JAIN COLLEGE OF ENGINEERING, BELGAUM

BATCH NO : 14

NAME : SHREYA PAKHARE

GMAIL : shreyapakhare20@gmail.com

CONTENT

1.	Table of figures/graph
2.	Abstract
3.	Chapters
4.	Complete code
5.	Conclusion

TABLE OF FIGURES/GRAPH

Figure 3.1…… 8
Figure 3.2…… 8
Figure 3.3…… 9
Figure 3.4…… 14
Figure 3.5…… 15

ABSTRACT

The healthcare industry collects huge amounts of healthcare data which, unfortunately, are not "mined" to discover hidden information for effective decision making. Discovery of hidden patterns and relationships often goes unexploited. Advanced data mining techniques can help remedy this situation. This research has developed a prototype Intelligent Heart Disease Prediction System (IHDPS) using data mining techniques, namely, Logistic Regression, Decision Trees, Naive Bayes and Neural Network. Results show that each technique has its unique strength in realizing the objectives of the defined mining goals. IHDPS can answer complex "what if" queries which traditional decision support systems cannot. Using medical profiles such as age, sex, blood pressure and blood sugar it can predict the likelihood of patients getting a heart disease. It enables significant knowledge, e.g., patterns, between medical factors related to heart disease, to be established.

CHAPTER-1

INTRODUCTION

Cardiovascular diseases (CVDs) are the leading cause of death globally, taking an estimated 17.9 million lives each year. CVDs are a group of disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease and other conditions. More than four out of five CVD deaths are due to heart attacks and strokes, and one third of these deaths occur prematurely in people under 70 years of age.

The most important behavioural risk factors of heart disease and stroke are unhealthy diet, physical inactivity, tobacco use and harmful use of alcohol. The effects of behavioural risk factors may show up in individuals as raised blood pressure, raised blood glucose, raised blood lipids, and overweight and obesity. These “intermediate risks factors” can be measured in primary care facilities and indicate an increased risk of heart attack, stroke, heart failure and other complications.

Cessation of tobacco use, reduction of salt in the diet, eating more fruit and vegetables, regular physical activity and avoiding harmful use of alcohol have been shown to reduce the risk of cardiovascular disease. Health policies that create conducive environments for making healthy choices affordable and available are essential for motivating people to adopt and sustain healthy behaviours.

Identifying those at highest risk of CVDs and ensuring they receive appropriate treatment can prevent premature deaths. Access to noncommunicable disease medicines and basic health technologies in all primary health care facilities is essential to ensure that those in need receive treatment and counselling.

Discovering these patterns from the patient’s health and analysing them with the help of various machine learning algorithms, can prevent the deaths by giving the appropriate treatment.

CHAPTER-2

SOFTWARE REQUIREMENTS

Following are the software requirements necessary for the project:

1. Python 3.6 (64 – bit) and above
2. Anaconda
3. Windows 7/10, any Linux OS flavours.

2.1 Some of the libraries used for the project:

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Sklearn

CHAPTER-3

IMPLEMENTATION

Now, let's implement the data and will try getting the insights from the given dataset.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import accuracy_score
```

- At start, we are importing the required libraries such as numpy, pandas, matplotlib for exploratory analysis, sklearn library for performing logistic regression and statistics calculations.

```
In [2]: data = pd.read_csv("heart.csv")
```

- Here we're reading the csv file using pandas and storing it in the variable data.

```
In [3]: data.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

- We get the first 5 data from the dataset used using the head() function.

- Using the describe

```
In [4]: data.describe()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.3
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.6
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0

() function we get the overlook about the statistics results.

```
In [5]: data["target"].unique()
Out[5]: array([1, 0], dtype=int64)
```

- Using the unique() function we try to find the various unique values from the dependent variable i.e., from the target. The target column consists of data 0 and 1. 0 symbolises that the person doesn't has heart disease and 1symbolises that the person has heart disease.

```
In [16]: sns.countplot(x='target',data=data)
Out[16]: <AxesSubplot:xlabel='target', ylabel='count'>
```

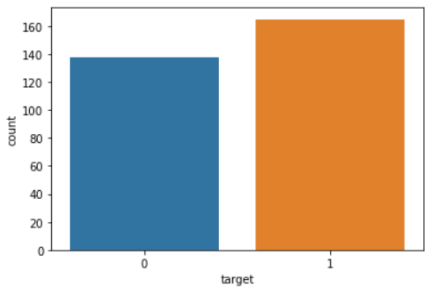

Figure 3.1

- We visualize and check various plots using the countplot() in our target feature and get to know that our dataset consists of people with more number of heart patients.

```
In [17]: sns.countplot(x='target',hue='sex',data=data)
Out[17]: <AxesSubplot:xlabel='target', ylabel='count'>
```

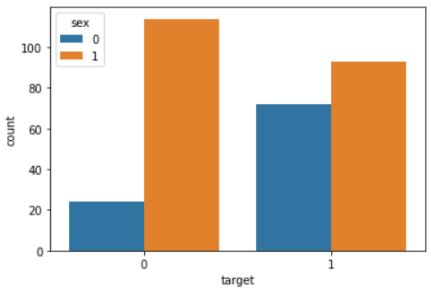

Figure 3.2

- Also, we visualize the number of patients of both the gender.

```
In [18]: sns.pairplot(data)
Out[18]: <seaborn.axisgrid.PairGrid at 0x240ce755fa0>
```

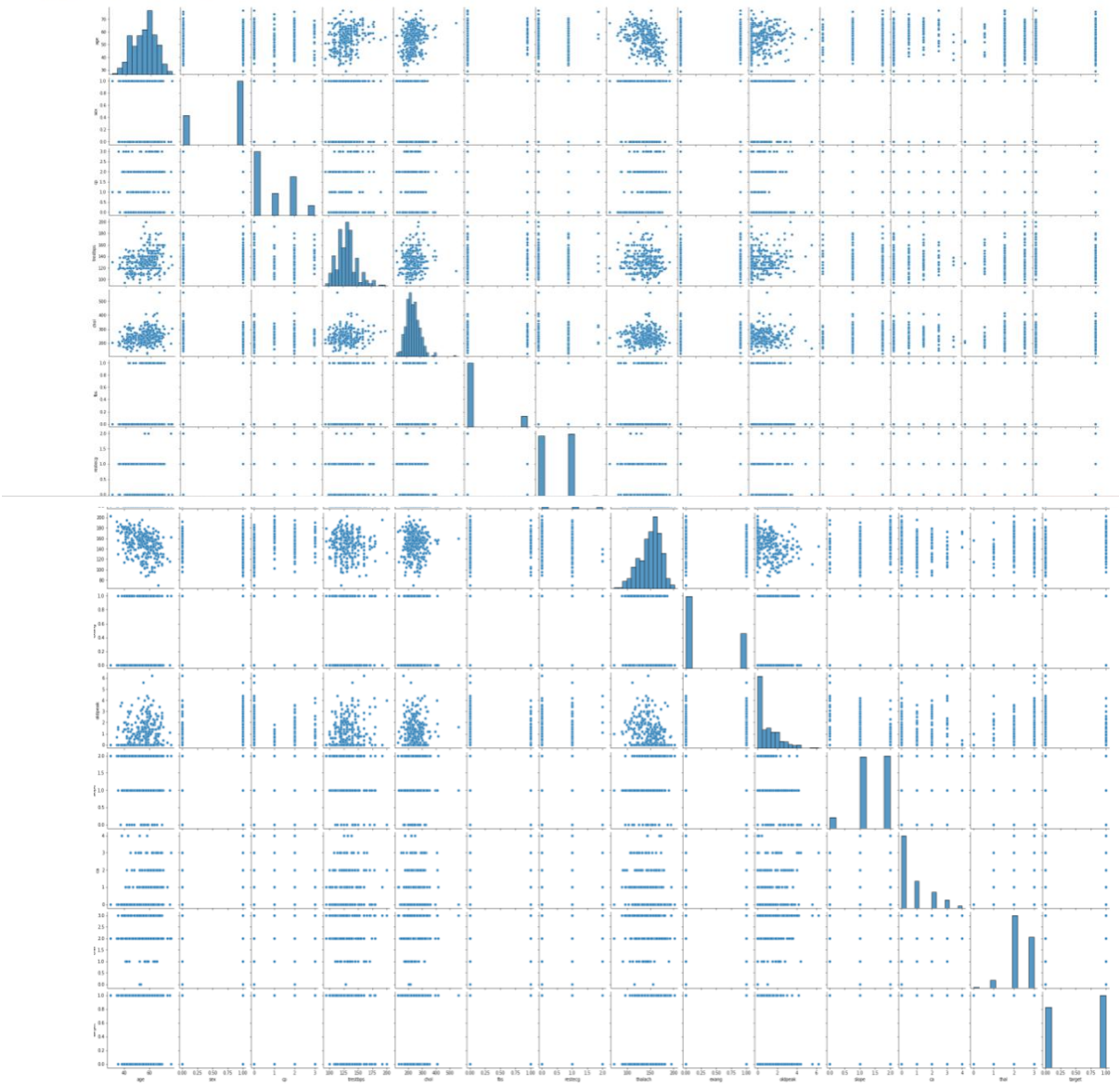

Figure 3.3

- The pairplot() helps us to understand the dependency of each independent feature with one another.

```
In [6]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

- Using the info() function we check if we have any null values into the dataset. Here in our dataset, we find that we don't have any null values. Hence, we can go ahead with creating a model using the Logistic Regression.

```
In [7]: X = data.iloc[:,:-1]
```

```
In [8]: Y = data.iloc[:,-1]
```

- Now, let's segregate the independent variables and dependent variables.
- X variable consists of all the independent features such as age, sex, cholesterol etc and Y variable consists of dependent feature which is target column in our dataset.

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

- Using the train_test_split() function from the sklearn library we divide the independent features and the dependent feature into train dataset and test dataset.
- Here in our model, we divide data into 80% into training data and 20% into testing data.

```
In [10]: from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Out[10]: LogisticRegression()
```

- Here, we import the LogisticRegression from the sklearn library and we try to fit the train dataset from X and Y variable.

In [11]:
```
print (clf.intercept_, clf.coef_)
```

```
[0.0293818] [[ 0.01048323 -1.21876918  0.77587069 -0.01152404 -0.00182218 -0.00163632
   0.546386    0.02752004 -0.84005435 -0.67763383  0.62861569 -0.74957961
  -1.02303045]]
```

- We have our intercept value = 0.0293818 and the co-efficient of the independent features.
- The co-efficient helps us to understand the dependability of the feature on the target variable.

In [12]:
```
pred = clf.predict(X_test)
print ('Accuracy from sk-learn: {0}'.format(clf.score(X_test, y_test)))
```

```
Accuracy from sk-learn: 0.8852459016393442
```

- Now, we use the predict function on the test data of independent features.
- And then compare the accuracy by comparing the predicted value from the test dataset and the actual values of dependent features stored in the y_test.
- As we have obtained the accuracy value of 88.52% which is quiet good performance obtained from the model.

Now, let's check the performance of the metrics.

In [13]:
```
from sklearn.metrics import recall_score, auc
```

In [14]:
```
recall_score(pred, y_test)
```

Out[14]:
```
0.8787878787878788
```

- With the help of sklearn.metrics we can use various statistical measure's and validate the results.
- We have obtained the recall score to be 87.87% which is quite significant.

In [15]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

- Using the classification_report() function we check various measures such as precision, recall, f1-score and support.

In [20]:
```
from statsmodels.tools import add_constant as add_constant
heart_df_constant = add_constant(data)
heart_df_constant.head()
```

Out[20]:

	const	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	1.0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	1.0	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	1.0	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	1.0	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	1.0	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [21]:
```
import scipy.stats as st
import statsmodels.api as sm
st.chisqprob = lambda chisq, df: st.chi2.sf(chisq, df)
cols=heart_df_constant.columns[:-1]
model=sm.Logit(data.target,heart_df_constant[cols])
result=model.fit()
result.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.348904
         Iterations 7
```

Out[21]:

Logit Regression Results

Dep. Variable:	target	**No. Observations:**	303
Model:	Logit	**Df Residuals:**	289
Method:	MLE	**Df Model:**	13
Date:	Tue, 03 Aug 2021	**Pseudo R-squ.:**	0.4937
Time:	09:21:10	**Log-Likelihood:**	-105.72
converged:	True	**LL-Null:**	-208.82
Covariance Type:	nonrobust	**LLR p-value:**	7.262e-37

	coef	std err	z	P>|z|	[0.025	0.975]
const	3.4505	2.571	1.342	0.180	-1.590	8.490
age	-0.0049	0.023	-0.212	0.832	-0.050	0.041
sex	-1.7582	0.469	-3.751	0.000	-2.677	-0.839
cp	0.8599	0.185	4.638	0.000	0.496	1.223
trestbps	-0.0195	0.010	-1.884	0.060	-0.040	0.001
chol	-0.0046	0.004	-1.224	0.221	-0.012	0.003
fbs	0.0349	0.529	0.066	0.947	-1.003	1.073
restecg	0.4663	0.348	1.339	0.181	-0.216	1.149
thalach	0.0232	0.010	2.219	0.026	0.003	0.044
exang	-0.9800	0.410	-2.391	0.017	-1.783	-0.177
oldpeak	-0.5403	0.214	-2.526	0.012	-0.959	-0.121
slope	0.5793	0.350	1.656	0.098	-0.106	1.265
ca	-0.7733	0.191	-4.051	0.000	-1.147	-0.399
thal	-0.9004	0.290	-3.104	0.002	-1.469	-0.332

- Here, we chcek for statistical measures of each independent features such as co-efficient, standard error, z-value, P-value.

In [22]:
```
def back_feature_elem (data_frame,dep_var,col_list):
    """ Takes in the dataframe, the dependent variable and a list of column names, runs the regression repeatedly eleminating fea
    P-value above alpha one at a time and returns the regression summary with all p-values below alpha"""

    while len(col_list)>0 :
        model=sm.Logit(dep_var,data_frame[col_list])
        result=model.fit(disp=0)
        largest_pvalue=round(result.pvalues,3).nlargest(1)
        if largest_pvalue[0]<(0.05):
            return result
            break
        else:
            col_list=col_list.drop(largest_pvalue.index)

result=back_feature_elem(heart_df_constant,data.target,cols)
```

- For the independent features having P-value greater than 0.05 are been dropped and result is stored back into the result variable.

In [23]:
```
result.summary()
```

Out[23]:

Logit Regression Results

Dep. Variable:	target	**No. Observations:**	303
Model:	Logit	**Df Residuals:**	296
Method:	MLE	**Df Model:**	6
Date:	Tue, 03 Aug 2021	**Pseudo R-squ.:**	0.4651
Time:	09:21:16	**Log-Likelihood:**	-111.71
converged:	True	**LL-Null:**	-208.82
Covariance Type:	nonrobust	**LLR p-value:**	3.209e-39

	coef	std err	z	P>|z|	[0.025	0.975]
sex	-1.3898	0.405	-3.431	0.001	-2.184	-0.596
cp	0.7861	0.174	4.509	0.000	0.444	1.128
thalach	0.0261	0.004	5.905	0.000	0.017	0.035
exang	-1.0130	0.376	-2.695	0.007	-1.750	-0.276
oldpeak	-0.7262	0.176	-4.130	0.000	-1.071	-0.382
ca	-0.7053	0.173	-4.087	0.000	-1.043	-0.367
thal	-0.8674	0.259	-3.351	0.001	-1.375	-0.360

- Here we can see that the we have obtained few independent features which hae P-value less than 0.05.

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

- The dataset is separated into training dataset and test dataset.

In [25]:
```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

- We use the LogisticRegression() function from the sklearn library and fit the training dataset and predict the values.

In [26]:
```
accuracy_score(y_test, y_pred)
```

Out[26]:
```
0.8852459016393442
```

- Also, as above we had obtained the accuracy to be 88.5% here through choosing some of the required features, we have obtained the same results. Hence, we can apply this model.

In [27]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[27]: <AxesSubplot:>

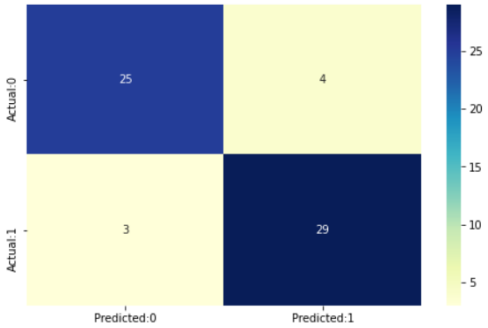

Figure 3.4

- Now, we calculate the confusion matrix which is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

In [28]:
```
TN=cm[0,0]
TP=cm[1,1]
FN=cm[1,0]
FP=cm[0,1]
sensitivity=TP/float(TP+FN)
specificity=TN/float(TN+FP)
```

- Here, we calculate the sensitivity and specificity.
- Sensitivity measures how apt the model is to detecting events in the positive class.
- Specificity measures how exact the assignment to the positive class is.

In [29]:
```
print('The acuuracy of the model = TP+TN/(TP+TN+FP+FN) = ',(TP+TN)/float(TP+TN+FP+FN),'\n',

'The Missclassification = 1-Accuracy = ',1-((TP+TN)/float(TP+TN+FP+FN)),'\n',

'Sensitivity or True Positive Rate = TP/(TP+FN) = ',TP/float(TP+FN),'\n',

'Specificity or True Negative Rate = TN/(TN+FP) = ',TN/float(TN+FP),'\n',

'Positive Predictive value = TP/(TP+FP) = ',TP/float(TP+FP),'\n',

'Negative predictive Value = TN/(TN+FN) = ',TN/float(TN+FN),'\n',

'Positive Likelihood Ratio = Sensitivity/(1-Specificity) = ',sensitivity/(1-specificity),'\n',

'Negative likelihood Ratio = (1-Sensitivity)/Specificity = ',(1-sensitivity)/specificity)
```

```
The acuuracy of the model = TP+TN/(TP+TN+FP+FN) =  0.8852459016393442
 The Missclassification = 1-Accuracy =  0.11475409836065575
 Sensitivity or True Positive Rate = TP/(TP+FN) =  0.90625
 Specificity or True Negative Rate = TN/(TN+FP) =  0.8620689655172413
 Positive Predictive value = TP/(TP+FP) =  0.8787878787878788
 Negative predictive Value = TN/(TN+FN) =  0.8928571428571429
 Positive Likelihood Ratio = Sensitivity/(1-Specificity) =  6.570312499999997
 Negative likelihood Ratio = (1-Sensitivity)/Specificity =  0.10875000000000001
```

- We obtained the overall results as shown above.

```
In [32]: from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_yes[:,1])
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive rate (1-Specificity)')
plt.ylabel('True positive rate (Sensitivity)')
plt.grid(True)
```

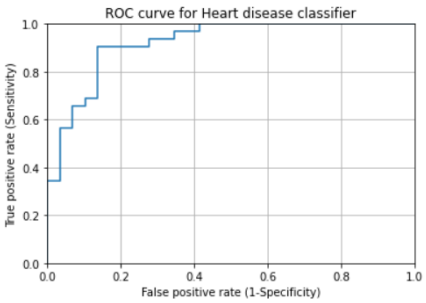

Figure 3.5

- We also check the roc_curve and from the obtained graph we find that we have greater area. And hence we can use our model to predict the future values.

```
In [33]: from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,y_pred_prob_yes[:,1])
```

```
Out[33]: 0.9202586206896552
```

- Also, the roc_auc_score is quite significant and is 92.05%.

COMPLETE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
data = pd.read_csv("heart.csv")
data.head()
data.describe()
data["target"].unique()
sns.countplot(x='target', data=data)
sns.countplot(x='target',hue = 'sex',  data=data)
sns.pairplot(data)
data.info()
X = data.iloc[:,:-1]
Y = data.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
print (clf.intercept_, clf.coef_)
pred = clf.predict(X_test)
print ('Accuracy from sk-learn: {0}'.format(clf.score(X_test, y_test)))
from sklearn.metrics import recall_score, auc
recall_score(pred, y_test)
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
from statsmodels.tools import add_constant as add_constant

heart_df_constant = add_constant(data)

heart_df_constant.head()

import scipy.stats as st

import statsmodels.api as sm

st.chisqprob = lambda chisq, df: st.chi2.sf(chisq, df)

cols=heart_df_constant.columns[:-1]

model=sm.Logit(data.target,heart_df_constant[cols])

result=model.fit()

result.summary()

def back_feature_elem (data_frame,dep_var,col_list):

    """ Takes in the dataframe, the dependent variable and a list of column names, runs the
regression repeatedly eleminating feature with the highest

    P-value above alpha one at a time and returns the regression summary with all p-values below
alpha"""

 while len(col_list)>0 :

        model=sm.Logit(dep_var,data_frame[col_list])

        result=model.fit(disp=0)

        largest_pvalue=round(result.pvalues,3).nlargest(1)

        if largest_pvalue[0]<(0.05):

            return result

            break

        else:

            col_list=col_list.drop(largest_pvalue.index)

result=back_feature_elem(heart_df_constant,data.target,cols)

result.summary()

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

from sklearn.linear_model import LogisticRegression

logreg=LogisticRegression()

logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)

accuracy_score(y_test, y_pred)
```

```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:
1'])

plt.figure(figsize = (8,5))

sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

TN=cm[0,0]

TP=cm[1,1]

FN=cm[1,0]

FP=cm[0,1]

sensitivity=TP/float(TP+FN)

specificity=TN/float(TN+FP)

print('The acuuracy of the model = TP+TN/(TP+TN+FP+FN) = ',(TP+TN)/float(TP+TN+FP+FN),'\n',

'The Missclassification = 1-Accuracy = ',1-((TP+TN)/float(TP+TN+FP+FN)),'\n',

'Sensitivity or True Positive Rate = TP/(TP+FN) = ',TP/float(TP+FN),'\n',

'Specificity or True Negative Rate = TN/(TN+FP) = ',TN/float(TN+FP),'\n',

'Positive Predictive value = TP/(TP+FP) = ',TP/float(TP+FP),'\n',

'Negative predictive Value = TN/(TN+FN) = ',TN/float(TN+FN),'\n',

'Positive Likelihood Ratio = Sensitivity/(1-Specificity) = ',sensitivity/(1-specificity),'\n',

'Negative likelihood Ratio = (1-Sensitivity)/Specificity = ',(1-sensitivity)/specificity)

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_yes[:,1])

plt.plot(fpr,tpr)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.0])

plt.title('ROC curve for Heart disease classifier')

plt.xlabel('False positive rate (1-Specificity)')

plt.ylabel('True positive rate (Sensitivity)')

plt.grid(True)

from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,y_pred_prob_yes[:,1])
```

CONCLUSION

It is quite significant from our obtained results that our model is good for predicting the future values. And, can help to predict the likelihood of patients getting a heart disease.