HEART DISEASE CLASSIFICATIONS USING MACHINE LEARNING

Project Report
Submitted to

DEVI AHILYA VISHWAVIDHYALYA INDORE

In Partial Fulfillment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

in

BIG DATA ANALYTIC

by

Manish Kumar Singh
DS7B-2206

Under the Supervision of

Dr. Dinesh Bhati
Assistant Professor

SCHOOL OF DATA SCIENCE AND FORECASTING

2022-23

DECLARATION

I hereby declare that the work presented in this project report entitled "HEART DISEASE CLASSIFICATIONS USING MACHINE LEARNING" was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources. I affirm that no portion of my work is plagiarized, and the experiments and results reported in the project report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Manish Kumar Singh

DS7B-2206

(Candidate Signature)

CERTIFICATE

Certified that Manish Kumar Singh (roll number DS7B-2206) has carried out the project work titled "Titanic decision tree algorithm" for the award of Master of Technology in Big Data Analytic DEVI AHILYA VISHWAVIDHYALYA INDORE under my supervision. The work embodies results of work, carried out by the student himself/herself and the contents of the work do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Name of Supervisor: Dr. Dinesh Bhati

Designation: Assistant Professor

Signature

Date

Head of the Department: Dr. V.B Gupta

Signature

Date

ABSTRACT

You have just been hired as a **Data Scientist** at a Hospital with an alarming number of patients coming in reporting various cardiac symptoms. A cardiologist measures vitals & hands you this data to **perform Data Analysis** and **predict** whether certain patients have Heart Disease. We would like to make a **Machine Learning algorithm** where we can train our AI to learn & improve from experience. Thus, we would want to **classify** patients as either positive or negative for Heart Disease.

INDEX

INTRODUCTION

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. We will use logistic regression (classification) algorithm.

In [2]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/dock
# For example, here's several helpful packages to load in

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list t

import os
print(os.listdir(".."))

# Any results you write to the current directory are saved as output.
```

```
['All Users', 'Default', 'Default User', 'desktop.ini', 'PC', 'Public']
```

Read Data

In [40]:
```
# We are reading our data
df = pd.read_csv("heart.csv")
```

In [41]:
```
# First 5 rows of our data
df.head()
```

Out[41]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalach	exng	oldpeak	slope	caa	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)

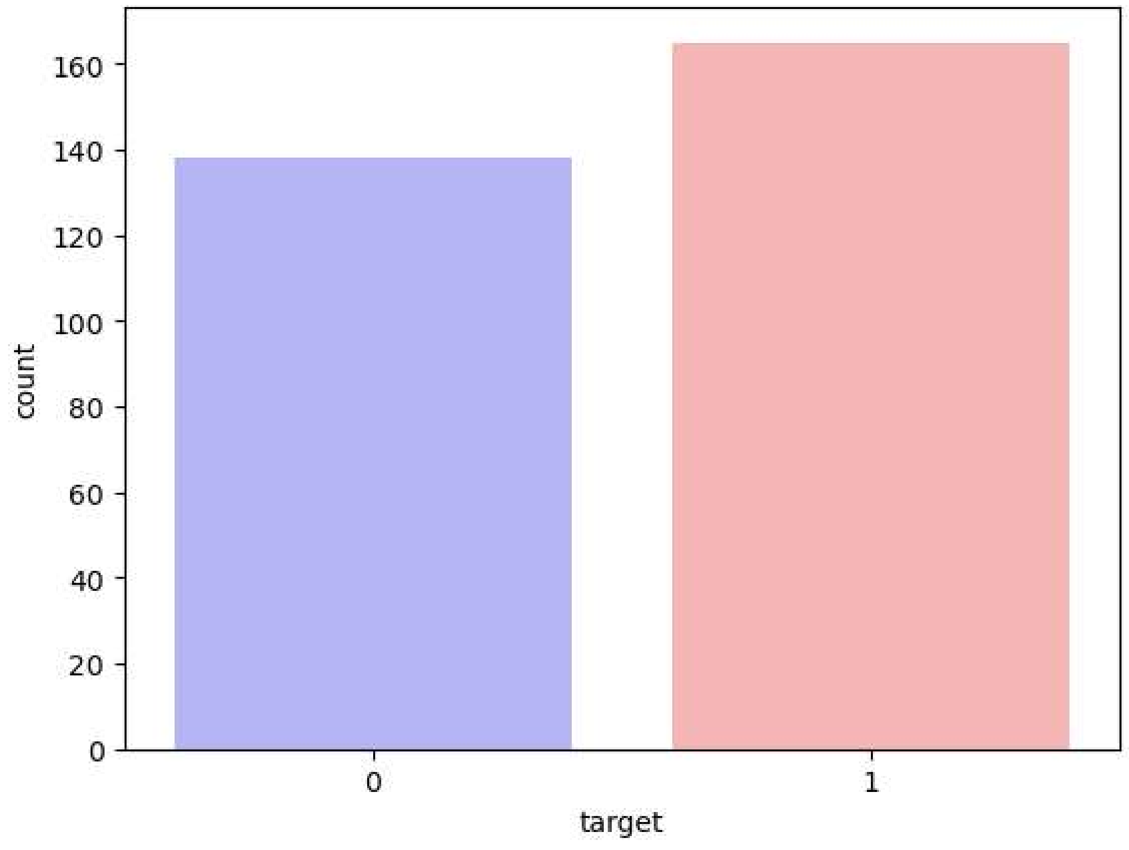
- cp - chest pain type
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg - resting electrocardiographic results
- thalach - maximum heart rate achieved
- exang - exercise 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)

Data Exploration

In [19]:
```
df.target.value_counts()
```

Out[19]:
```
1    165
0    138
Name: target, dtype: int64
```

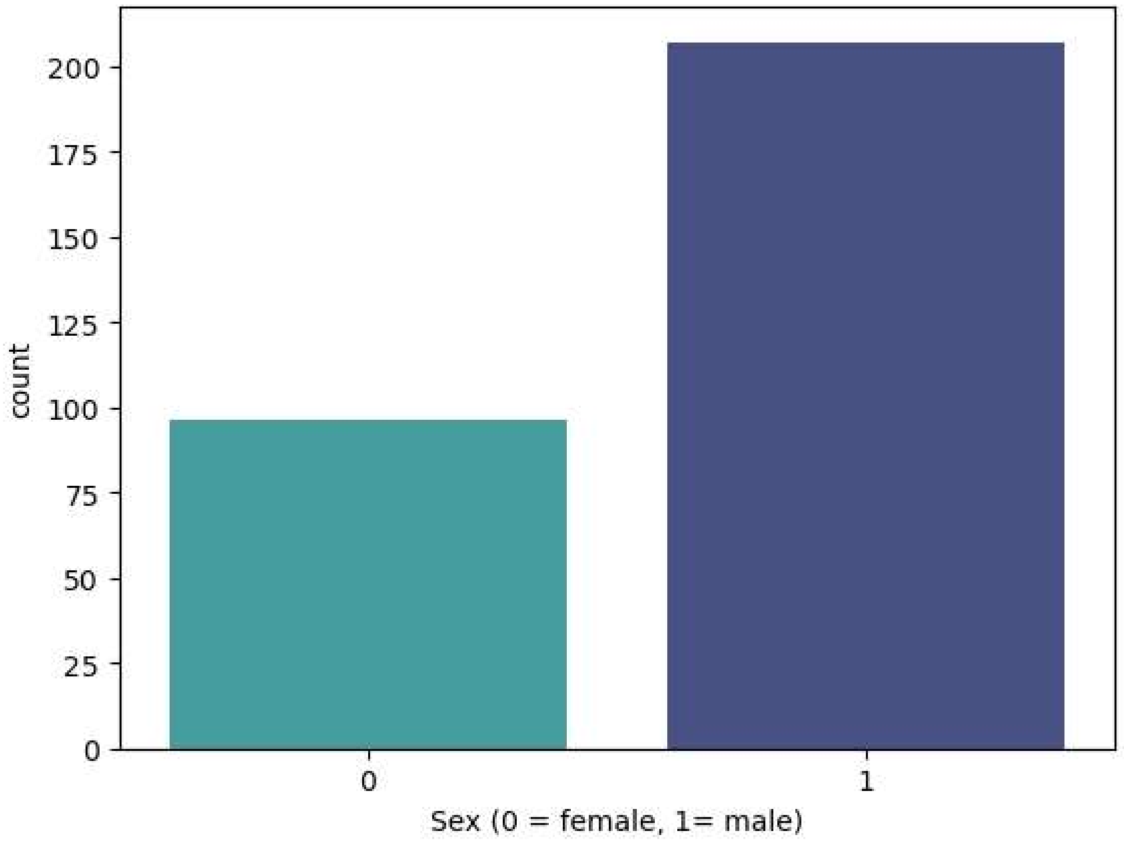
In [20]:
```
sns.countplot(x="target", data=df, palette="bwr")
plt.show()
```



In [21]:
```
countNoDisease = len(df[df.target == 0])
countHaveDisease = len(df[df.target == 1])
print("Percentage of Patients Haven't Heart Disease: {:.2f}%".format((countNoDise
print("Percentage of Patients Have Heart Disease: {:.2f}%".format((countHaveDisea
```

```
Percentage of Patients Haven't Heart Disease: 45.54%
Percentage of Patients Have Heart Disease: 54.46%
```

In [22]:
```
sns.countplot(x='sex', data=df, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



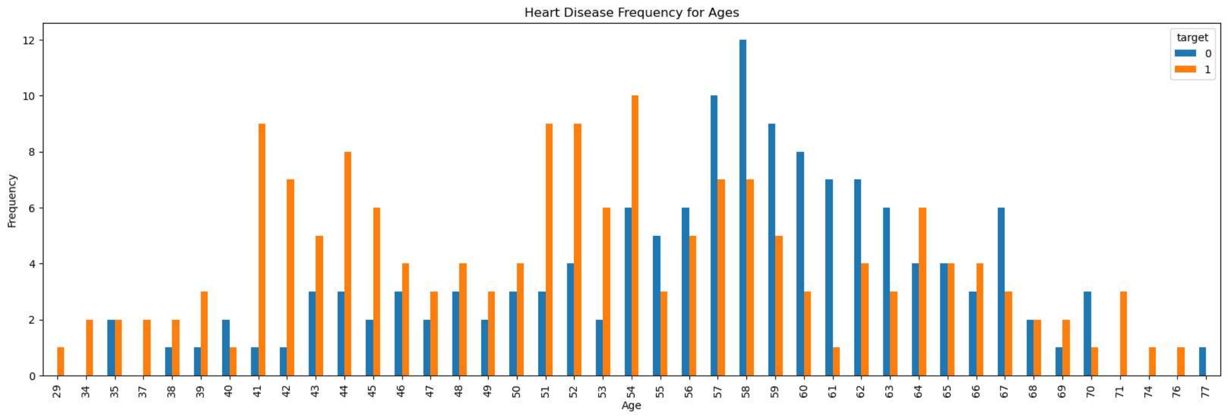
In [23]:
```
countFemale = len(df[df.sex == 0])
countMale = len(df[df.sex == 1])
print("Percentage of Female Patients: {:.2f}%".format((countFemale / (len(df.sex)
print("Percentage of Male Patients: {:.2f}%".format((countMale / (len(df.sex))*10
```

```
Percentage of Female Patients: 31.68%
Percentage of Male Patients: 68.32%
```

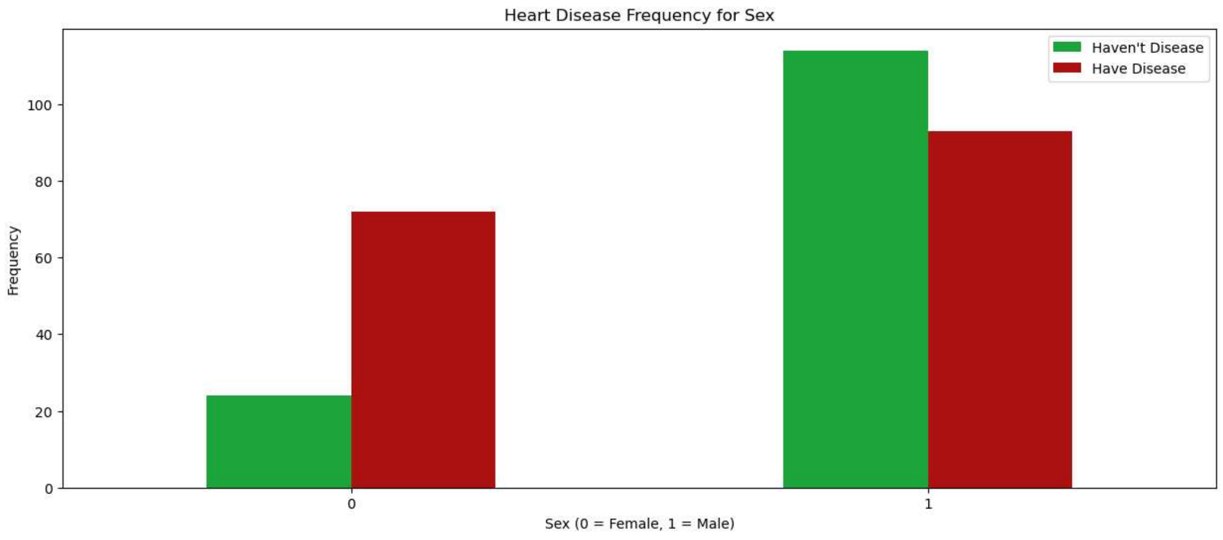
In [24]:
```
df.groupby('target').mean()
```

Out[24]:
```
             age       sex        cp      trtbps        chol       fbs   restecg    thalachh
target
     0 56.601449  0.826087  0.478261  134.398551  251.086957  0.159420  0.449275  139.101449  0.5
     1 52.496970  0.563636  1.375758  129.303030  242.230303  0.139394  0.593939  158.466667  0.1
```

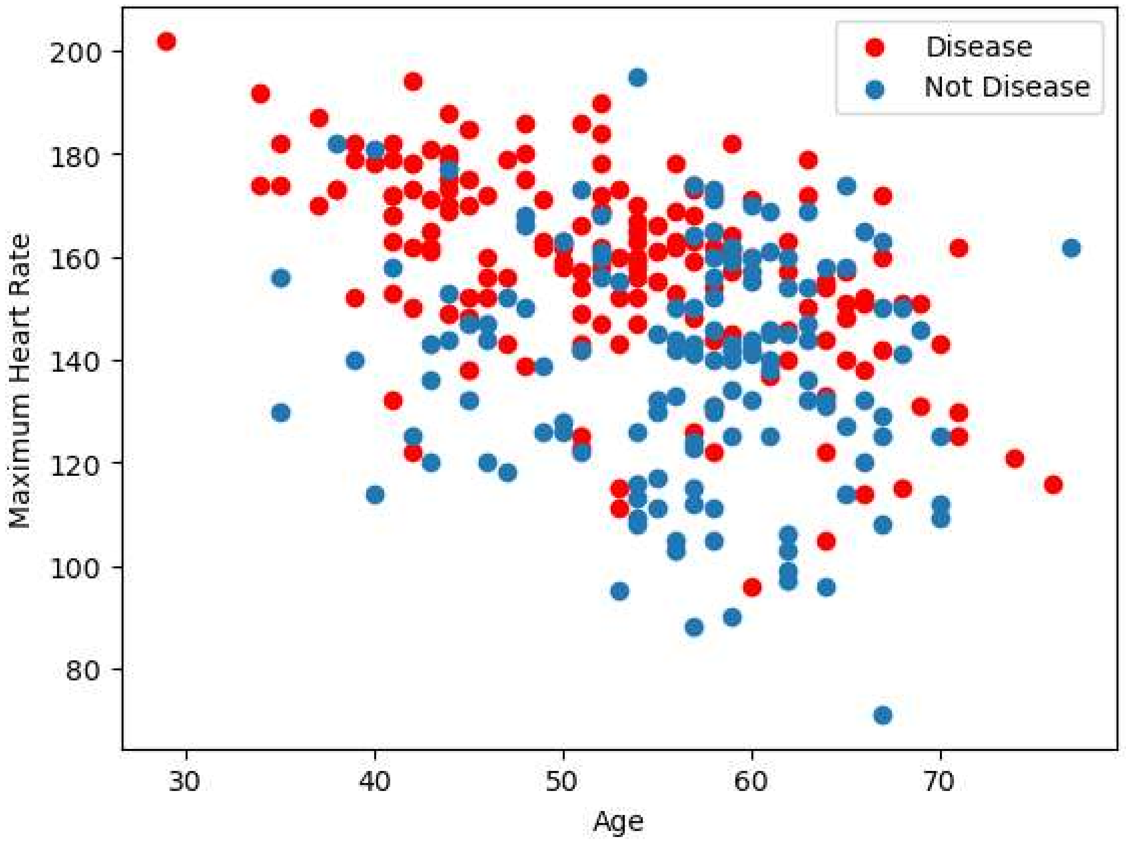
In [25]:
```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



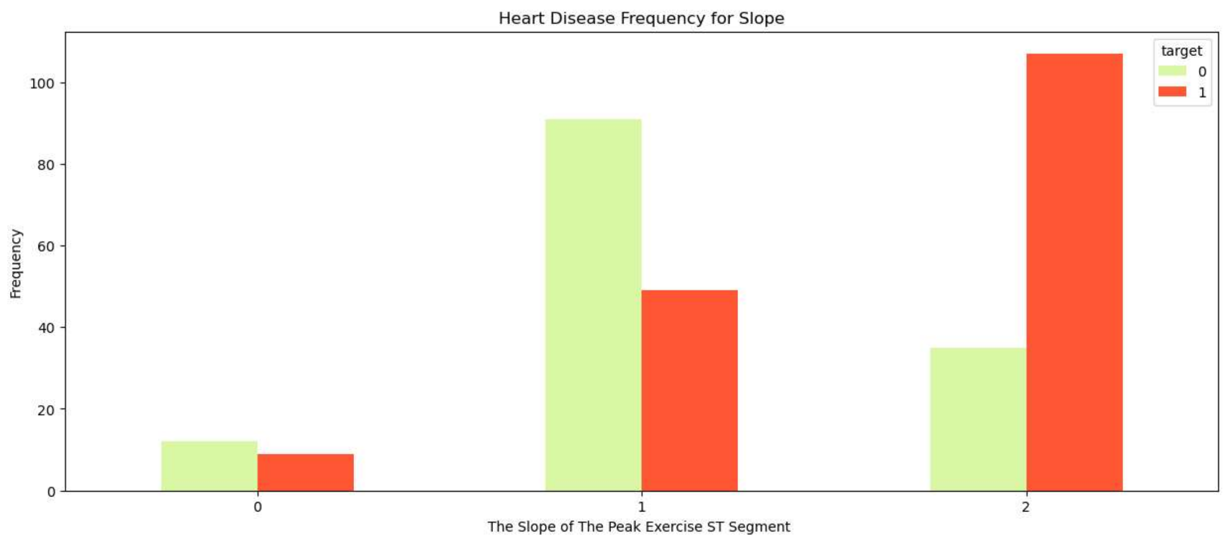
In [26]:
```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#1CA53B','#A
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()
```



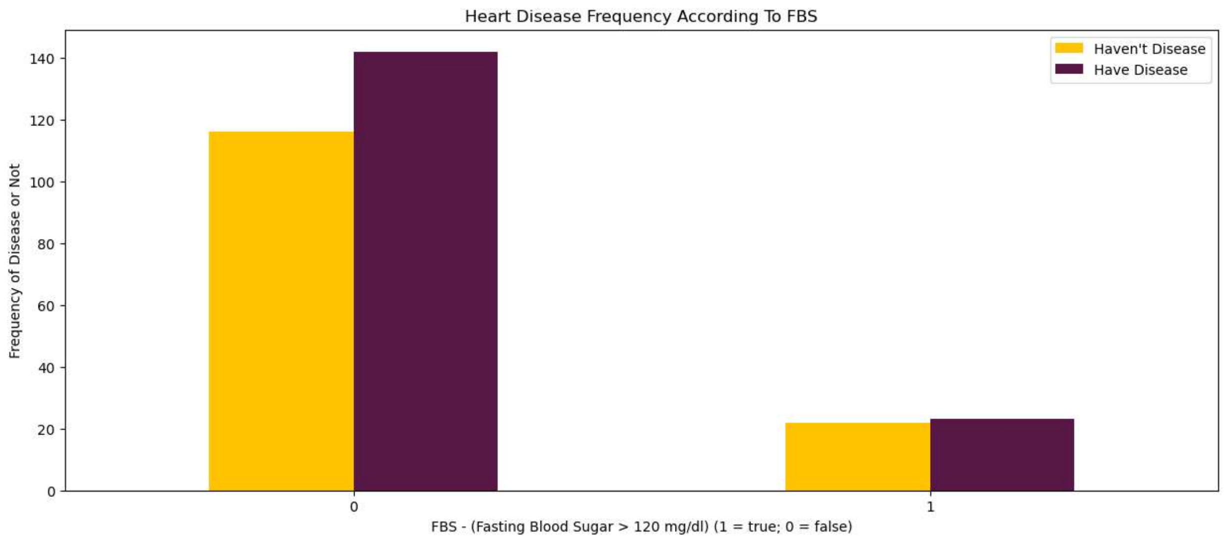
In [31]:
```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



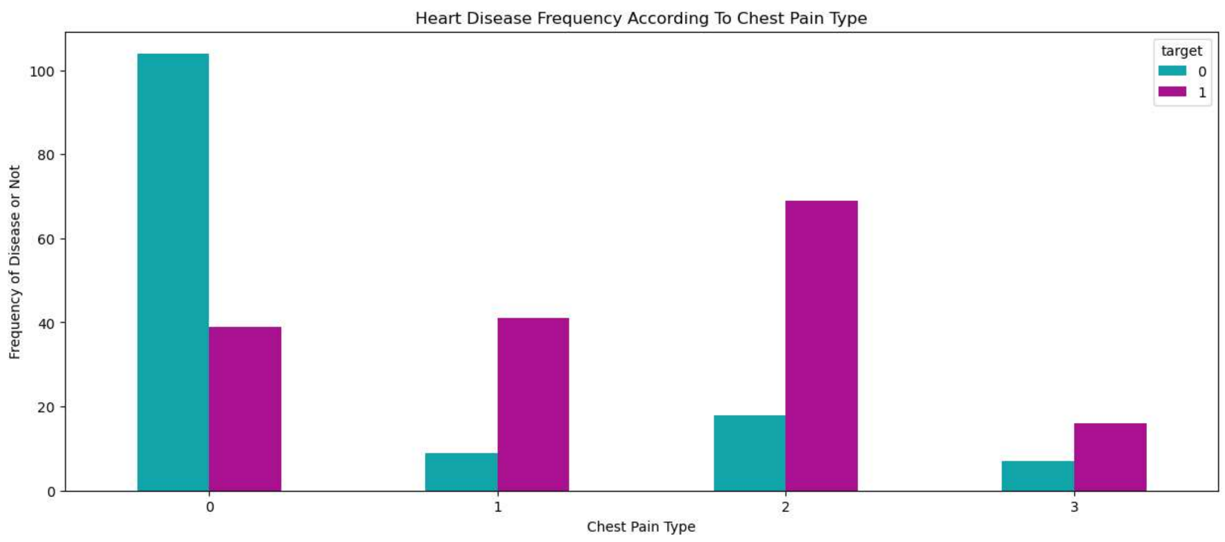
In [36]:
```
pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#DAF7A6','
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



In [37]:
```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#FFC300','#5
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



In [38]:
```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA','#AA
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



Creating Dummy Variables

Since 'cp', 'thal' and 'slope' are categorical variables we'll turn them into dummy variables.

In [42]:
```
a = pd.get_dummies(df['cp'], prefix = "cp")
b = pd.get_dummies(df['thal'], prefix = "thal")
c = pd.get_dummies(df['slope'], prefix = "slope")
```

In [43]:
```python
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()
```

Out[43]:
```
   age  sex  cp  trtbps  chol  fbs  restecg  thalach  exng  oldpeak  ...  cp_1  cp_2  cp_3  thal
0   63    1   3     145   233    1        0      150     0      2.3  ...     0     0     1
1   37    1   2     130   250    0        1      187     0      3.5  ...     0     1     0
2   41    0   1     130   204    0        0      172     0      1.4  ...     1     0     0
3   56    1   1     120   236    0        1      178     0      0.8  ...     1     0     0
4   57    0   0     120   354    0        1      163     1      0.6  ...     0     0     0

5 rows × 25 columns
```

In [44]:
```python
df = df.drop(columns = ['cp', 'thal', 'slope'])
df.head()
```

Out[44]:
```
   age  sex  trtbps  chol  fbs  restecg  thalach  exng  oldpeak  caa  ...  cp_1  cp_2  cp_3  thal_(
0   63    1     145   233    1        0      150     0      2.3    0  ...     0     0     1      (
1   37    1     130   250    0        1      187     0      3.5    0  ...     0     1     0      (
2   41    0     130   204    0        0      172     0      1.4    0  ...     1     0     0      (
3   56    1     120   236    0        1      178     0      0.8    0  ...     1     0     0      (
4   57    0     120   354    0        1      163     1      0.6    0  ...     0     0     0      (

5 rows × 22 columns
```

Creating Model for Logistic Regression

We can use sklearn library or we can write functions ourselves. Let's them both. Firstly we will write our functions after that we'll use sklearn library to calculate score.

In [45]:
```python
y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

Normalize Data

In [46]:
```
# Normalize
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

```
C:\Users\PC\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWar
ning: In a future version, DataFrame.min(axis=None) will return a scalar min ov
er the entire DataFrame. To retain the old behavior, use 'frame.min(axis=0)' or
just 'frame.min()'
  return reduction(axis=axis, out=out, **passkwargs)
C:\Users\PC\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWar
ning: In a future version, DataFrame.max(axis=None) will return a scalar max ov
er the entire DataFrame. To retain the old behavior, use 'frame.max(axis=0)' or
just 'frame.max()'
  return reduction(axis=axis, out=out, **passkwargs)
C:\Users\PC\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWar
ning: In a future version, DataFrame.min(axis=None) will return a scalar min ov
er the entire DataFrame. To retain the old behavior, use 'frame.min(axis=0)' or
just 'frame.min()'
  return reduction(axis=axis, out=out, **passkwargs)
```

We will split our data. 80% of our data will be train data and 20% of it will be test data.

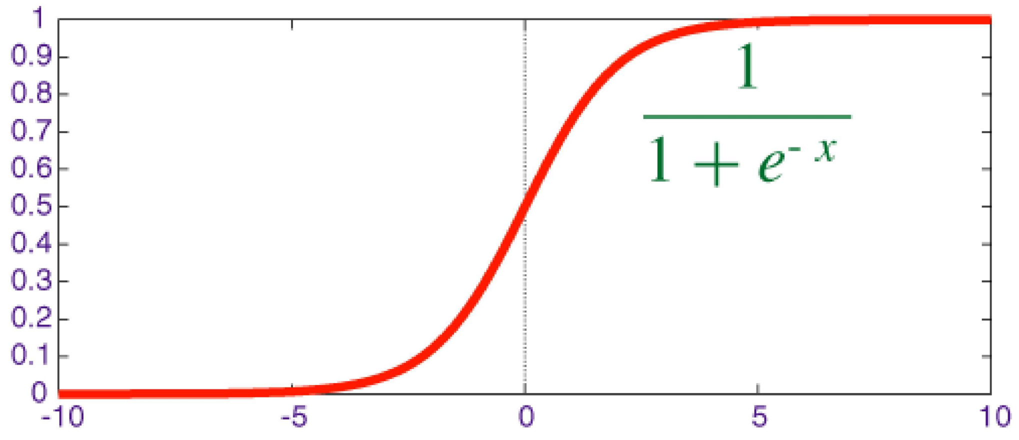
In [47]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_st
```

In [48]:
```
#transpose matrices
x_train = x_train.T
y_train = y_train.T
x_test = x_test.T
y_test = y_test.T
```

Let's say weight = 0.01 and bias = 0.0

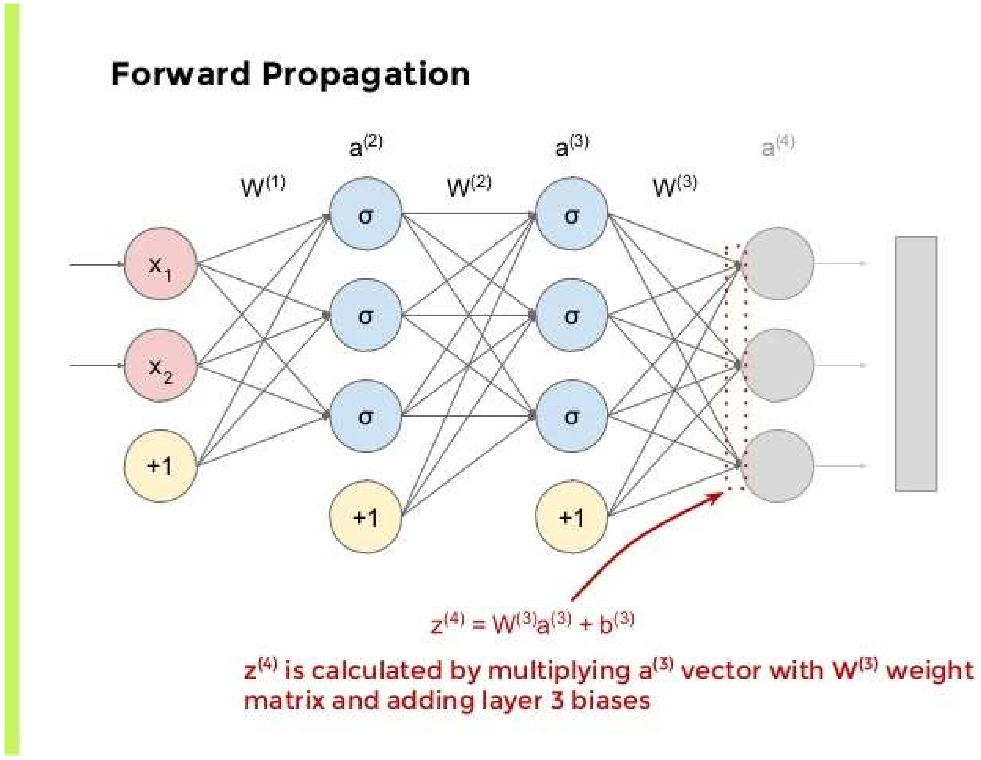
In [49]:
```
#initialize
def initialize(dimension):

    weight = np.full((dimension,1),0.01)
    bias = 0.0
    return weight,bias
```

Sigmoid Function



In [50]:
```
def sigmoid(z):

    y_head = 1/(1+ np.exp(-z))
    return y_head
```

Forward and Backward Propagation

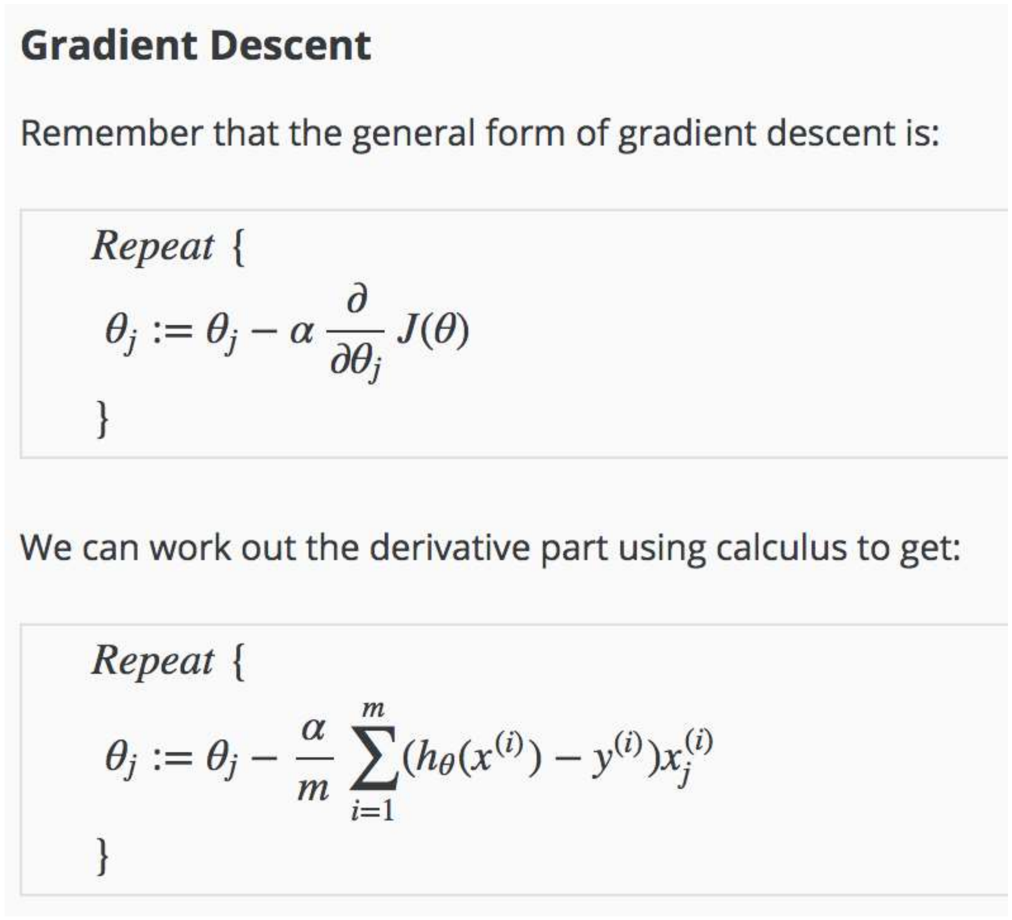


Forward Propagation
$a^{(2)}$
$a^{(3)}$
$a^{(4)}$
$W^{(1)}$
$W^{(2)}$
$W^{(3)}$
σ
x_1
x_2
+1
$z^{(4)} = W^{(3)}a^{(3)} + b^{(3)}$
$z^{(4)}$ is calculated by multiplying $a^{(3)}$ vector with $W^{(3)}$ weight matrix and adding layer 3 biases

Cost Function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Gradient Descent



Gradient Descent

Remember that the general form of gradient descent is:

$$\begin{aligned}&Repeat\ \{\\&\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\\&\}\end{aligned}$$

We can work out the derivative part using calculus to get:

$$\begin{aligned}&Repeat\ \{\\&\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}\\&\}\end{aligned}$$

By the way in formulas;

- h0(x^i)= y_head
- y^i = y_train
- x^i = x_train

```
In [51]: def forwardBackward(weight,bias,x_train,y_train):
             # Forward

             y_head = sigmoid(np.dot(weight.T,x_train) + bias)
             loss = -(y_train*np.log(y_head) + (1-y_train)*np.log(1-y_head))
             cost = np.sum(loss) / x_train.shape[1]

             # Backward
             derivative_weight = np.dot(x_train,((y_head-y_train).T))/x_train.shape[1]
             derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]
             gradients = {"Derivative Weight" : derivative_weight, "Derivative Bias" : der

             return cost,gradients
```

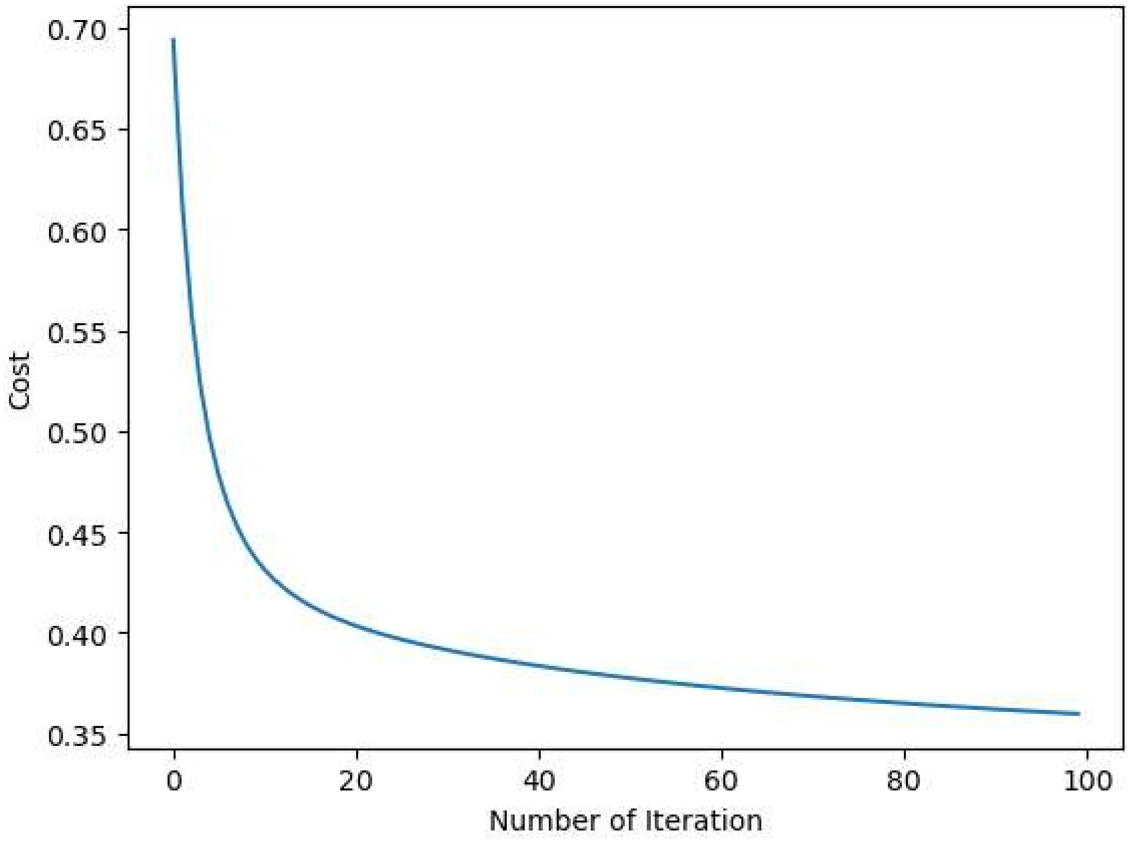
In [52]:
```
def update(weight,bias,x_train,y_train,learningRate,iteration) :
    costList = []
    index = []

    #for each iteration, update weight and bias values
    for i in range(iteration):
        cost,gradients = forwardBackward(weight,bias,x_train,y_train)
        weight = weight - learningRate * gradients["Derivative Weight"]
        bias = bias - learningRate * gradients["Derivative Bias"]

        costList.append(cost)
        index.append(i)

    parameters = {"weight": weight,"bias": bias}

    print("iteration:",iteration)
    print("cost:",cost)

    plt.plot(index,costList)
    plt.xlabel("Number of Iteration")
    plt.ylabel("Cost")
    plt.show()

    return parameters, gradients
```

In [53]:
```
def predict(weight,bias,x_test):
    z = np.dot(weight.T,x_test) + bias
    y_head = sigmoid(z)

    y_prediction = np.zeros((1,x_test.shape[1]))

    for i in range(y_head.shape[1]):
        if y_head[0,i] <= 0.5:
            y_prediction[0,i] = 0
        else:
            y_prediction[0,i] = 1
    return y_prediction
```

In [56]:
```
def logistic_regression(x_train,y_train,x_test,y_test,learningRate,iteration):
    dimension = x_train.shape[0]
    weight,bias = initialize(dimension)

    parameters, gradients = update(weight,bias,x_train,y_train,learningRate,itera

    y_prediction = predict(parameters["weight"],parameters["bias"],x_test)

    print("Manuel Test Accuracy: {:.2f}%".format((100 - np.mean(np.abs(y_predicti
```

In [55]:
```
logistic_regression(x_train,y_train,x_test,y_test,1,100)
```

```
iteration: 100
cost: 0.3597736123664534
```



```
Manuel Test Accuracy: 86.89%
```

Manuel Test Accuracy is 86.89%

Let's find out sklearn's score.

Sklearn Logistic Regression

In [57]:
```
accuracies = {}

lr = LogisticRegression()
lr.fit(x_train.T,y_train.T)
acc = lr.score(x_test.T,y_test.T)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))
```

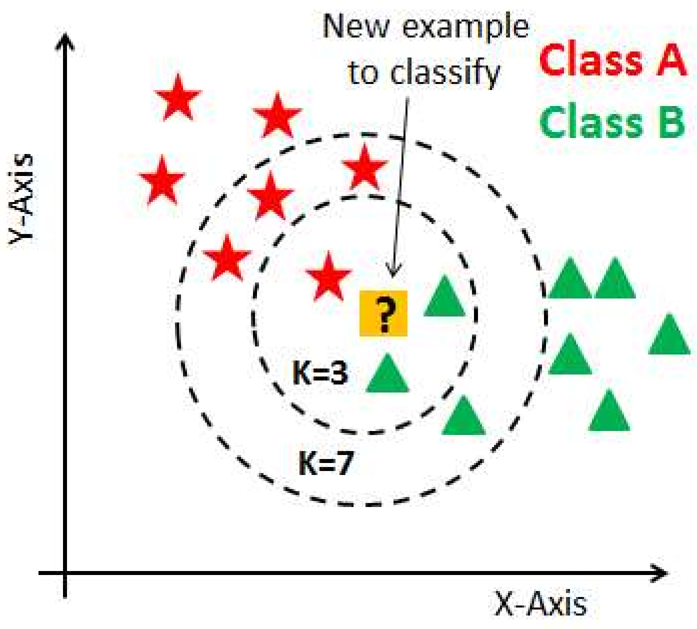
```
Test Accuracy 86.89%
```

1. **Our model works with 86.89% accuracy.**

K-Nearest Neighbour (KNN) Classification

Let's see what will be score if we use KNN algorithm.

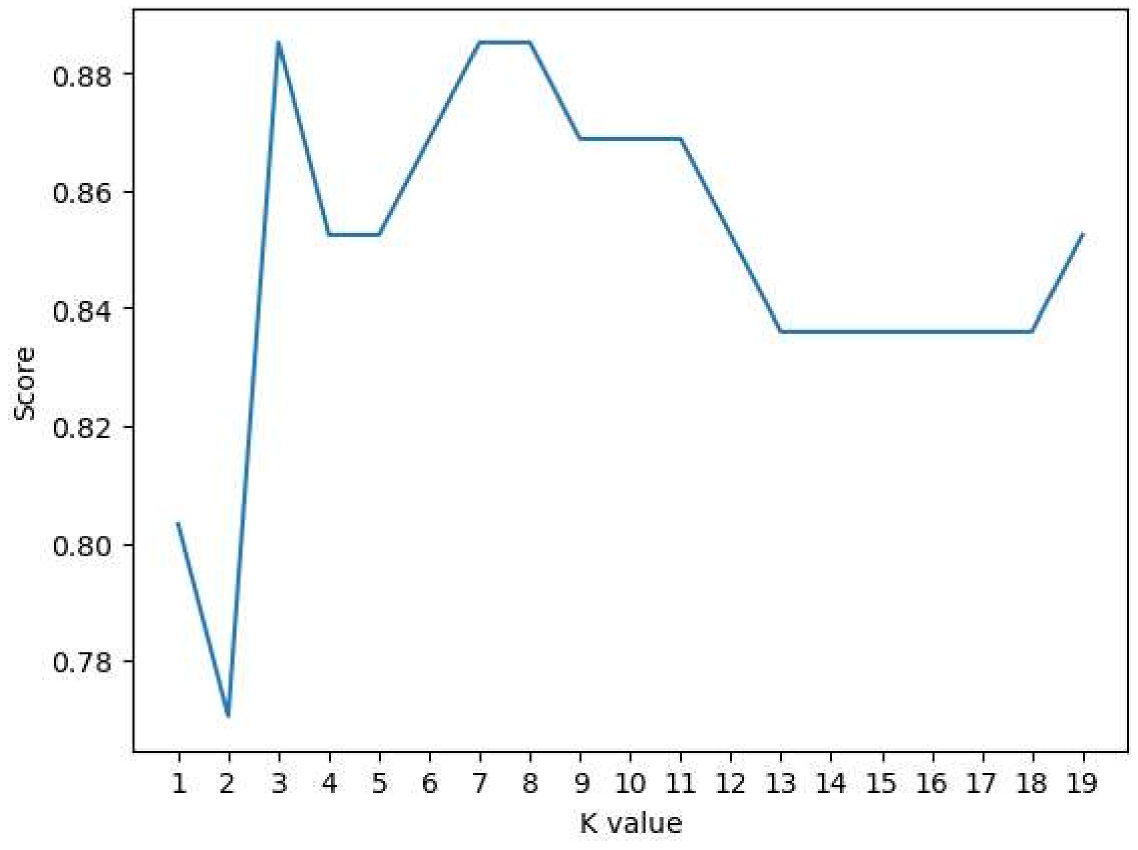
KNN Algorithm



In [58]:
```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train.T, y_train.T)
prediction = knn.predict(x_test.T)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test.T, y_test.T)*100))
```

```
2 NN Score: 77.05%
```

In [59]:
```
# try ro find best k value
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
    scoreList.append(knn2.score(x_test.T, y_test.T))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))
```



```
Maximum KNN Score is 88.52%
```

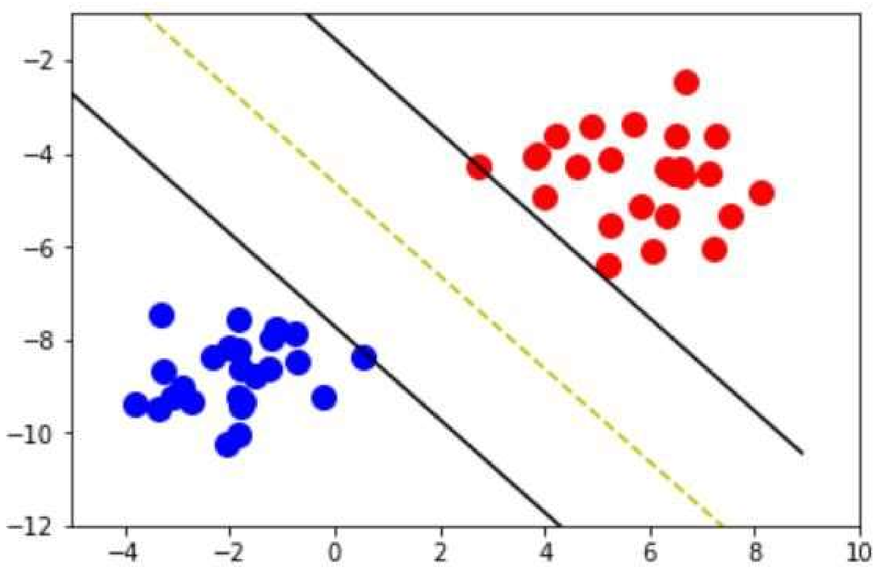
As you can see above if we define k as 3-7-8 we will reach maximum score.

KNN Model's Accuracy is 88.52%

Support Vector Machine (SVM) Algorithm

Now we will use SVM algorithm.

Support Vector Machine Algorithm



```
In [60]: from sklearn.svm import SVC
```

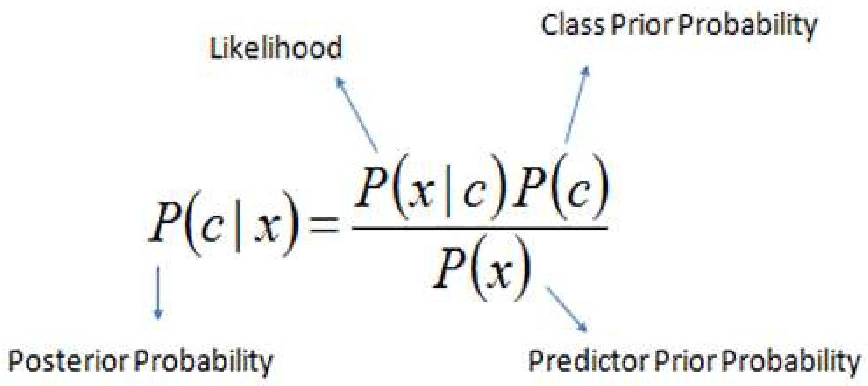
```
In [61]: svm = SVC(random_state = 1)
         svm.fit(x_train.T, y_train.T)

         acc = svm.score(x_test.T,y_test.T)*100
         accuracies['SVM'] = acc
         print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
```

```
Test Accuracy of SVM Algorithm: 88.52%
```

Test Accuracy of SVM Algorithm is 86.89%

Naive Bayes Algorithm

Naive Bayes Algorithm



$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

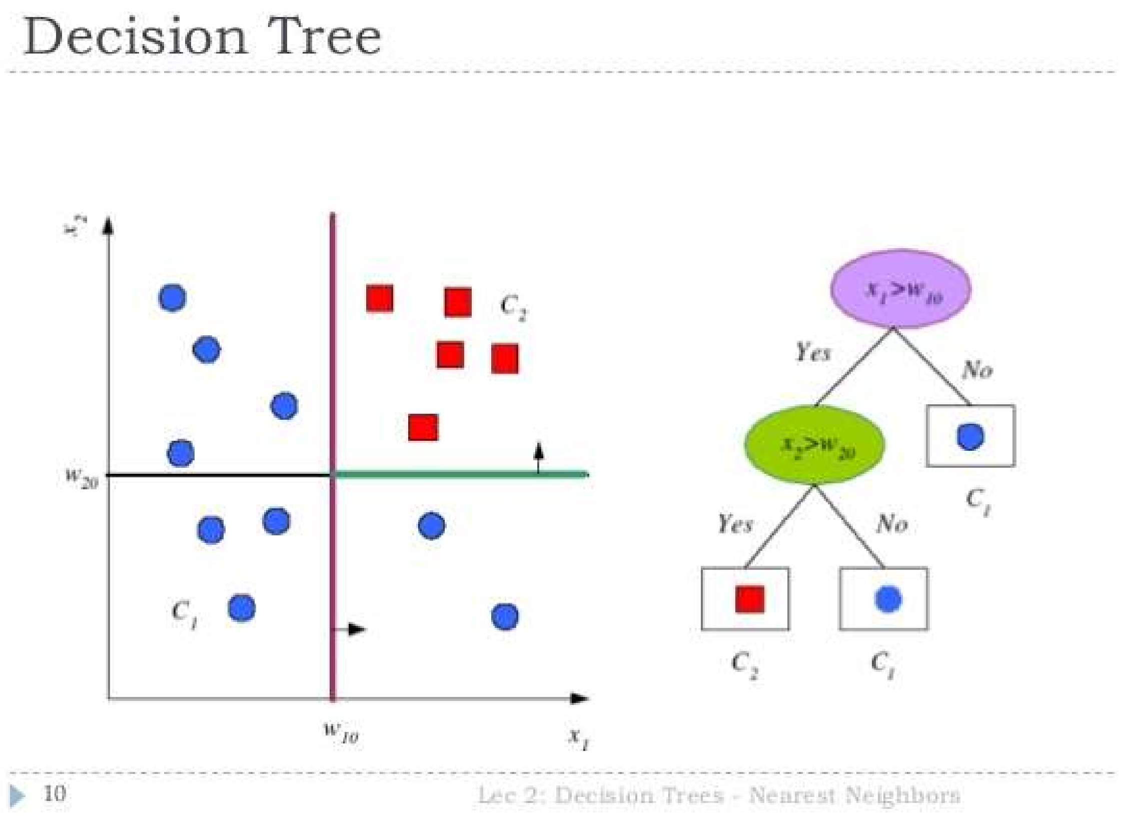
```
In [62]: from sklearn.naive_bayes import GaussianNB
         nb = GaussianNB()
         nb.fit(x_train.T, y_train.T)

         acc = nb.score(x_test.T,y_test.T)*100
         accuracies['Naive Bayes'] = acc
         print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
```

```
Accuracy of Naive Bayes: 86.89%
```

Accuracy of Naive Bayes: 86.89%

Decision Tree Algorithm

Decision Tree Algorithm



In [63]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train.T, y_train.T)

acc = dtc.score(x_test.T, y_test.T)*100
accuracies['Decision Tree'] = acc
print("Decision Tree Test Accuracy {:.2f}%".format(acc))
```

```
Decision Tree Test Accuracy 80.33%
```

Test Accuracy of Decision Tree Algorithm: 78.69%

Random Forest Classification

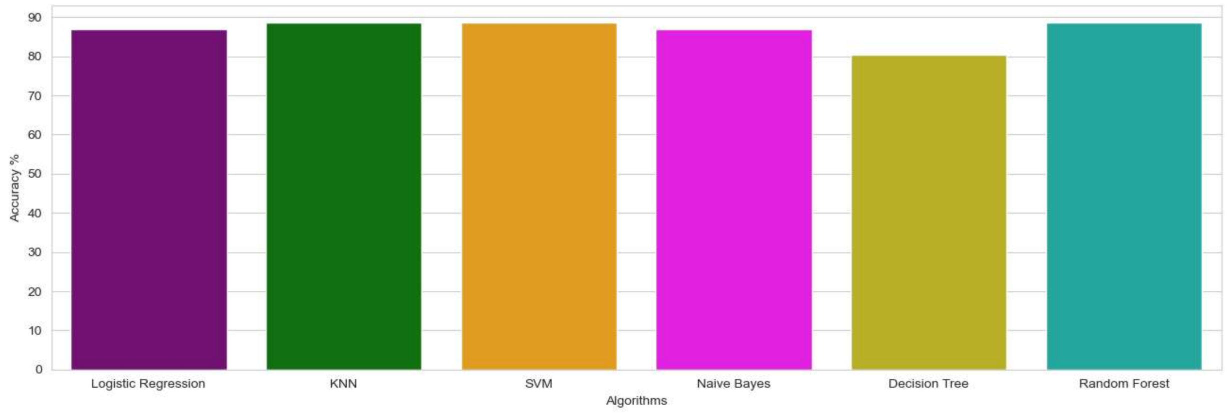
In [64]:
```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train.T, y_train.T)

acc = rf.score(x_test.T,y_test.T)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
```

```
Random Forest Algorithm Accuracy Score : 88.52%
```

Test Accuracy of Random Forest: 88.52%

Comparing Models

In [65]:
```
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=list(accuracies.keys()), y=list(accuracies.values()), palette=color
plt.show()
```



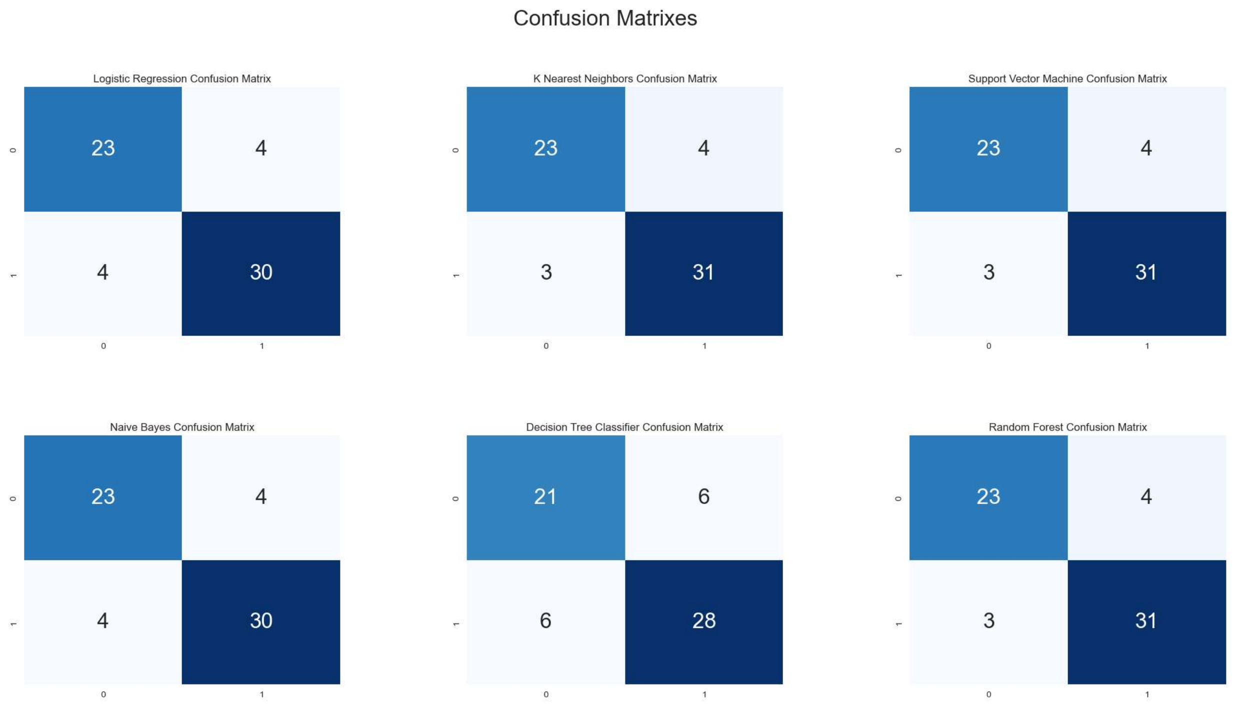
Our models work fine but best of them are KNN and Random Forest with 88.52% of accuracy. Let's look their confusion matrixes.

Confusion Matrix

In [66]:
```
# Predicted values
y_head_lr = lr.predict(x_test.T)
knn3 = KNeighborsClassifier(n_neighbors = 3)
knn3.fit(x_train.T, y_train.T)
y_head_knn = knn3.predict(x_test.T)
y_head_svm = svm.predict(x_test.T)
y_head_nb = nb.predict(x_test.T)
y_head_dtc = dtc.predict(x_test.T)
y_head_rf = rf.predict(x_test.T)
```

In [67]:
```
from sklearn.metrics import confusion_matrix

cm_lr = confusion_matrix(y_test,y_head_lr)
cm_knn = confusion_matrix(y_test,y_head_knn)
cm_svm = confusion_matrix(y_test,y_head_svm)
cm_nb = confusion_matrix(y_test,y_head_nb)
cm_dtc = confusion_matrix(y_test,y_head_dtc)
cm_rf = confusion_matrix(y_test,y_head_rf)
```

In [68]:
```
plt.figure(figsize=(24,12))

plt.suptitle("Confusion Matrixes",fontsize=24)
plt.subplots_adjust(wspace = 0.4, hspace= 0.4)

plt.subplot(2,3,1)
plt.title("Logistic Regression Confusion Matrix")
sns.heatmap(cm_lr,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size":

plt.subplot(2,3,2)
plt.title("K Nearest Neighbors Confusion Matrix")
sns.heatmap(cm_knn,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size"

plt.subplot(2,3,3)
plt.title("Support Vector Machine Confusion Matrix")
sns.heatmap(cm_svm,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size"

plt.subplot(2,3,4)
plt.title("Naive Bayes Confusion Matrix")
sns.heatmap(cm_nb,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size":

plt.subplot(2,3,5)
plt.title("Decision Tree Classifier Confusion Matrix")
sns.heatmap(cm_dtc,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size"

plt.subplot(2,3,6)
plt.title("Random Forest Confusion Matrix")
sns.heatmap(cm_rf,annot=True,cmap="Blues",fmt="d",cbar=False, annot_kws={"size":

plt.show()
```



** Thanks for your time.**