Source code

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

import os

df = pd.read\_csv("heart.csv")

df.head()

df.target.value\_counts()

sns.countplot(x="target", data=df, palette="bwr")

plt.show()

countTdkSakit = len(df[df.target == 0])

countSakit = len(df[df.target == 1])

print("Percentage of patients who are not sick: {:.2f}%".format((countTdkSakit / (len(df.target))\*100)))

print("Percentage of patients who are sick: {:.2f}%".format((countSakit / (len(df.target))\*100)))

sns.countplot(x='sex', data=df, palette="mako\_r")

plt.xlabel("Gender (0 = Female, 1= Male)")

plt.show()

countWanita = len(df[df.sex == 0])

countPria = len(df[df.sex == 1])

print("Presentage of Female Patients: {:.2f}%".format((countWanita / (len(df.sex))\*100)))

print("Presentage of Male Patients: {:.2f}%".format((countPria / (len(df.sex))\*100)))

df.groupby('target').mean()

pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))

plt.title('Heart Disease Frequency based on Age')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.savefig('heartDiseaseAndAges.png')

plt.show()

pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#20639B','#ED553B' ])

plt.title('Heart Disease Frequency based on Gender')

plt.xlabel('Sex (0 = Female, 1 = Male)')

plt.xticks(rotation=0)

plt.legend(["Not Sick", "Sick"])

plt.ylabel('Frequency')

plt.show()

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)], c="green")

plt.legend(["Sick", "Not Sick"])

plt.xlabel("Age")

plt.ylabel("Heart Rate Max")

plt.show()

pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#6C5B7B','#F8B195' ])

plt.title('Heart Disease Frequency based on Slope')

plt.xlabel('The Slope of The Peak Exercise ST Segment ')

plt.xticks(rotation = 0)

plt.ylabel('Frequency')

plt.show()

pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#009999','#00FF00' ])

plt.title('Heart Disease Frequency According To FBS')

plt.xlabel('FBS > 120 mg/dl (1 = true; 0 = false)')

plt.xticks(rotation = 0)

plt.legend(["Not Sick", "Sick"])

plt.ylabel('Frequency Sick/Not Sick')

plt.show()

pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#0000CC','#FFFF99' ])

plt.title('Heart Disease Frequency According To Chest Pain Type')

plt.xlabel('Chest Pain Type')

plt.xticks(rotation = 0)

plt.ylabel('Frequency Sick/Not Sick')

plt.show()

a = pd.get\_dummies(df['cp'], prefix = "cp")

b = pd.get\_dummies(df['thal'], prefix = "thal")

c = pd.get\_dummies(df['slope'], prefix = "slope")

frames = [df, a, b, c]

df = pd.concat(frames, axis = 1)

df.head()

df = df.drop(columns = ['cp', 'thal', 'slope'])

df.head()

y = df.target.values

x\_data = df.drop(['target'], axis = 1)

#Normalization

x = (x\_data - np.min(x\_data)) / (np.max(x\_data) - np.min(x\_data)).values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size = 0.2,random\_state=0)

x\_train = x\_train.T

y\_train = y\_train.T

x\_test = x\_test.T

y\_test = y\_test.T

def initialize(dimension):

    weight = np.full((dimension,1),0.01)

    bias = 0.0

    return weight,bias

#Sigmoid Function

def sigmoid(z):

    y\_head = 1/(1+ np.exp(-z))

    return y\_head

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" : derivative\_bias}

    return cost,gradients

def update(weight,bias,x\_train,y\_train,learningRate,iteration) :

    costList = []

    index = []

    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

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

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,iteration)

    y\_prediction = predict(parameters["weight"],parameters["bias"],x\_test)

    print("Manuel Test Accuracy: {:.2f}%".format((100 - np.mean(np.abs(y\_prediction - y\_test))\*100)/100\*100))

logistic\_regression(x\_train,y\_train,x\_test,y\_test,1,100)

lr = LogisticRegression()

lr.fit(x\_train.T,y\_train.T)

print("Test Accuracy {:.2f}%".format(lr.score(x\_test.T,y\_test.T)\*100))

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors = 2)

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))

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()

print("KNN Score Max {:.2f}%".format((max(scoreList))\*100))

from sklearn.svm import SVC

svm = SVC(random\_state = 1)

svm.fit(x\_train.T, y\_train.T)

print("SVM ALgorithm Test Accuracy: {:.2f}%".format(svm.score(x\_test.T,y\_test.T)\*100))

from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(x\_train.T, y\_train.T)

print("Accuracy of Naive Bayes: {:.2f}%".format(nb.score(x\_test.T,y\_test.T)\*100))

from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

dtc.fit(x\_train.T, y\_train.T)

print("Decision Tree Test Accuracy {:.2f}%".format(dtc.score(x\_test.T, y\_test.T)\*100))

methods = ["Logistic Regression", "KNN", "SVM", "Naive Bayes", "Decision Tree", "Random Forest"]

accuracy = [86.89, 88.52, 86.89, 86.89, 78.69, 88.52]

colors = ["red", "blue", "yellow", "green","purple","orange"]

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()

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)

from sklearn.metrics import confusion\_matrix

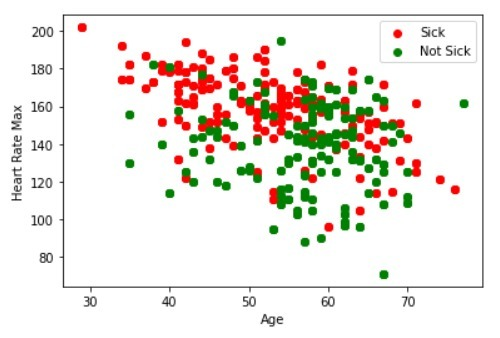
cm\_lr = confusion\_matrix(y\_test,y\_head\_lr)

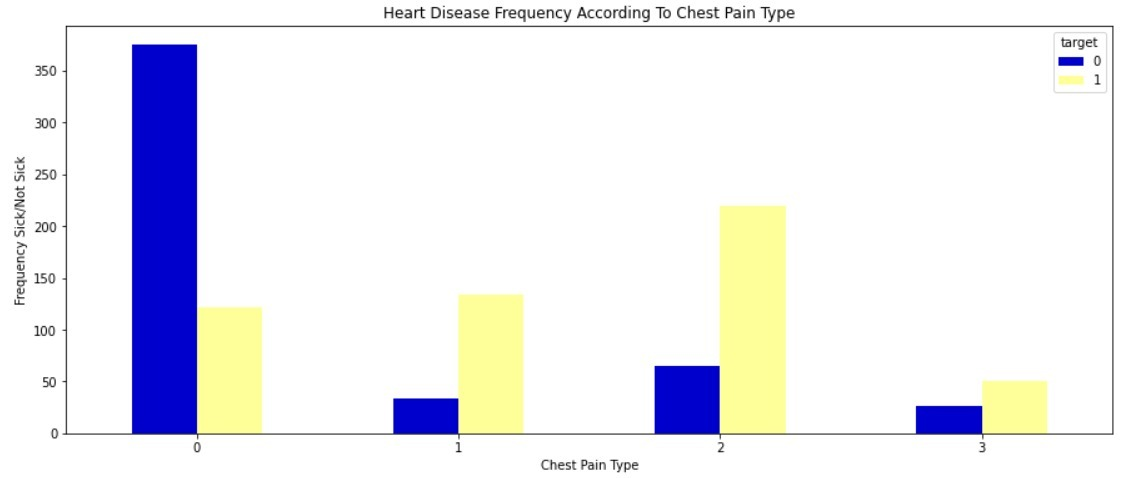
cm\_knn = confusion\_matrix(y\_test,y\_head\_knn)

cm\_svm = confusion\_matrix(y\_test,y\_head\_svm)

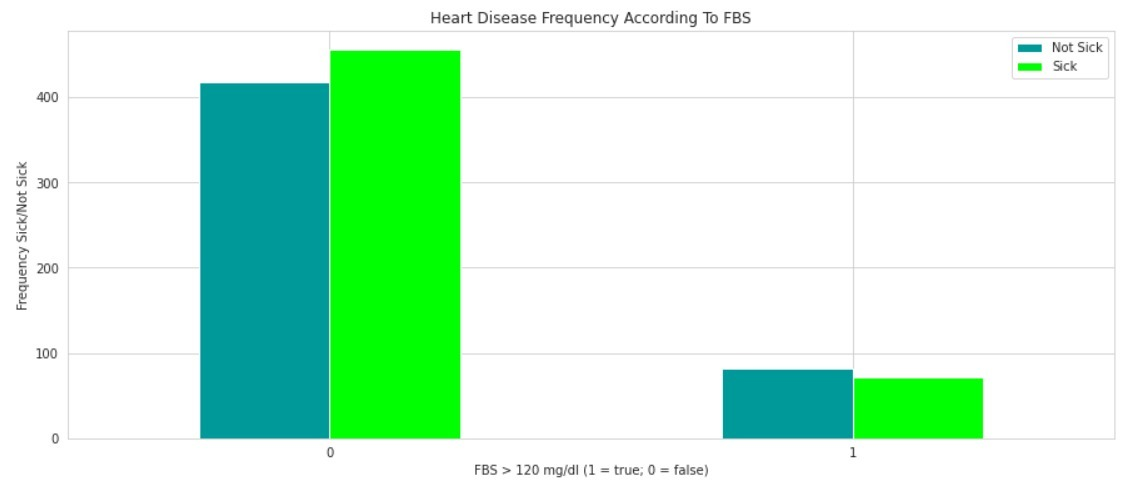
cm\_nb = confusion\_matrix(y\_test,y\_head\_nb)

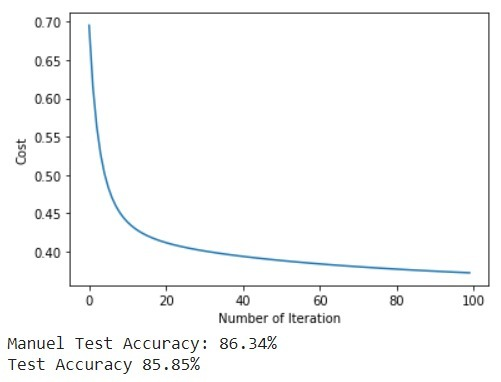
cm\_dtc = confusion\_matrix(y\_test,y\_head\_dtc)

Screenshots

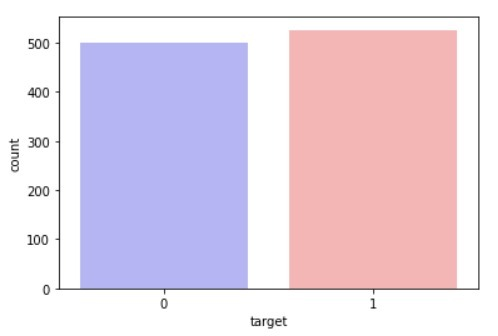
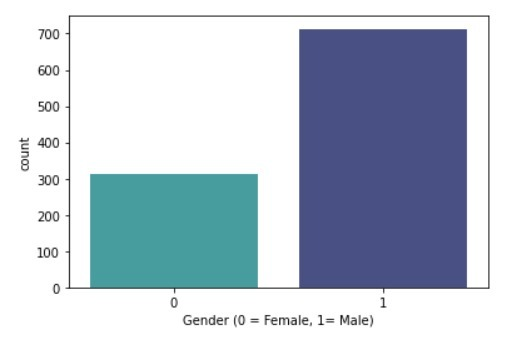
**Plot Graph of the Output**

**Heart Disease Frequency According to chest pain**

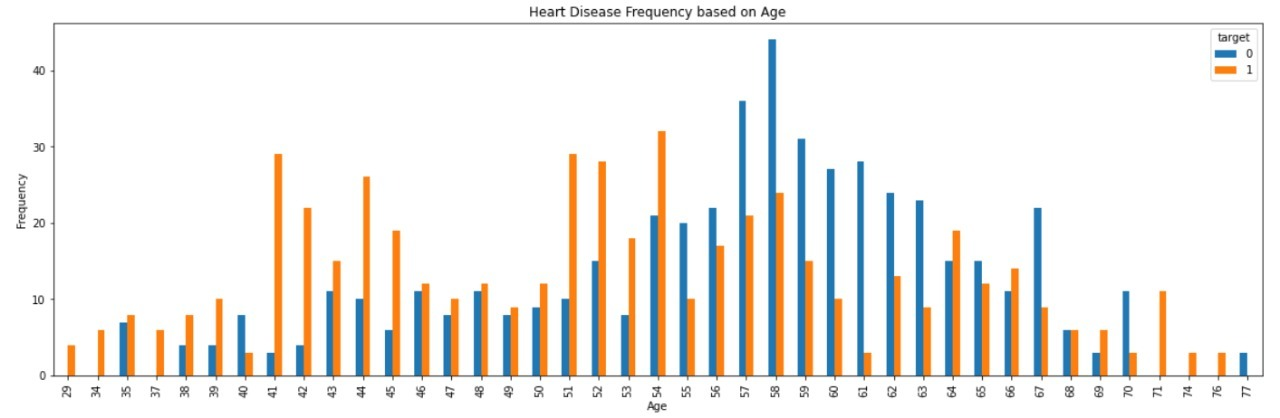
**Heart Disease Frequency According to FBS**

****

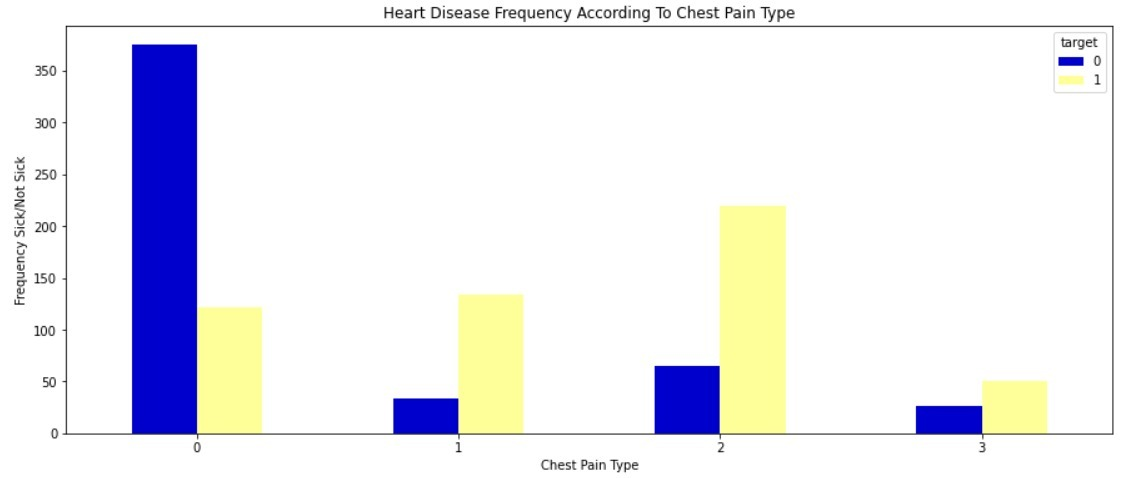
**Accuracy Rate Testing**

****

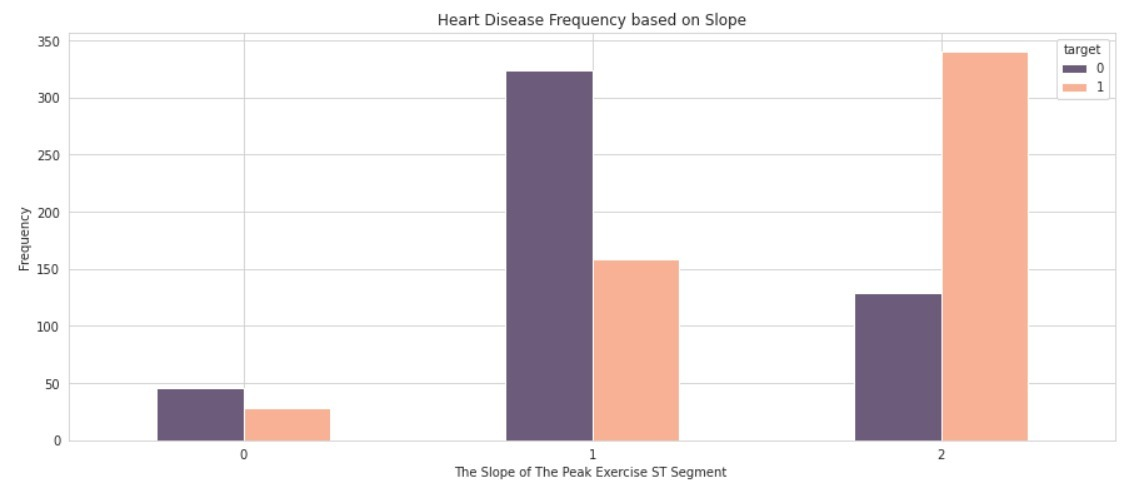
**No.of Patients Affected/ Not Affected**

****

**Heart Disease Frequency based on Age**

****

**Heart Disease Frequency based on Chest Pain Type**

****

**Heart Disease Frequency based on ST slope**