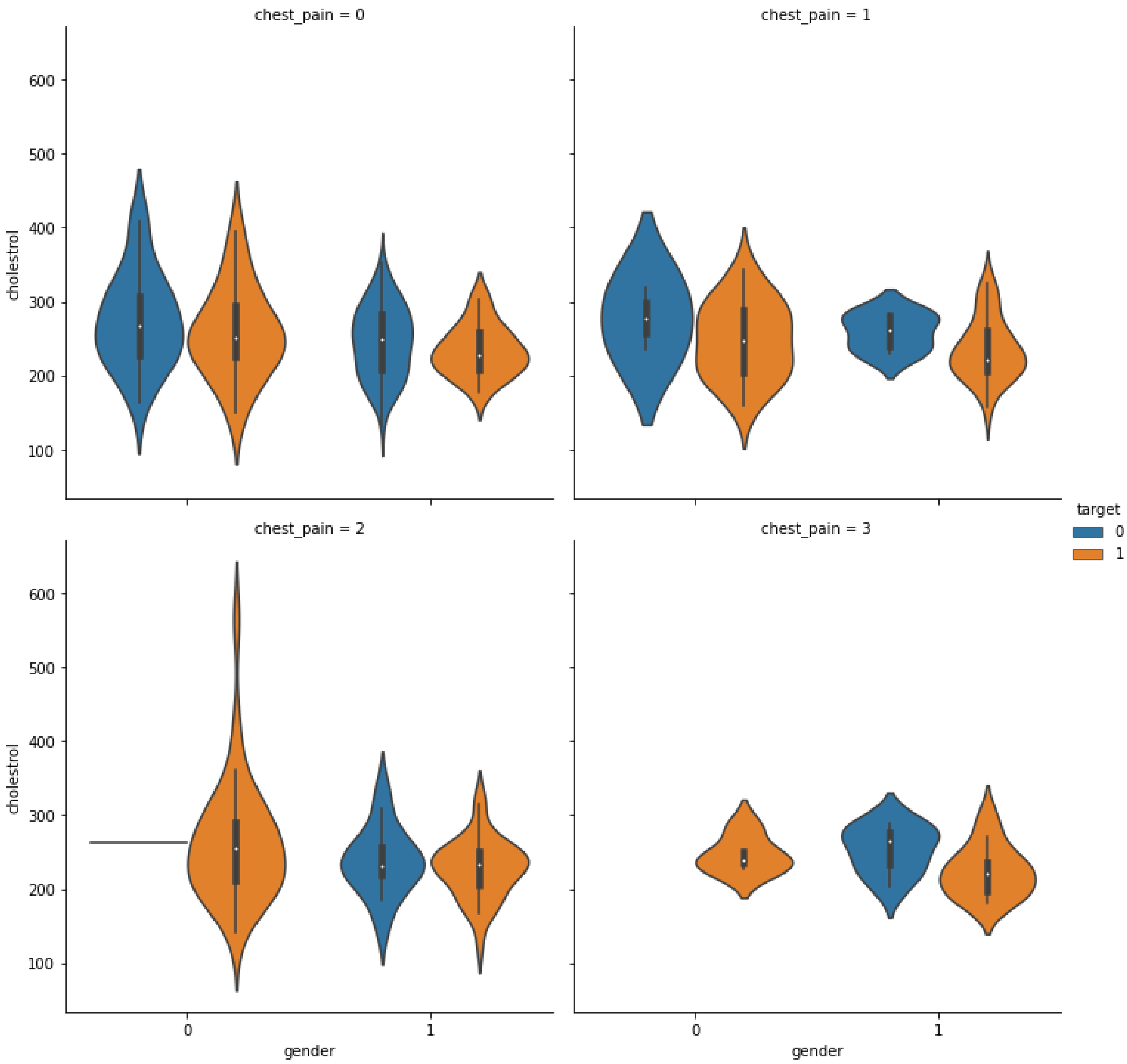
	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer_
0	63	1	3	145	233	1	0	150	
1	37	1	2	130	250	0	1	187	
2	41	0	1	130	204	0	0	172	
3	56	1	1	120	236	0	1	178	
4	57	0	0	120	354	0	1	163	

In [63]:

```
sns.catplot(x='gender',y='cholestrol',data=df,hue='target',col='chest_pain',col_wrap=2,kind
```

Out[63]:

```
<seaborn.axisgrid.FacetGrid at 0x239b7659888>
```

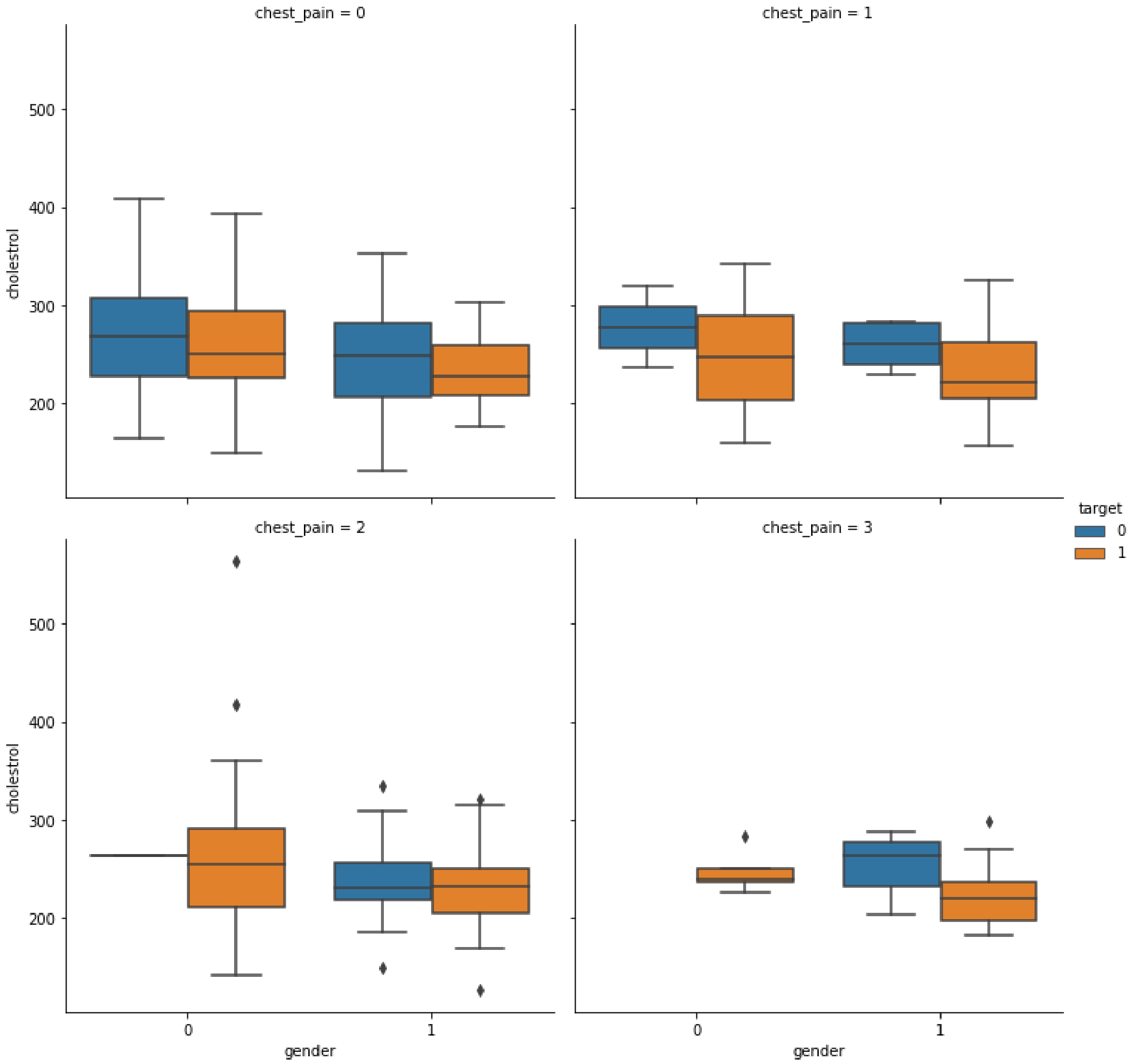


In [64]:

```
sns.catplot(x='gender',y='cholestrol',data=df,hue='target',col='chest_pain',col_wrap=2,kind
```

Out[64]:

```
<seaborn.axisgrid.FacetGrid at 0x239b7660208>
```



In [ ]:

```

```

In [ ]:

```

```

Now,the dataset given is to analyse the classifications,so we can use following classification methods,

1.LOGISTIC REGRESSION

2.RANDOM FOREST

3.NAIVE BAYES

4.KNN

5.SVM

In [65]:

```
df.head()
```

Out[65]:

	age	gender	chest_pain	rest_bps	cholestrol	fasting_blood_sugar	rest_ecg	thalach	exer_
0	63	1	3	145	233	1	0	150	
1	37	1	2	130	250	0	1	187	
2	41	0	1	130	204	0	0	172	
3	56	1	1	120	236	0	1	178	
4	57	0	0	120	354	0	1	163	

Now we have to separate the independent and dependent variables in the dataset

In [66]:

```
x_ind=df.drop(['target'],axis=1)
```

In [67]:

```
y_dep=df['target']
```

Before building the model,the dataset have to split by

means of train and test and train_test_split is used for spliting the dataset

Train_split = 80%

Test_split = 20%

In [69]:

```
from sklearn.model_selection import train_test_split
```

In [70]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
```

Now the dataset is splitted with respect to,

x_train = 80% of independent data

x_test = 20% of independent data

y_train = 80% of dependent data

y_test = 20% of dependent data

MODEL BUILDING

In [ ]:

```

```

LOGISTIC REGRESSION

Logistic regression is mainly used to predict binary classification.

Initially it will be predicting the values of 0 and 1 by default threshold value of 0.5.

To improve the accuracy we have to use ROC curve and to find the threshold value at which state it will gaining maximum accuracy

To find the p-values we have to use STATSMODELS library and LOGIT function

In [71]:

```
import statsmodels.api as sm
```

In [72]:

```
my_fit=sm.Logit(y_train,x_train)
```

In [73]:

```
p_value=my_fit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.364027
         Iterations 7
```

In [74]:

```
p_value.summary2()
```

Out[74]:

Model:	Logit	Pseudo R-squared:	0.470
Dependent Variable:	target	AIC:	202.1892
Date:	2021-01-05 11:30	BIC:	247.5454
No. Observations:	242	Log-Likelihood:	-88.095
Df Model:	12	LL-Null:	-166.34
Df Residuals:	229	LLR p-value:	2.7162e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0129	0.0214	0.6015	0.5475	-0.0291	0.0548
gender	-1.9024	0.5157	-3.6889	0.0002	-2.9132	-0.8917
chest_pain	0.8209	0.2061	3.9835	0.0001	0.4170	1.2248
rest_bps	-0.0177	0.0105	-1.6848	0.0920	-0.0383	0.0029
cholestrol	-0.0061	0.0041	-1.4863	0.1372	-0.0141	0.0019
fasting_blood_sugar	0.0443	0.5599	0.0791	0.9370	-1.0531	1.1417
rest_ecg	0.6516	0.3818	1.7066	0.0879	-0.0967	1.3998
thalach	0.0378	0.0097	3.9149	0.0001	0.0189	0.0567
exer_angina	-0.9961	0.4581	-2.1743	0.0297	-1.8940	-0.0982
old_peak	-0.3785	0.2403	-1.5753	0.1152	-0.8494	0.0924
slope	0.5233	0.3855	1.3574	0.1747	-0.2323	1.2789
ca	-0.7332	0.2123	-3.4543	0.0006	-1.1492	-0.3172
thalassemia	-0.7915	0.3113	-2.5427	0.0110	-1.4016	-0.1814

From this summary we have to find that which variables are significant or not,when the variables are not significant for 5% level of significance we have to drop that column and compare AIC value of the particular column with overall column.when overall < individual AIC we should not drop the column and when vice versa we have to drop the column and can have the clearnce upto 2% level.

In [82]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('age',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.364782
         Iterations 7
```

Out[82]:

Model:	Logit	Pseudo R-squared:	0.469
Dependent Variable:	target	AIC:	200.5546
Date:	2021-01-05 11:49	BIC:	242.4219
No. Observations:	242	Log-Likelihood:	-88.277
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	8.3075e-28
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
gender	-1.9040	0.5117	-3.7207	0.0002	-2.9070	-0.9010
chest_pain	0.8262	0.2055	4.0202	0.0001	0.4234	1.2290
rest_bps	-0.0149	0.0094	-1.5876	0.1124	-0.0332	0.0035
cholestrol	-0.0053	0.0039	-1.3702	0.1706	-0.0129	0.0023
fasting_blood_sugar	0.0772	0.5556	0.1390	0.8894	-1.0118	1.1662
rest_ecg	0.6662	0.3788	1.7586	0.0786	-0.0763	1.4086
thalach	0.0376	0.0096	3.8984	0.0001	0.0187	0.0565
exer_angina	-0.9673	0.4530	-2.1353	0.0327	-1.8552	-0.0794
old_peak	-0.3552	0.2353	-1.5093	0.1312	-0.8165	0.1061
slope	0.5455	0.3834	1.4231	0.1547	-0.2058	1.2969
ca	-0.7134	0.2102	-3.3937	0.0007	-1.1255	-0.3014
thalassemia	-0.7574	0.3048	-2.4847	0.0130	-1.3548	-0.1599

In [76]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('rest_bps',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.370042
         Iterations 7
```

Out[76]:

Model:	Logit	Pseudo R-squared:	0.462
Dependent Variable:	target	AIC:	203.1003
Date:	2021-01-05 11:43	BIC:	244.9676
No. Observations:	242	Log-Likelihood:	-89.550
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	2.7578e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	-0.0034	0.0188	-0.1824	0.8553	-0.0404	0.0335
gender	-1.8350	0.4995	-3.6732	0.0002	-2.8141	-0.8559
chest_pain	0.7713	0.2014	3.8304	0.0001	0.3766	1.1659
cholestrol	-0.0061	0.0039	-1.5398	0.1236	-0.0138	0.0017
fasting_blood_sugar	-0.0641	0.5452	-0.1176	0.9064	-1.1326	1.0044
rest_ecg	0.6244	0.3744	1.6677	0.0954	-0.1095	1.3583
thalach	0.0299	0.0081	3.6722	0.0002	0.0139	0.0458
exer_angina	-1.1259	0.4442	-2.5346	0.0113	-1.9965	-0.2553
old_peak	-0.4325	0.2337	-1.8507	0.0642	-0.8906	0.0255
slope	0.4735	0.3812	1.2421	0.2142	-0.2737	1.2207
ca	-0.7054	0.2094	-3.3692	0.0008	-1.1158	-0.2951
thalassemia	-0.8119	0.3061	-2.6525	0.0080	-1.4118	-0.2120

In [77]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('cholestrol',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.368520
         Iterations 7
```

Out[77]:

Model:	Logit	Pseudo R-squared:	0.464
Dependent Variable:	target	AIC:	202.3635
Date:	2021-01-05 11:44	BIC:	244.2307
No. Observations:	242	Log-Likelihood:	-89.182
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	1.9489e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0030	0.0197	0.1549	0.8769	-0.0355	0.0416
gender	-1.7037	0.4867	-3.5007	0.0005	-2.6576	-0.7499
chest_pain	0.8190	0.2043	4.0079	0.0001	0.4185	1.2195
rest_bps	-0.0183	0.0105	-1.7446	0.0811	-0.0388	0.0023
fasting_blood_sugar	0.0623	0.5505	0.1132	0.9099	-1.0166	1.1412
rest_ecg	0.7178	0.3735	1.9220	0.0546	-0.0142	1.4498
thalach	0.0324	0.0087	3.7446	0.0002	0.0154	0.0494
exer_angina	-1.0353	0.4489	-2.3064	0.0211	-1.9151	-0.1555
old_peak	-0.4179	0.2363	-1.7687	0.0769	-0.8810	0.0452
slope	0.4801	0.3814	1.2587	0.2081	-0.2675	1.2277
ca	-0.7065	0.2079	-3.3979	0.0007	-1.1140	-0.2990
thalassemia	-0.8474	0.3051	-2.7775	0.0055	-1.4454	-0.2494

In [78]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('fasting_blood_sugar',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.364040
         Iterations 7
```

Out[78]:

Model:	Logit	Pseudo R-squared:	0.470
Dependent Variable:	target	AIC:	200.1955
Date:	2021-01-05 11:45	BIC:	242.0628
No. Observations:	242	Log-Likelihood:	-88.098
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	7.0132e-28
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0130	0.0213	0.6124	0.5403	-0.0287	0.0548
gender	-1.8990	0.5140	-3.6943	0.0002	-2.9065	-0.8915
chest_pain	0.8233	0.2040	4.0351	0.0001	0.4234	1.2232
rest_bps	-0.0176	0.0104	-1.6853	0.0919	-0.0381	0.0029
cholestrol	-0.0061	0.0041	-1.4884	0.1367	-0.0141	0.0019
rest_ecg	0.6508	0.3817	1.7052	0.0882	-0.0972	1.3989
thalach	0.0378	0.0097	3.9164	0.0001	0.0189	0.0567
exer_angina	-0.9951	0.4582	-2.1719	0.0299	-1.8931	-0.0971
old_peak	-0.3804	0.2392	-1.5901	0.1118	-0.8492	0.0885
slope	0.5184	0.3803	1.3630	0.1729	-0.2270	1.2639
ca	-0.7312	0.2107	-3.4708	0.0005	-1.1440	-0.3183
thalassemia	-0.7968	0.3038	-2.6227	0.0087	-1.3923	-0.2014

In [79]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('rest_ecg',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.370131
         Iterations 7
```

Out[79]:

Model:	Logit	Pseudo R-squared:	0.462
Dependent Variable:	target	AIC:	203.1434
Date:	2021-01-05 11:45	BIC:	245.0107
No. Observations:	242	Log-Likelihood:	-89.572
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	2.8144e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0156	0.0211	0.7416	0.4583	-0.0257	0.0569
gender	-1.9436	0.5091	-3.8180	0.0001	-2.9413	-0.9458
chest_pain	0.7686	0.2002	3.8395	0.0001	0.3763	1.1610
rest_bps	-0.0172	0.0104	-1.6467	0.0996	-0.0377	0.0033
cholestrol	-0.0071	0.0041	-1.7539	0.0794	-0.0151	0.0008
fasting_blood_sugar	0.0200	0.5530	0.0362	0.9711	-1.0638	1.1039
thalach	0.0397	0.0096	4.1212	0.0000	0.0208	0.0586
exer_angina	-0.9675	0.4563	-2.1204	0.0340	-1.8618	-0.0732
old_peak	-0.3674	0.2375	-1.5471	0.1219	-0.8329	0.0981
slope	0.5503	0.3775	1.4577	0.1449	-0.1896	1.2902
ca	-0.7330	0.2124	-3.4509	0.0006	-1.1493	-0.3167
thalassemia	-0.7348	0.3042	-2.4154	0.0157	-1.3311	-0.1385

In [80]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('old_peak',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.369389
         Iterations 7
```

Out[80]:

Model:	Logit	Pseudo R-squared:	0.463
Dependent Variable:	target	AIC:	202.7841
Date:	2021-01-05 11:46	BIC:	244.6514
No. Observations:	242	Log-Likelihood:	-89.392
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	2.3761e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	**Std.Err.**	**z**	**P>|z|**	**[0.025**	**0.975]**
age	0.0078	0.0210	0.3685	0.7125	-0.0335	0.0490
gender	-2.0026	0.5070	-3.9500	0.0001	-2.9962	-1.0089
chest_pain	0.8009	0.2028	3.9488	0.0001	0.4034	1.1984
rest_bps	-0.0198	0.0102	-1.9497	0.0512	-0.0397	0.0001
cholestrol	-0.0068	0.0040	-1.7093	0.0874	-0.0147	0.0010
fasting_blood_sugar	0.1244	0.5528	0.2251	0.8219	-0.9590	1.2078
rest_ecg	0.6246	0.3720	1.6791	0.0931	-0.1045	1.3537
thalach	0.0384	0.0094	4.0717	0.0000	0.0199	0.0569
exer_angina	-1.0185	0.4514	-2.2564	0.0240	-1.9033	-0.1338
slope	0.8239	0.3301	2.4957	0.0126	0.1769	1.4709
ca	-0.7930	0.2085	-3.8038	0.0001	-1.2016	-0.3844
thalassemia	-0.7700	0.3029	-2.5420	0.0110	-1.3637	-0.1763

In [81]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('slope',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.367773
         Iterations 7
```

Out[81]:

Model:	Logit	Pseudo R-squared:	0.465
Dependent Variable:	target	AIC:	202.0023
Date:	2021-01-05 11:46	BIC:	243.8696
No. Observations:	242	Log-Likelihood:	-89.001
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	1.6439e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0157	0.0214	0.7341	0.4629	-0.0263	0.0577
gender	-1.8216	0.5040	-3.6143	0.0003	-2.8094	-0.8338
chest_pain	0.7998	0.2042	3.9175	0.0001	0.3996	1.1999
rest_bps	-0.0167	0.0105	-1.5989	0.1098	-0.0373	0.0038
cholestrol	-0.0057	0.0041	-1.3976	0.1622	-0.0136	0.0023
fasting_blood_sugar	-0.0669	0.5454	-0.1226	0.9024	-1.1359	1.0022
rest_ecg	0.6794	0.3803	1.7864	0.0740	-0.0660	1.4247
thalach	0.0410	0.0094	4.3490	0.0000	0.0225	0.0594
exer_angina	-1.0566	0.4577	-2.3086	0.0210	-1.9536	-0.1596
old_peak	-0.5387	0.2126	-2.5341	0.0113	-0.9553	-0.1220
ca	-0.6767	0.2029	-3.3355	0.0009	-1.0743	-0.2790
thalassemia	-0.8080	0.3117	-2.5919	0.0095	-1.4190	-0.1970

In [83]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
x_train.drop('slope',inplace=True,axis=1)
my_fit=sm.Logit(y_train,x_train)
p_value=my_fit.fit()
p_value.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.367773
         Iterations 7
```

Out[83]:

Model:	Logit	Pseudo R-squared:	0.465
Dependent Variable:	target	AIC:	202.0023
Date:	2021-01-05 11:51	BIC:	243.8696
No. Observations:	242	Log-Likelihood:	-89.001
Df Model:	11	LL-Null:	-166.34
Df Residuals:	230	LLR p-value:	1.6439e-27
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
age	0.0157	0.0214	0.7341	0.4629	-0.0263	0.0577
gender	-1.8216	0.5040	-3.6143	0.0003	-2.8094	-0.8338
chest_pain	0.7998	0.2042	3.9175	0.0001	0.3996	1.1999
rest_bps	-0.0167	0.0105	-1.5989	0.1098	-0.0373	0.0038
cholestrol	-0.0057	0.0041	-1.3976	0.1622	-0.0136	0.0023
fasting_blood_sugar	-0.0669	0.5454	-0.1226	0.9024	-1.1359	1.0022
rest_ecg	0.6794	0.3803	1.7864	0.0740	-0.0660	1.4247
thalach	0.0410	0.0094	4.3490	0.0000	0.0225	0.0594
exer_angina	-1.0566	0.4577	-2.3086	0.0210	-1.9536	-0.1596
old_peak	-0.5387	0.2126	-2.5341	0.0113	-0.9553	-0.1220
ca	-0.6767	0.2029	-3.3355	0.0009	-1.0743	-0.2790
thalassemia	-0.8080	0.3117	-2.5919	0.0095	-1.4190	-0.1970

In [ ]:

Since every AIC values are at 2% clearance level so we no need to drop those columns

In [84]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,test_size=0.2,random_state=5)
```

In [85]:

```
from sklearn.linear_model import LogisticRegression
```

In [86]:

```
model_lr=LogisticRegression()
model_lr_fit=model_lr.fit(x_train,y_train)
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\sklearn
\linear_model\_logistic.py:764: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [87]:

```
y_pred_lr=model_lr_fit.predict(x_test)
```

In [88]:

```
y_pred_lr
```

Out[88]:

```
array([1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0], dtype=int64)
```

In [89]:

```
# To find the accuracy and the errors we have to import confusion_matrix and accuracy_score
```

In [90]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [91]:

```
confusion_matrix(y_test,y_pred_lr)
```

Out[91]:

```
array([[27,  3],
       [ 2, 29]], dtype=int64)
```

In [92]:

```
acc_lr=accuracy_score(y_test,y_pred_lr)
```

In [93]:

```
print('The accuracy score for Logistic regression model using gini method :',acc_lr*100)
```

```
The accuracy score for Logistic regression model using gini method : 91.8032
7868852459
```

ROC curve

In [98]:

```
from sklearn import metrics
```

In [99]:

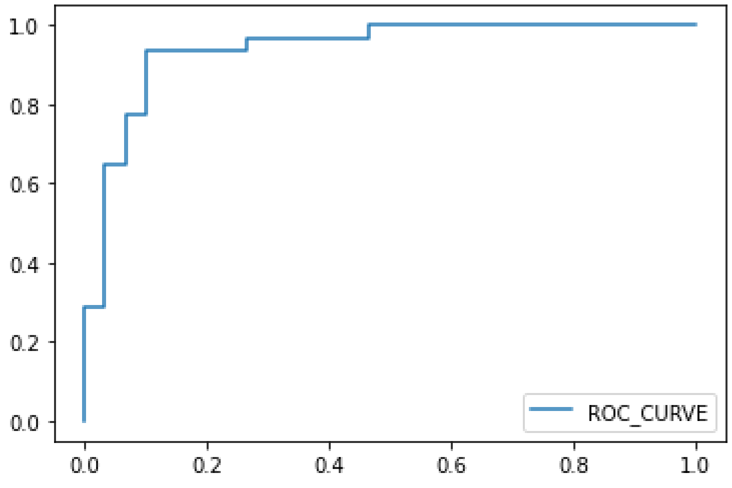
```
y_pred_prob=model_lr.predict_proba(x_test)[:,1]
fpr,tpr,_=metrics.roc_curve(y_test,y_pred_prob)
```

In [100]:

```
plt.plot(fpr,tpr,label='ROC_CURVE')
plt.legend()
```

Out[100]:

```
<matplotlib.legend.Legend at 0x239b94ef988>
```



In [101]:

```
update_roc=LogisticRegression(class_weight='balanced')
update_roc.fit(x_train,y_train)
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\sklearn
\linear_model\_logistic.py:764: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[101]:

```
LogisticRegression(class_weight='balanced')
```

In [267]:

```
thr=np.where(update_roc.predict_proba(x_test)[:,1]>0.48995,1,0)
```

In [268]:

```
acc_log=accuracy_score(y_test,thr)
```

In [269]:

```
print('The accuracy_score for Logistic regression using ROC curve is :',acc_log*100)
```

```
The accuracy_score for Logistic regression using ROC curve is : 88.524590163
93442
```

In [ ]:

```

```

In [ ]:

```

```

NAIVE BAYES

This model is based on bayes theorom

this model will gives the accuracy based on individual probability

This model is used only for classification

In [159]:

```
from sklearn.naive_bayes import GaussianNB
```

In [160]:

```
model_nb=GaussianNB()
model_nb.fit(x_train,y_train)
```

Out[160]:

```
GaussianNB()
```

This works based on Gaussian or normal distribution

In [161]:

```
y_pred_nb=model_nb.predict(x_test)
```

In [162]:

```
confusion_matrix(y_test,y_pred_nb)
```

Out[162]:

```
array([[26,  4],
       [ 4, 27]], dtype=int64)
```

In [163]:

```
acc_nb=accuracy_score(y_test,y_pred_nb)
```

In [164]:

```
print('The accuracy score for Naive Bayes model is :',acc_nb*100)
```

```
The accuracy score for Naive Bayes model is : 86.88524590163934
```

RANDOM FOREST

It is used for both classification and Regression analysis

Random forest can have many number of trees and it is predicted by means of GINI or ENTROPY

Both GINI and ENTROPY methods are used to minimize the impurities

In [165]:

```
from sklearn.ensemble import RandomForestClassifier
```

criteria = GINI

In [166]:

```
model_rf=RandomForestClassifier(n_estimators=5,random_state=5)
```

In [167]:

```
model_rf.fit(x_train,y_train)
```

Out[167]:

```
RandomForestClassifier(n_estimators=5, random_state=5)
```

In [168]:

```
y_pred_rf=model_rf.predict(x_test)
```

In [169]:

```
y_pred_rf
```

Out[169]:

```
array([0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0], dtype=int64)
```

To find the accuracy and the errors we have to import confusion_matrix and accuracy_score

In [170]:

```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [171]:

```
confusion_matrix(y_test,y_pred_rf)
```

Out[171]:

```
array([[26,  4],
       [ 2, 29]], dtype=int64)
```

In [172]:

```
acc_rf=accuracy_score(y_test,y_pred_rf)
```

Accuracy score using GINI in RANDOMFOREST

In [173]:

```
print('The accuracy score for random forest model using gini method :',acc_rf*100)
```

```
The accuracy score for random forest model using gini method : 90.1639344262
295
```

HYPER PARAMETERS

We have to import Randomized search cv library to find the hyper parameters

In [176]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [181]:

```
parameters={'n_estimators':(100,200,300,400,800),'criterion':('gini','entropy'),
           'max_features':('auto','sqrt','log2'),'min_samples_split':(2,4,6,8)}
```

In [182]:

```
rf_grid=RandomizedSearchCV(RandomForestClassifier(),param_distributions=parameters,cv=5)
```

In [183]:

```
rf_grid.fit(x_train,y_train)
```

Out[183]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(),
                   param_distributions={'criterion': ('gini', 'entropy'),
                                        'max_features': ('auto', 'sqrt',
                                                         'log2'),
                                        'min_samples_split': (2, 4, 6, 8),
                                        'n_estimators': (100, 200, 300, 400,
                                                         800)})
```

In [180]:

```
rf_grid.best_estimator_
```

Out[180]:

```
RandomForestClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=8, n_estimators=300)
```

In [184]:

```
best_model=RandomForestClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=8, n_estimators=300)
```

In [185]:

```
best_model.fit(x_train,y_train)
```

Out[185]:

```
RandomForestClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=8, n_estimators=300)
```

In [186]:

```
y_pred_rfhp=best_model.predict(x_test)
```

In [187]:

```
confusion_matrix(y_test,y_pred_rfhp)
```

Out[187]:

```
array([[25,  5],
       [ 2, 29]], dtype=int64)
```

In [188]:

```
acc_rf_hp=accuracy_score(y_test,y_pred_rfhp)
```

In [189]:

```
print('The accuracy of Random forest model using hyperparameters :',acc_rf_hp*100)
```

```
The accuracy of Random forest model using hyperparameters : 88.5245901639344
2
```

In [ ]:

```

```

KNEIGHBORS CLASSIFIER

KNN is mainly used to find the correct category to which the output have to fall in the particular category

TWO METHODS IN KNN:

1.Square Root

2.Error method

From sklearn importing the kneighbors classifier model

In [190]:

```
from sklearn.neighbors import KNeighborsClassifier
```

WE have to normalize the X_train and x_test values using standard scalar to get the values within the range and to avoid the spread

In [191]:

```
from sklearn.preprocessing import StandardScaler
```

In [192]:

```
norm=StandardScaler()
```

In [193]:

```
x_train1=norm.fit_transform(x_train)
x_test1=norm.fit_transform(x_test)
```

SQUARE ROOT METHOD

In [194]:

```
(x_train1.shape[0])**(0.5)
```

Out[194]:

```
15.556349186104045
```

The square root of x_train rows should be used in n_neighbors for square root method

In [195]:

```
mod_knn=KNeighborsClassifier(n_neighbors=15 ,p=2,metric='euclidean')
mod_knn.fit(x_train1,y_train)
```

Out[195]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=15)
```

In [196]:

```
y_pred_knnsq=mod_knn.predict(x_test1)
```

In [197]:

```
confusion_matrix(y_test,y_pred_knnsq)
```

Out[197]:

```
array([[23,  7],
       [ 2, 29]], dtype=int64)
```

In [198]:

```
acc_knnsq=accuracy_score(y_test,y_pred_knnsq)
```

In [199]:

```
print('The accuracy score of KNN model using Square method is :',(acc_knnsq)*100)
```

```
The accuracy score of KNN model using Square method is : 85.24590163934425
```

ERROR METHOD

In [202]:

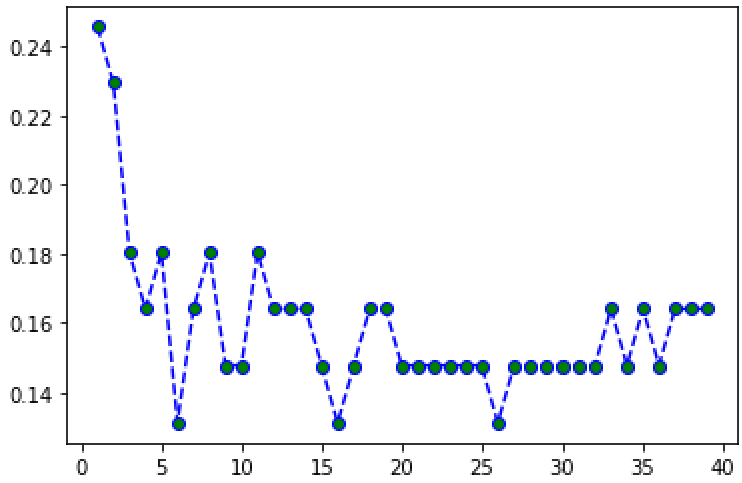
```
error_pred=[]
for i in range (1,40):
    knn_new=KNeighborsClassifier(n_neighbors=i)
    knn_new.fit(x_train1,y_train)
    y_pred_knnerr=knn_new.predict(x_test1)
    error_pred.append(np.mean(y_test != y_pred_knnerr))
```

In [204]:

```
plt.plot(range(1,40),error_pred,color='Blue',markerfacecolor='Green',marker='o',linestyle='
```

Out[204]:

```
[<matplotlib.lines.Line2D at 0x239ba9fb448>]
```



In [205]:

```
mod_knn_err=KNeighborsClassifier(n_neighbors=9 ,p=2,metric='euclidean')
mod_knn_err.fit(x_train1,y_train)
```

Out[205]:

```
KNeighborsClassifier(metric='euclidean', n_neighbors=9)
```

In [206]:

```
y_pred_knnerr=mod_knn.predict(x_test1)
```

In [207]:

```
confusion_matrix(y_test,y_pred_knnerr)
```

Out[207]:

```
array([[23,  7],
       [ 2, 29]], dtype=int64)
```

In [208]:

```
acc_knnsq=accuracy_score(y_test,y_pred_knnerr)
```

In [209]:

```
print('The accuracy score of KNN model using error method is :',(acc_knnsq)*100)
```

```
The accuracy score of KNN model using error method is : 85.24590163934425
```

SUPPORT VECTOR MACHINE

It is used to find the correct category for which the values to be predicted.

It is used only for classification

It will be prediction using margin and data points

In [210]:

```
from sklearn.svm import SVC
```

In [211]:

```
model_svm=SVC(kernel='linear')
model_svm.fit(x_train1,y_train)
```

Out[211]:

```
SVC(kernel='linear')
```

In [212]:

```
model_svm.n_support_
```

Out[212]:

```
array([51, 52])
```

In [217]:

```
y_pred_svm = model_svm.predict(x_test1)
```

In [218]:

```
confusion_matrix(y_test,y_pred_svm)
```

Out[218]:

```
array([[26,  4],
       [ 2, 29]], dtype=int64)
```

In [219]:

```
acc_svm=accuracy_score(y_test,y_pred_svm)
```

In [220]:

```
print('The accuracy score for SVM is :', acc_svm)
```

```
The accuracy score for SVM is : 0.9016393442622951
```

In [ ]:

```

```

Ensembling -VOTING CLASSIFIER

In [221]:

```
model1 = LogisticRegression()
model2 = RandomForestClassifier()
model3 = GaussianNB()
model4 = KNeighborsClassifier()
model5 = SVC()
```

In [222]:

```
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
model4.fit(x_train,y_train)
model5.fit(x_train,y_train)
```

```
C:\Users\VIMAL MADHAN\AppData\Roaming\Python\Python37\site-packages\sklearn
\linear_model\_logistic.py:764: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[222]:

```
SVC()
```

In [223]:

```
pred1=model1.predict(x_test)
pred2=model2.predict(x_test)
pred3=model3.predict(x_test)
pred4=model4.predict(x_test)
pred5=model5.predict(x_test)
```

In [224]:

```
import statistics as st
final_predict=np.array([])
for i in range(0,len(x_test)):
    final_predict=np.append(final_predict,st.mode([pred1[i],pred2[i],pred3[i],pred4[i],pred
```

In [225]:

```
final_predict=pd.DataFrame(final_predict)
```

In [230]:

```
final_predict.head(10)
```

Out[230]:

	0
0	1.0
1	0.0
2	0.0
3	0.0
4	1.0
5	0.0
6	1.0
7	0.0
8	1.0
9	1.0

In [231]:

```
x_test.shape
```

Out[231]:

```
(61, 13)
```

In [232]:

```
y_test.shape
```

Out[232]:

```
(61,)
```

In [233]:

```
final_predict.shape
```

Out[233]:

```
(61, 1)
```

In [234]:

```
confusion_matrix(y_test,final_predict)
```

Out[234]:

```
array([[25,  5],
       [ 3, 28]], dtype=int64)
```

In [235]:

```
acc_vc=accuracy_score(y_test,final_predict)
```

In [236]:

```
print('The accuracy score for voting classifier emsembling :',acc_vc*100)
```

```
The accuracy score for voting classifier emsembling : 86.88524590163934
```

In [ ]:

```

```

MAXIMUM ACCURACY

In [270]:

```
accuracy = pd.DataFrame({'model':['Logistic_Regression','Naive_Bayes','Random_Forest','KNN'
                        'accuracy_score':[acc_log,acc_nb,acc_rf_hp,acc_knnsq,acc_svm]})
```

In [271]:

```
accuracy
```

Out[271]:

	model	accuracy_score
0	Logistic_Regression	0.885246
1	Naive_Bayes	0.868852
2	Random_Forest	0.885246
3	KNN	0.852459
4	SVM	0.901639

In [272]:

```
accuracy['accuracy_score'].max()
```

Out[272]:

```
0.9016393442622951
```

HIGH ACCURACY MODEL

In [273]:

```
accuracy.nlargest(1, ['accuracy_score'])
```

Out[273]:

	model	accuracy_score
4	SVM	0.901639

In [ ]:

```

```