Mpercept Academy

Final Project

Data Science with Python

Report on “Heart Disease Analysis”

Submitted By:

Manish KC

Submitted To:

Mpercept Academy

Date: 30th May 2019

Table of Contents

Abstract

A popular saying goes that we are living in an “information age”. Terabytes of data are produced every day. Data analysis is a process of inspecting, cleansing, transforming, and modeling **data** with the goal of discovering useful information, informing conclusions, and supporting decision-making.

The health care industry generates a huge amount of data daily. However, most of it is not effectively used. Efficient tools to extract knowledge from these databases for clinical detection of diseases or other purposes are not much prevalent. The aim of this paper is to analyze various data on predicting heart diseases using various techniques, analyses the various combinations of mining algorithms used and conclude which technique(s) are effective and efficient

Introduction

The heart is one of the main organs of the human body. It pumps blood trough the blood vessels of the circulatory system. The circulatory system is extremely important because it transports blood, oxygen and other materials to the different organs of the body. Heart plays the most crucial role in circulatory system. If the heart does not function properly then it will lead to serious health conditions including death,

Heart disease describes a range of conditions that affect your heart. Diseases under the heart disease umbrella include blood vessel diseases, such as coronary artery disease; heart rhythm problems (arrhythmias); and heart defects you're born with (congenital heart defects), among others. 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.

Here we have a data which classified if patients have heart disease or not according to features in it. We will try to use this data to create a model which tries predict if a patient is suffering from a heart disease.

Tools Used For Data Analysis

1. Numpy

Numpy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array. Using Numpy we can perform mathematical and logical operation on arrays, we can implement shape manipulation, and we can also execute operation related to linear algebra. Numpy has in-built function for linear algebra and random number generation.

```
import numpy as np
```

2. Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. We can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

```
import pandas as pd
```

3. Matplotlib

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.

```
import matplotlib.pyplot as plt
```

4. Seaborn

Seaborn is a visualization library in Python. It is built on top of Matplotlib. Seaborn is built on top of Python's core visualization library Matplotlib. It is meant to serve as a complement, and not a replacement. However, Seaborn comes with some very important features such as

- Built in themes for styling Matplotlib graphics
- Visualizing univariate and bivariate data
- Fitting in and visualizing linear regression models
- Plotting statistical time series data

```
import seaborn as sns
```

5. Scikit-learn

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like Numpy, pandas, and Matplotlib.

The functionality that Scikit-learn provides include:

- Regression, including Linear and Logistic Regression
- Classification, including K-Nearest Neighbors
- Clustering, including K-Means and K-Means++
- Model selection
- Preprocessing, including Min-Max Normalization

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

Project

Reading the Data

We have a data which classified if patients have heart disease or not according to features in it. We will try to use this data to create a model which tries predict if a patient has this disease or not.

First the dataset was downloaded from kaggle, then the dataset was saved in the working directory as heart.csv. With the help of pandas library we used read_csv() to read the dataset and save it to the heart_data variable.

```
heart_data  = pd.read_csv('heart.csv')
```

```
heart_data.head()
```

	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

Data contains;

- age - age in years
- sex - (1 = male; 0 = female)
- cp - chest pain type
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg - resting electrocardiographic results
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest
- slope - the slope of the peak exercise ST segment
- ca - number of major vessels (0-3) colored by flourosopy
- thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
- target - have disease or not (1=yes, 0=no)

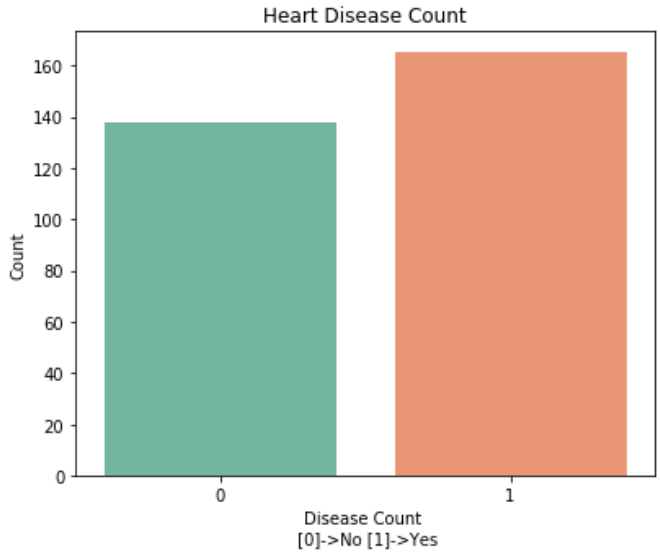
Exploration of data

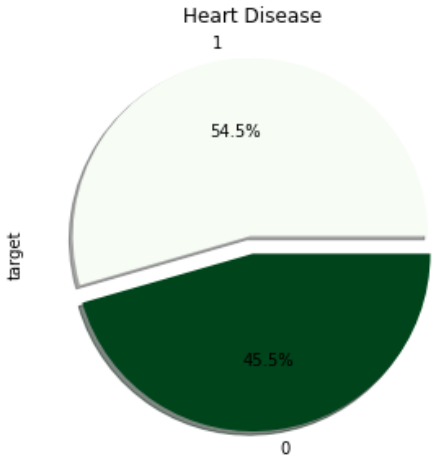
Our data is ready for slicing and dicing, we need to explore our data by using various plots using matplotlib.pyplot as plt library,seaborn as sns library. By doing this we can visually analyses each and every variables in our data.

1. Target

First of all we will look at our target variable in our data set

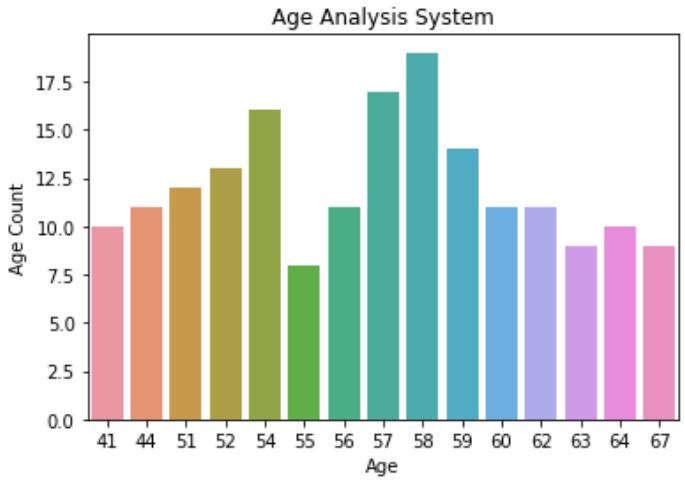
(Target - have disease or not (1=yes, 0=no))





The graph above shows that 165 patient are suffering from heart disease whereas 138 patient are not suffering from heart disease.

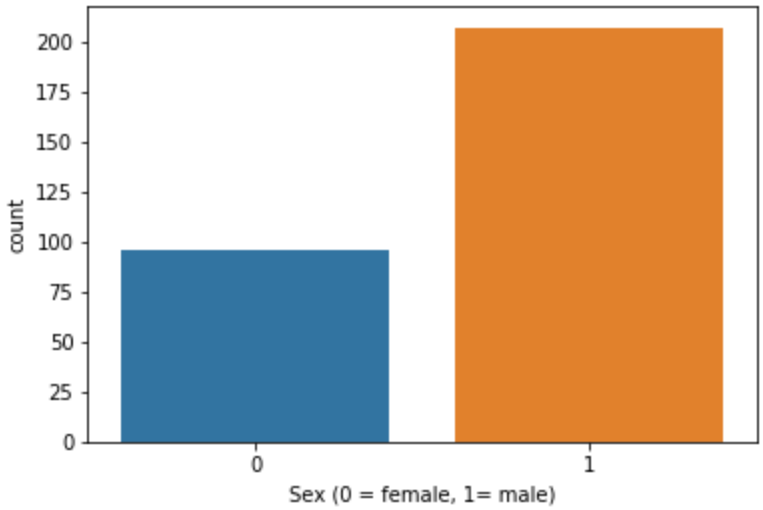
2. Age



Form the above graph we can view the number of patient according to the age group

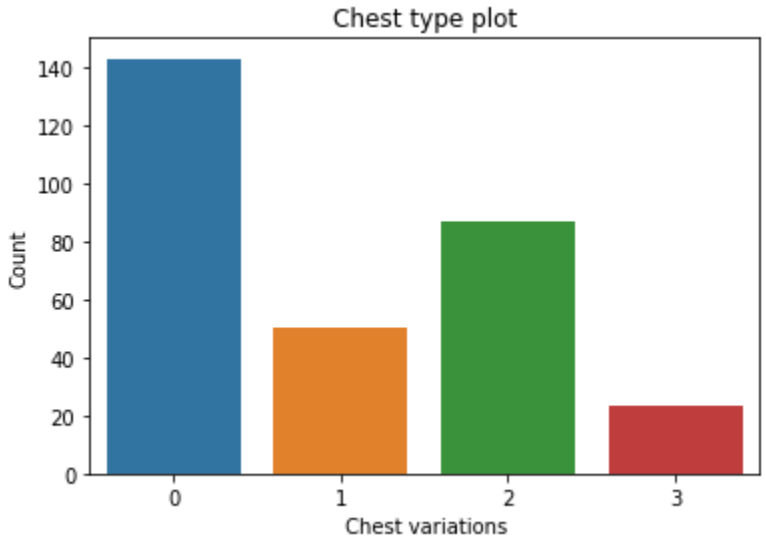
3. Sex

```
: sns.countplot(x ='sex',data=heart_data)
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



Form the above graph we can see in the output, we have 96 females and 207 males

4. Chest Type Pain



From the above graph we can view the categories of chest type pain

0 = Typical Angina

1= Atypical Angina

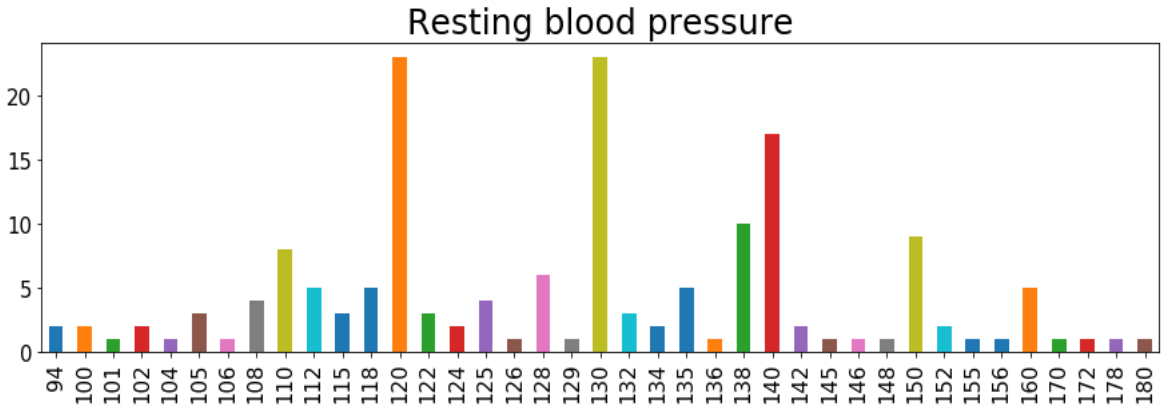
2 = Non-Angina Pain

3 = Asymptomatic

5. Resting Blood Pressure

```
plot = heart_data[heart_data['target'] ==1].trestbps.value_counts().sort_index().plot(kind='bar',figsize = (15,4),fontsize = 15)
plot.set_title("Resting blood pressure",fontsize = 25)
```

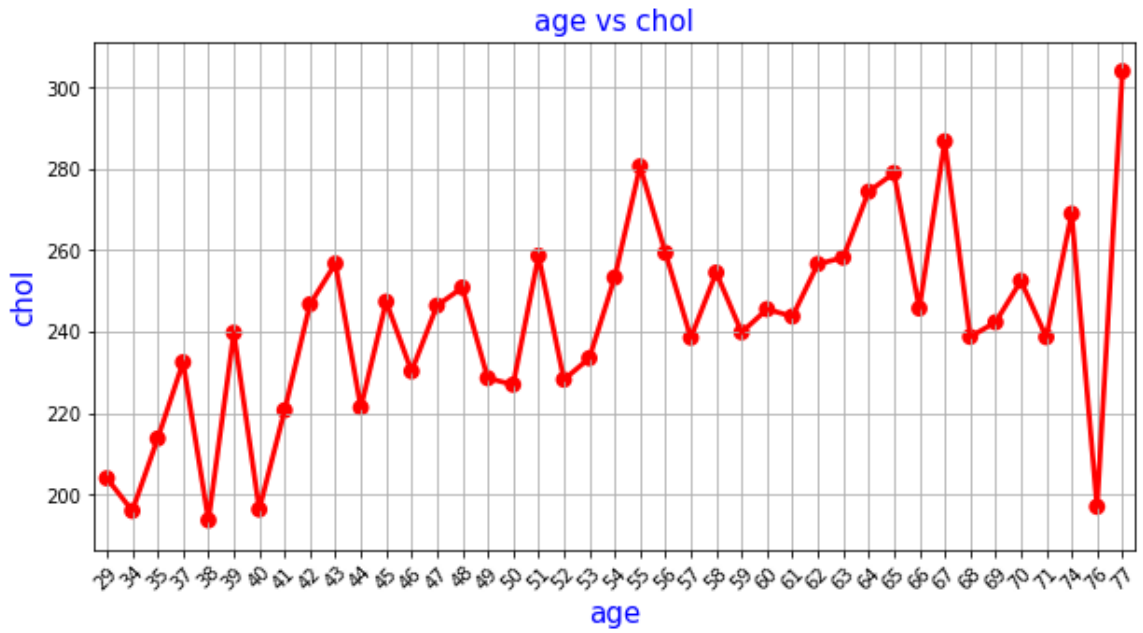
```
Text(0.5, 1.0, 'Resting blood pressure')
```



Form the above graph we can view that (resting blood pressure) variable has the maximum 130mm Hg on admission to the hospital for the person who have heart disease.

6. Cholesterol

```
plt.figure(figsize=(10,5))
sns.pointplot(x=age_unique,y=mean_chol,color='red',alpha=0.8)
plt.xlabel('age',fontsize = 15,color='blue')
plt.xticks(rotation=45)
plt.ylabel('chol',fontsize = 15,color='blue')
plt.title('age vs chol',fontsize = 15,color='blue')
plt.grid()
plt.show()
```



Form the above graph we can view the average cholesterol level according to the age group patient present in the hospital.

Making the Dataset Ready

1. Creating Dummy Variables

```
heart_data.head()
```

	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

In the above dataset we can view that the column cp , thal and slope contain categorical values.

Categorical data are variables that contain label values rather than numeric values. A categorical variable has too many levels. This pulls down performance level of the model. Hence we need to convert the categorical values for this process we have used **dummy coding**.

Dummy coding is a commonly used method for converting a categorical input variable into continuous variable. ‘Dummy’, as the name suggests is a duplicate variable which represents one level of a categorical variable. Presence of a level is represent by 1 and absence is represented by 0. For every level present, one dummy variable will be created.

```
data_heart.drop(['cp','thal','slope'], axis = 1,inplace = True)
data_heart.head()
```

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	target	cp_1	cp_2	cp_3	thal_1	thal_2	thal_3	slope_1	slope_2
0	63	1	145	233	1	0	150	0	2.3	0	1	0	0	1	1	0	0	0	0
1	37	1	130	250	0	1	187	0	3.5	0	1	0	1	0	0	1	0	0	0
2	41	0	130	204	0	0	172	0	1.4	0	1	1	0	0	0	1	0	0	1
3	56	1	120	236	0	1	178	0	0.8	0	1	1	0	0	0	1	0	0	1
4	57	0	120	354	0	1	163	1	0.6	0	1	0	0	0	0	1	0	0	1

As we can see in the above figure the dummy coding method has created dummy variables for cp, thal and slope

2. Separating the values

After creating the set we drop the categorical variables cp thal and slope in the dataset

We also separate the target variable for the dataset.

```
x = data_heart.drop(['target'],axis = 1)
y = data_heart['target']
x.head()
```

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	cp_1	cp_2	cp_3	thal_1	thal_2	thal_3	slope_1	slope_2
0	63	1	145	233	1	0	150	0	2.3	0	0	0	1	1	0	0	0	0
1	37	1	130	250	0	1	187	0	3.5	0	0	1	0	0	1	0	0	0
2	41	0	130	204	0	0	172	0	1.4	0	1	0	0	0	1	0	0	1
3	56	1	120	236	0	1	178	0	0.8	0	1	0	0	0	1	0	0	1
4	57	0	120	354	0	1	163	1	0.6	0	0	0	0	0	1	0	0	1

From the above figure we can see that the target variable was dropped and the dataset was assigned to x

The target variable was assigned to y

3. Normalizing the data

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

```
x = (x - x.min())/(x.max()-x.min())
x.head()
```

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	cp_1	cp_2	cp_3	thal_1	thal_2	thal_3	slope_1	slope_2
0	0.708333	1.0	0.481132	0.244292	1.0	0.0	0.603053	0.0	0.370968	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
1	0.166667	1.0	0.339623	0.283105	0.0	0.5	0.885496	0.0	0.564516	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
2	0.250000	0.0	0.339623	0.178082	0.0	0.0	0.770992	0.0	0.225806	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
3	0.562500	1.0	0.245283	0.251142	0.0	0.5	0.816794	0.0	0.129032	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
4	0.583333	0.0	0.245283	0.520548	0.0	0.5	0.702290	1.0	0.096774	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0

In the above dataset normalization is implemented using the formula of normalization.

$$\tilde{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}.$$

4. Splitting the data

After normalization we will split our data into training and test data

```
from sklearn.model_selection import train_test_split
X_train, X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2,random_state = 2)
```

Creating Model

Logistic Regression

In our dataset our dependent variable is a categorical variable so we will go for a Logistic Regression model. Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

```
from sklearn.linear_model import LogisticRegression
logReg = LogisticRegression()
logReg.fit(X_train,Y_train)
```

```
C:\Users\user\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

```
logReg.predict(X_test)
```

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

```
Train_Logistic_Score = logReg.score(X_train,Y_train)*100
Test_Logistic_Score = logReg.score(X_test,Y_test)*100

print("The accuracy score on the train set = ",Train_Logistic_Score)
print("The accuracy score on the test set = ",Test_Logistic_Score)
```

```
The accuracy score on the train set =  84.71074380165288
The accuracy score on the test set =  90.1639344262295
```

In the above we have implemented the logistic regression on the dataset where we have fitted the train model and then predicted passing the test model which result in measuring the accuracy of the predicted model as score.

Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

```
: from sklearn.tree import DecisionTreeClassifier

  dectree = DecisionTreeClassifier()
  dectree.fit(X_train,Y_train)
```

```
: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, presort=False, random_state=None,
              splitter='best')
```

```
: dectree.predict(X_test)
```

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

```
: Train_Decision_Score = dectree.score(X_train,Y_train)*100
  Test_Decision_Score = dectree.score(X_test,Y_test)*100

  print("The accuracy score on the train set = ",Train_Decision_Score)
  print("The accuracy score on the test set = ",Test_Decision_Score )

  The accuracy score on the train set =  100.0
  The accuracy score on the test set =  81.9672131147541
```

In the above we have implemented the Decision Tree on the dataset where we have fitted the train model and then predicted passing the test model which result in measuring the accuracy of the predicted model as score.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

```
: from sklearn.ensemble import RandomForestClassifier
  rforest = RandomForestClassifier()
  rforest.fit(X_train,Y_train)
```

```
  C:\Users\user\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureW
  change from 10 in version 0.20 to 100 in 0.22.
    "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
              max_depth=None, max_features='auto', max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
              oob_score=False, random_state=None, verbose=0,
              warm_start=False)
```

```
: rforest.predict(X_test)
```

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

```
: Train_Random_Score = rforest.score(X_train,Y_train)*100
  Test_Random_Score = rforest.score(X_test,Y_test)*100

  print("The accuracy score on the train set = ",Train_Random_Score)
  print("The accuracy score on the test set = ",Test_Random_Score)
```

```
  The accuracy score on the train set =  98.7603305785124
  The accuracy score on the test set =  86.88524590163934
```

In the above we have implemented the Random Forest on the dataset where we have fitted the train model and then predicted passing the test model which result in measuring the accuracy of the predicted model as score.

SVM(Support Vector Machine)

A **Support Vector Machine** (**SVM**) is a discriminative classifier formally defined by a separating hyper plane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyper plane which categorizes new examples.

```
from sklearn.svm import SVC
svm = SVC(random_state=1)
```

```
svm.fit(X_train,Y_train)
svm.predict(X_test)
```

```
C:\Users\user\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default value of gamma wi
'auto' to 'scale' in version 0.22 to account better for unscaled features. Set gamma explicitly to 'auto' or 's
his warning.
  "avoid this warning.", FutureWarning)
```

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

```
Train_SVM_Score = svm.score(X_train,Y_train)*100
Test_SVM_Score = svm.score(X_test,Y_test)*100

print("The accuracy score on the train set = ",Train_SVM_Score)
print("The accuracy score on the test set = ",Test_SVM_Score)
```

```
The accuracy score on the train set =  83.05785123966942
The accuracy score on the test set =  88.52459016393442
```

In the above we have implemented the SVM on the dataset where we have fitted the train model and then predicted passing the test model which result in measuring the accuracy of the predicted model as score.

KNN (K Nearest Neighbors)

In pattern recognition, the k-nearest neighbor's algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression: In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

```
from  sklearn.preprocessing import StandardScaler
stan_sc = StandardScaler()
x_train = stan_sc.fit_transform(X_train)
x_test = stan_sc.fit_transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5,metric = "minkowski",p = 2)
classifier.fit(x_train,Y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

```
classifier.predict(x_test)
```

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

```
Train_KNN_Score = svm.score(x_train,Y_train)*100
Test_KNN_Score = svm.score(x_test,Y_test)*100

print("The accuracy score on the train set = ",Train_KNN_Score)
print("The accuracy score on the test set = ",Test_KNN_Score)
```

```
The accuracy score on the train set =  83.88429752066115
The accuracy score on the test set =  85.24590163934425
```

In the above we have implemented the KNN on the dataset where we have fitted the train model and then predicted passing the test model which result in measuring the accuracy of the predicted model as score.

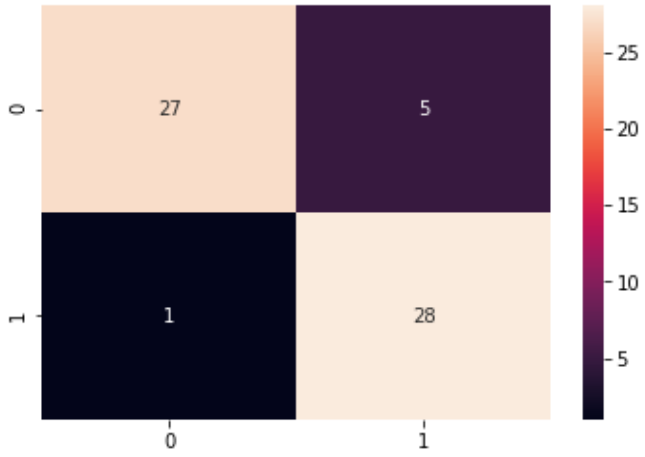
Explanation

For explanation purpose we will use the Confusion Matrix, confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

Logistic Regression

```
: #Creating a confusion maatrix for better explanation
from sklearn.metrics import confusion_matrix
cm_lg = confusion_matrix(Y_test,logReg.predict(X_test))
sns.heatmap(cm_lg,annot = True)
plt.plot()
```

```
: []
```



For the above Confusion Matrix we can state that-:

True positive block contain 27 patient those have heart disease and are correctly identified by the algorithm

True Negative block contain 28 patient those didn't have heart disease and was correctly identified by the algorithm

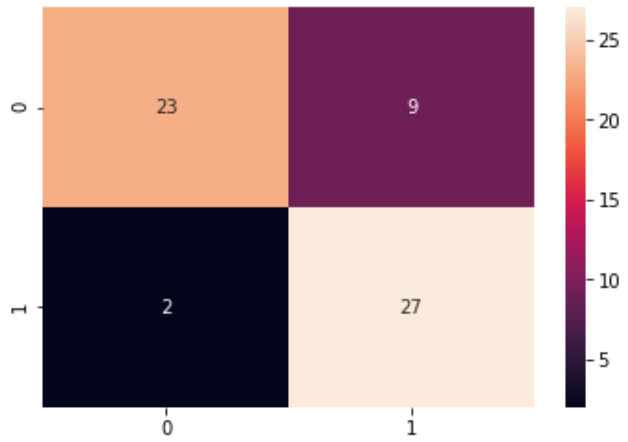
False Negative contain 1 patient who has heart disease but was not recognized by the algorithm

False Positive contain 5 patient those had heart disease but was not recognized by the algorithm

Decision Tree

```
from sklearn.metrics import confusion_matrix
tree = confusion_matrix(Y_test,dectree.predict(X_test))
sns.heatmap(tree,annot = True)
plt.plot
```

```
<function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)>
```



Form the above Confusion Matrix we can state that-:

True positive block contain 23 patient those have heart disease and are correctly identified by the algorithm

True Negative block contain 27 patient those didn't have heart disease and was correctly identified by the algorithm

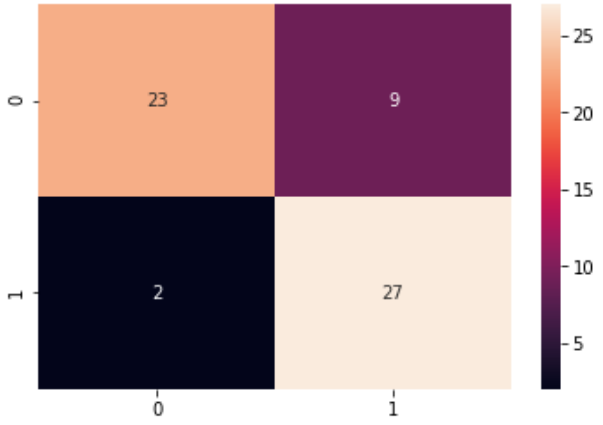
False Negative contain 2 patient who has heart disease but was not recognized by the algorithm

False Positive contain 9 patient those had heart disease but was not recognized by the algorithm

Random Forest

```
: from sklearn.metrics import confusion_matrix
  random_tree = confusion_matrix(Y_test,rforest.predict(X_test))
  sns.heatmap(tree,annot = True)
  plt.plot
```

```
: <function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)>
```



Form the above Confusion Matrix we can state that-:

True positive block contain 23 patient those have heart disease and are correctly identified by the algorithm

True Negative block contain 27 patient those didn't have heart disease and was correctly identified by the algorithm

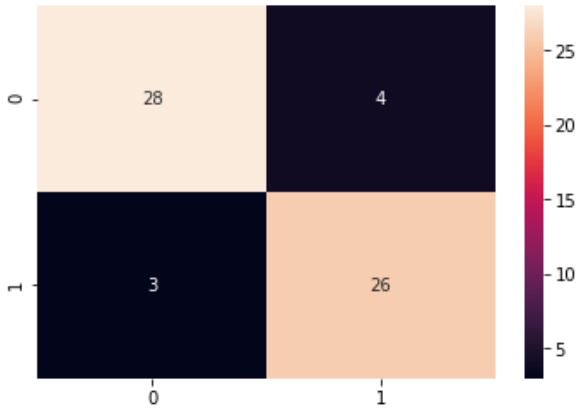
False Negative contain 2 patient who has heart disease but was not recognized by the algorithm

False Positive contain 9 patient those had heart disease but was not recognized by the algorithm

SVM (Support Vector Machine)

```
: from sklearn.metrics import confusion_matrix
  SVM = confusion_matrix(Y_test,svm.predict(X_test))
  sns.heatmap(SVM,annot = True)
  plt.plot
```

```
: <function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)>
```



Form the above Confusion Matrix we can state that-:

True positive block contain 28 patient those have heart disease and are correctly identified by the algorithm

True Negative block contain 26 patient those didn't have heart disease and was correctly identified by the algorithm

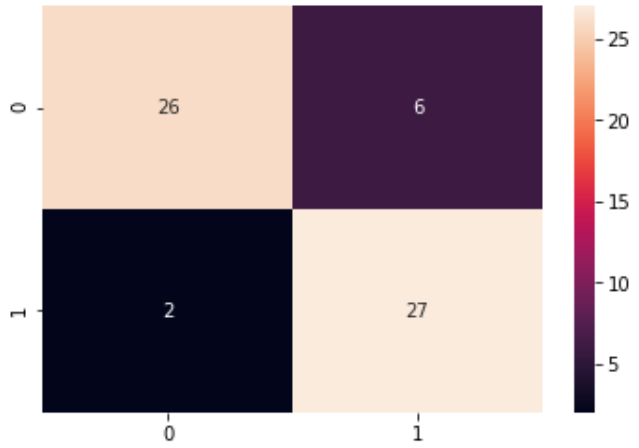
False Negative contain 3 patient who has heart disease but was not recognized by the algorithm

False Positive contain 4 patient those had heart disease but was not recognized by the algorithm

KNN (K Nearest Neighbors)

```
from sklearn.metrics import confusion_matrix
KNN = confusion_matrix(Y_test,classifier.predict(x_test))
sns.heatmap(KNN,annot = True)
plt.plot
```

```
<function matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)>
```



Form the above Confusion Matrix we can state that-:

True positive block contain 26 patient those have heart disease and are correctly identified by the algorithm

True Negative block contain 27 patient those didn't have heart disease and was correctly identified by the algorithm

False Negative contain 2 patient who has heart disease but was not recognized by the algorithm

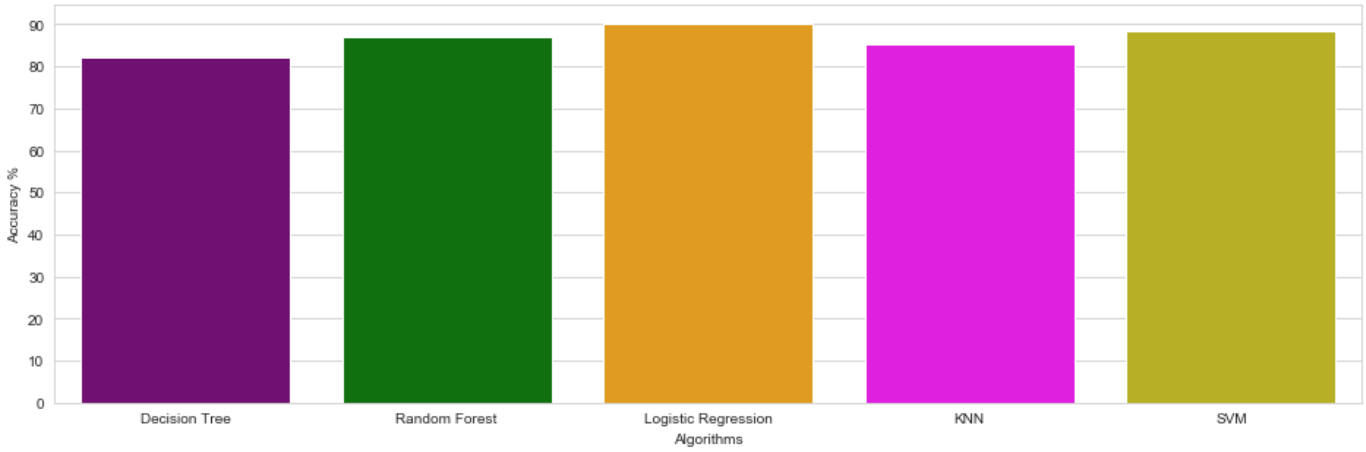
False Positive contain 6 patient those had heart disease but was not recognized by the algorithm

Comparisons

The comparisons were made to find out which model was efficient predicting whether the patient is suffering from a heart disease or not. Examining all the factors which directly relate to heart disease we have concluded that logistic regression is best suitable and has resulted more accuracy then other algorithms.

```
methods = ["Decision Tree", "Random Forest","Logistic Regression", "KNN", "SVM"]
accuracy = [Test_Decision_Score, Test_Random_Score,Test_Logistic_Score, Test_KNN_Score, Test_SVM_Score]
colors = ["purple", "green", "orange", "magenta","#CFC60E","#0FBBAE"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=methods, y=accuracy, palette=colors)
plt.show()
```



Algorithms	Accuracy
Logistic Regression	90.16
Decision Tree	81.96
Random Forest	86.88
Support Vector Machine	88.52
KNN	85.24