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((co
untTdkSakit / (len(df.target))*100)))
print("Percentage of patients who are sick: {:.2f}%".format((countS
akit / (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 / (l
en(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),colo
r=['#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")
```

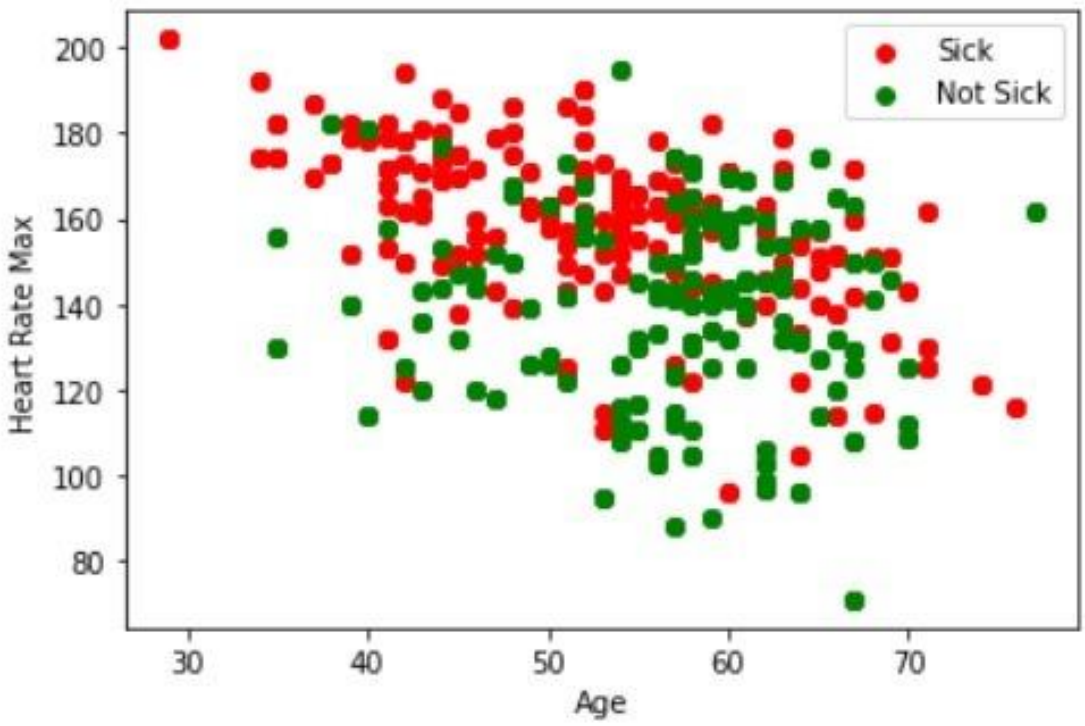
```
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()

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)).v
alues
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, "Derivati
ve 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_trai
n)
        weight = weight - learningRate * gradients["Derivative Weig
ht"]
        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,lear
ningRate,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)
```

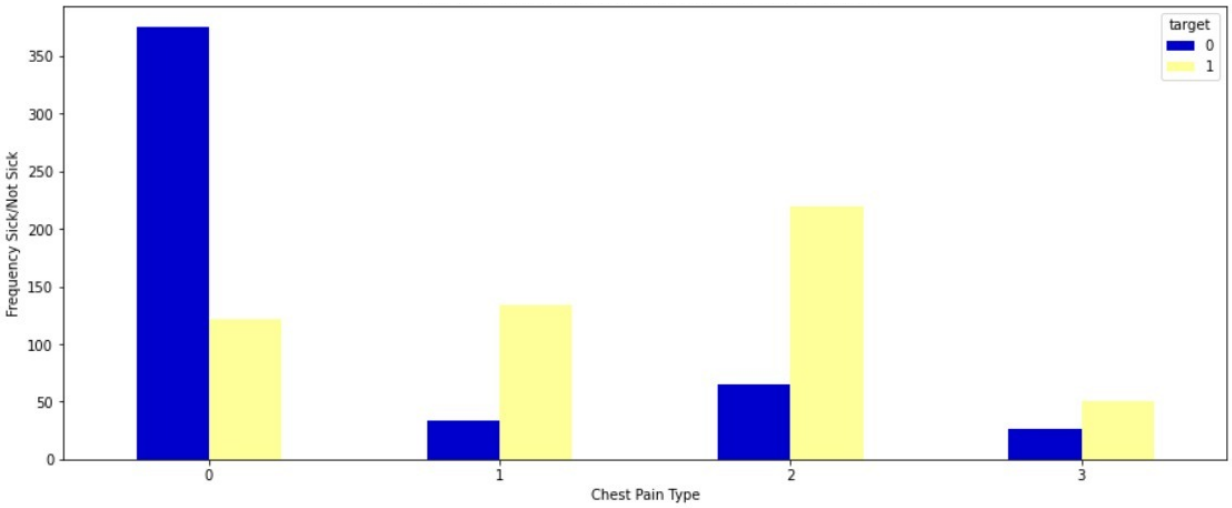
```
print("Test Accuracy {:.2f}%".format(lr.score(x_test.T,y_test.T)*10
0))

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 mea
ns 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_tes
t.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", "Dec
ision 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)
```

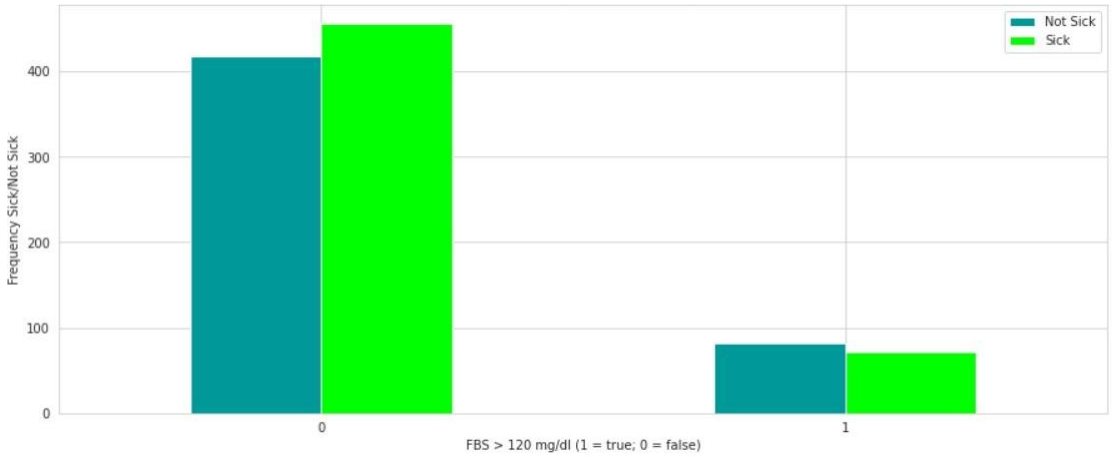
SCREENSHOTS



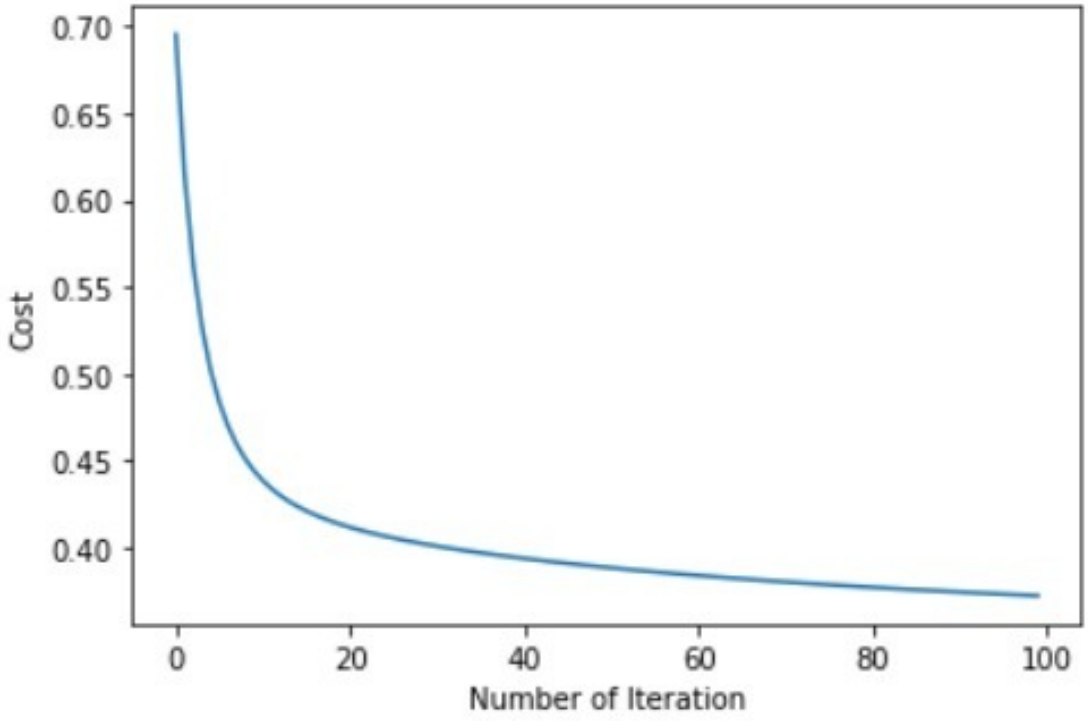
Plot Graph of the Output



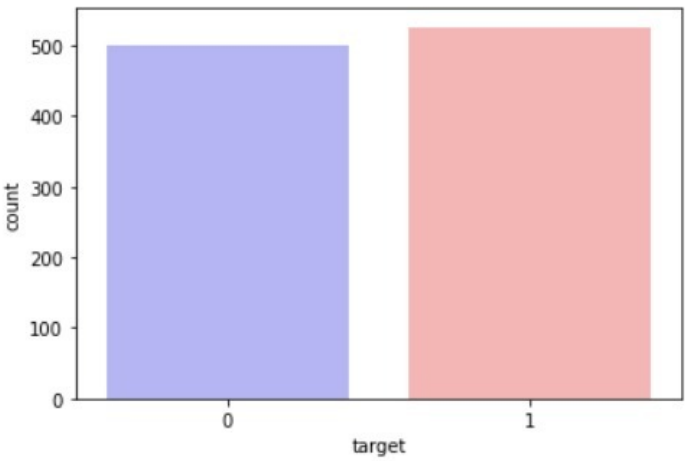
Heart Disease Frequency According to chest pain

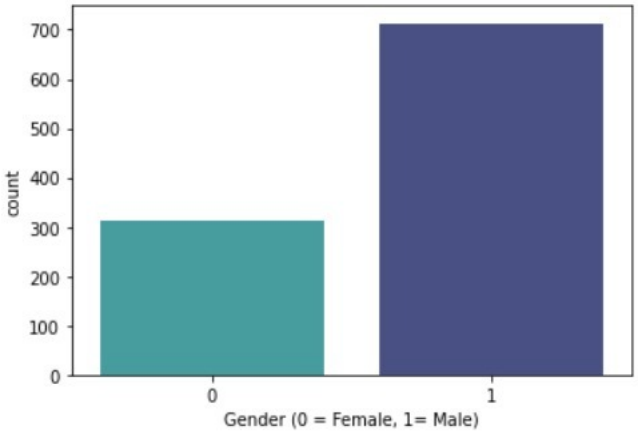


Heart Disease Frequency According to FBS

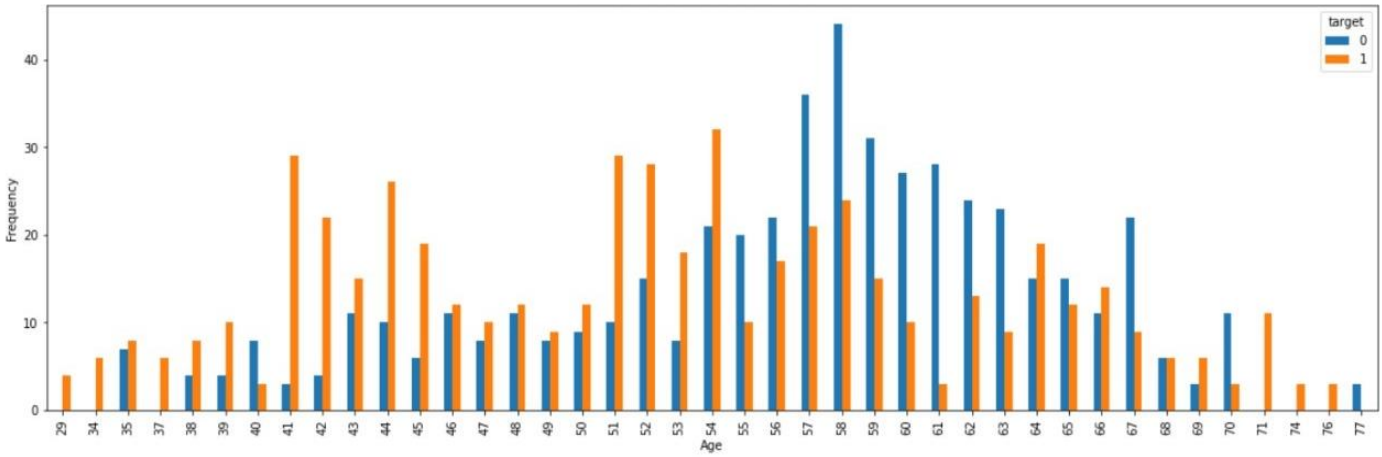


Accuracy Rate Testing

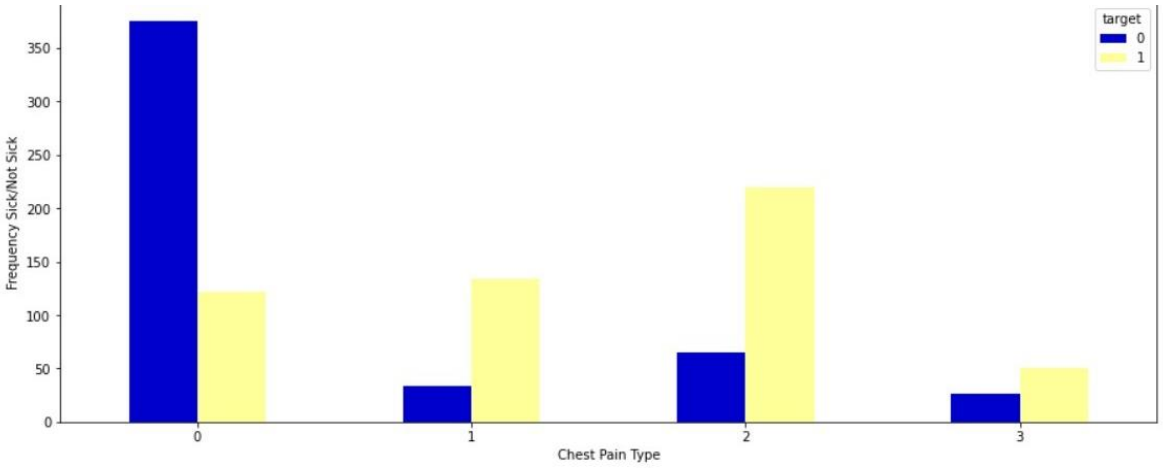




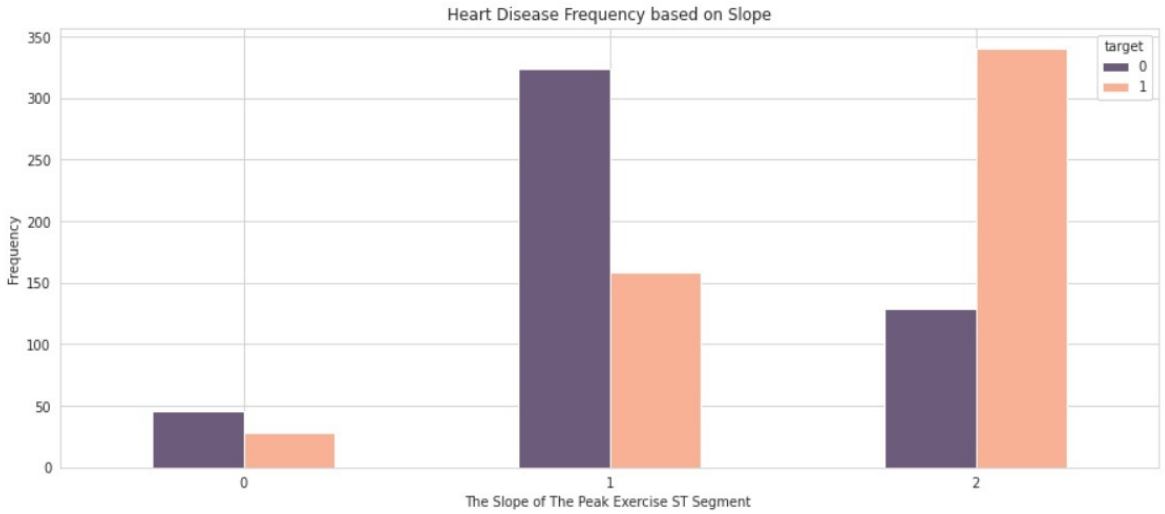
No.of Patients Affected/ Not Affected



Heart Disease Frequency based on Age



Heart Disease Frequency based on Chest Pain Type



Heart Disease Frequency based on ST slope