MACHINE LEARNING & ITS APPLICATIONS CASE STUDY

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

BATCH 2

Under the supervision of

LADE S

CHAKRAVARTHI SIR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING GITAM

(Deemed to be University)

VISAKHAPATNAM, ANDHRAPRADESH

2024

DEPARTMENT OF COMPUTER SCIENCE ANDENGINEERING

GITAM SCHOOL OF TECHNOLOGY GITAM

(GITAM DEEMED TO BE UNIVERSITY)

HEART DISEASE PREDICTION

MEMBERS OF THE TEAM

A. SAI KAUSHIK (VU21CSEN0300006)

K. SRIKAR (VU21CSEN0300058)

P. KUSHWANTH (VU21CSEN0300125)

S. KIRAN (VU21CSEN0300508)

ABSTRACT:

Heart disease remains a leading cause of mortality worldwide, necessitating effective predictive models for early detection and intervention. Then, different models were trained, and predictions are made with different algorithms KNN, Decision Tree, Random Forest, SVM, Logistic Regression. Performance metrics assess model efficacy, while interpretation techniques enhance transparency. Results demonstrate accurate prediction, aiding early intervention and personalized healthcare, advancing predictive analytics in cardiovascular disease management.

INTRODUCTION

Thus, preventing heart diseases has become more than necessary. Good data-driven systems for predicting heart diseases can improve the entire research and prevention process, making sure that more people can live healthy lives. This is where Machine Learning comes into play. Machine Learning helps in predicting the heart diseases, and the predictions made are quite accurate.

Used a variety of Machine Learning algorithms, implemented in Python, to predict the presence of heart disease in a patient. This is a classification problem, with input features as a variety of parameters, and the target variable as a binary variable, predicting whether heart disease is present or not.

Machine Learning algorithms used:

- Logistic Regression (Scikit-learn)
- Naive Bayes (Scikit-learn)
- Support Vector Machine (Linear) (Scikit-learn)

- K-Nearest Neighbours (Scikit-learn)
- Decision Tree (Scikit-learn)
- Random Forest (Scikit-learn)
- XGBoost (Scikit-learn)
- Artificial Neural Network with 1 Hidden layer (Keras)

IMPLEMENTATION:

STEP 1: IMPORTING LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

import os
print(os.listdir())

import warnings
warnings.filterwarnings('ignore')
```

```
['.config', 'heart.csv', 'sample_data']
```

STEP 2: IMPORTING DATASET

```
dataset = pd.read_csv("/content/heart.csv")
```

STEP 3: PRINTING FEW COLOUMNS

```
dataset.head(5)
```

	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

STEP 4: ANALYSING THE TARGET VALUE

```
dataset["target"].describe()
```

```
count    303.000000
mean       0.544554
std        0.498835
min        0.000000
25%        0.000000
50%        1.000000
75%        1.000000
max        1.000000
Name: target, dtype: float64
```

STEP 5: CHECKING CORRELATION BETWEEN COLUMNS

```
print(dataset.corr()["target"].abs().sort_values(ascending=False))
```

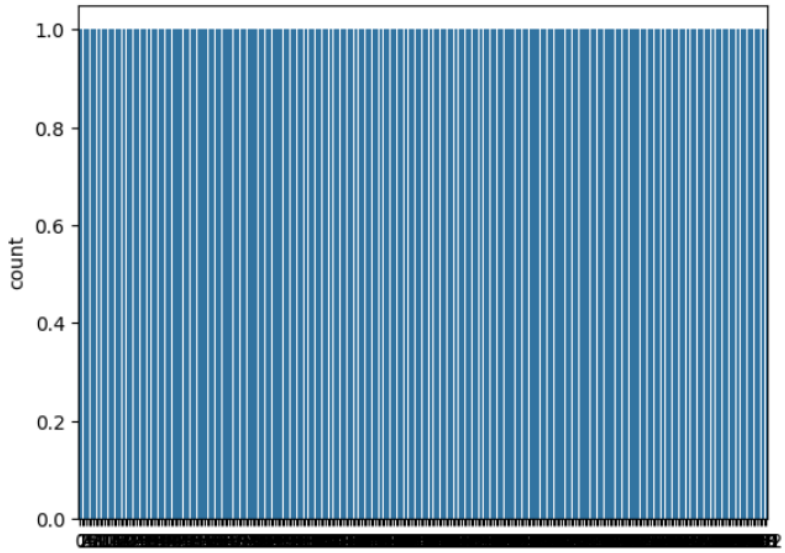
```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64
```

STEP 6: EXPLORATORY DATA ANALYSIS

Analysing the target Value

```
y = dataset["target"]

sns.countplot(y)

target_temp = dataset.target.value_counts()

print(target_temp)
```

```
1    165
0    138
Name: target, dtype: int64
```




```
print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/303,2)))
print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/303,2)))

#Alternatively,
# print("Percentage of patience with heart problems: "+str(y.where(y==1).count()*100/303))
# print("Percentage of patience with heart problems: "+str(y.where(y==0).count()*100/303))

# #Or,
# countNoDisease = len(df[df.target == 0])
# countHaveDisease = len(df[df.target == 1])
```

```
Percentage of patience without heart problems: 45.54
Percentage of patience with heart problems: 54.46
```

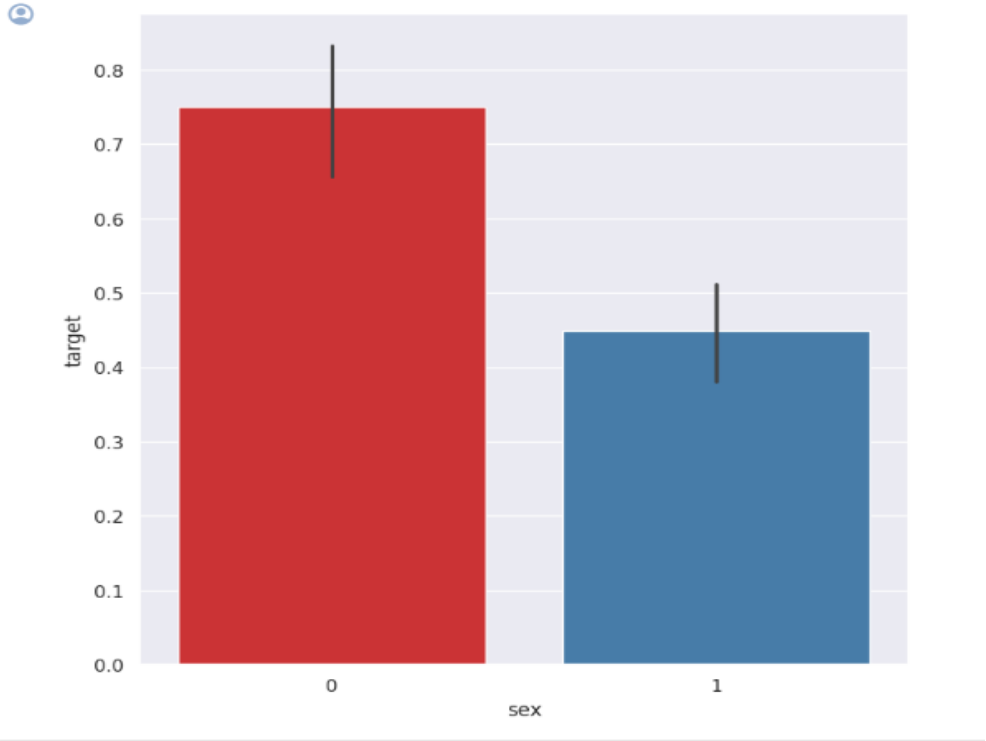
STEP 7: ANALYSING FEATURES

('sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca' and 'thal')

FEATURE 'SEX'

```
[ ] dataset["sex"].unique()

    array([1, 0])
```

```
plt.figure(figsize=(8,8))
sns.barplot(data=dataset, x="sex", y=y ,palette="Set1")
plt.show()
```

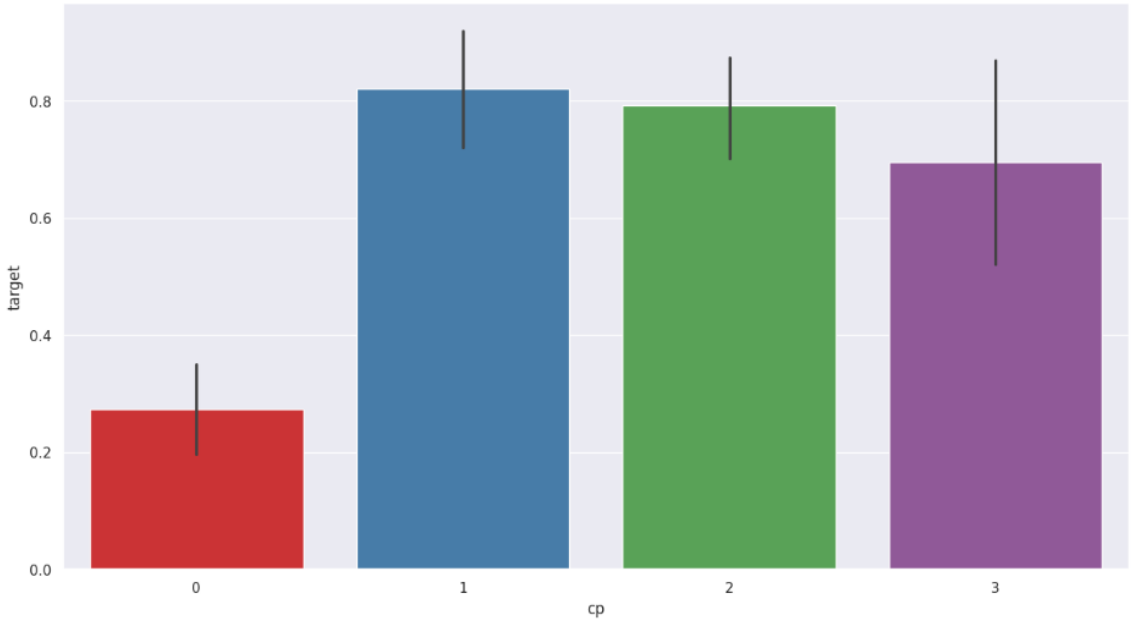


We notice, that females are more likely to have heart problems than males

FEATURE ‘CP’

```
[ ] dataset["cp"].unique()

    array([3, 2, 1, 0])
```

```
sns.barplot(data=dataset, x="cp", y=y,palette="Set1")

<Axes: xlabel='cp', ylabel='target'>
```

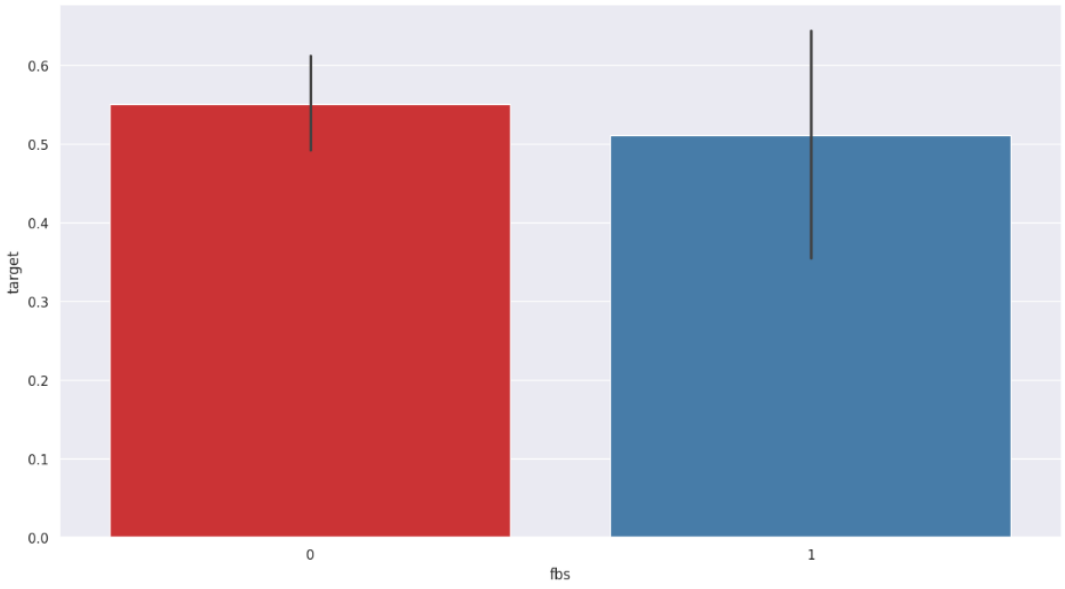


We notice, that chest pain of '0', i.e. the ones with typical angina are much less likely to have heart problems

FEATURE ‘FBS’

```
[ ] dataset["fbs"].unique()

    array([1, 0])
```

```
sns.barplot(data=dataset, x="fbs", y=y,palette="Set1")

<Axes: xlabel='fbs', ylabel='target'>
```



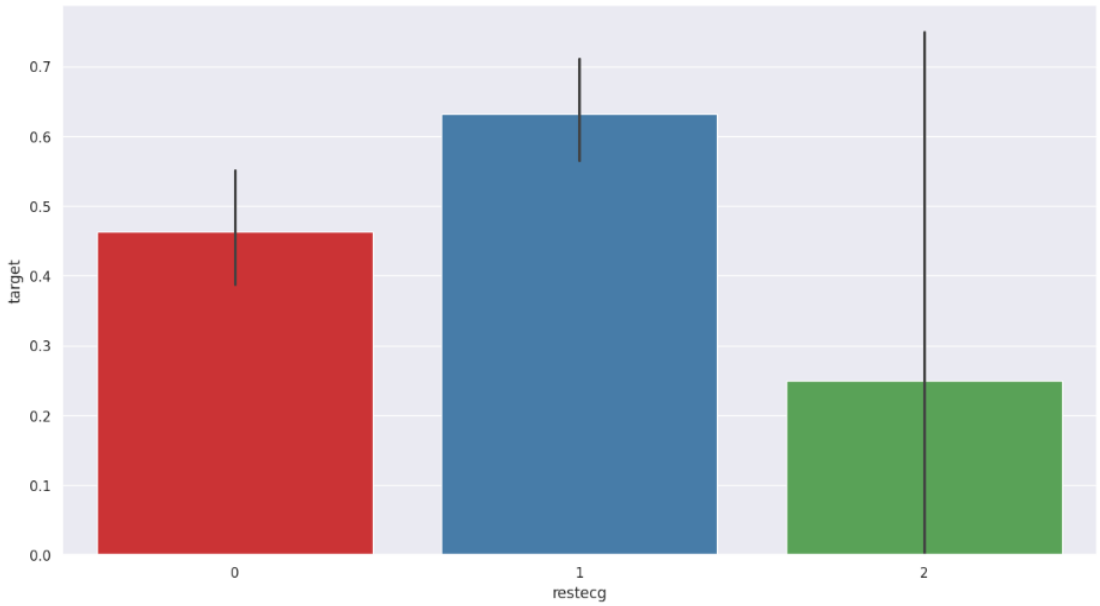
FEATURE ‘RESTECG’

```
[ ] dataset["restecg"].unique()
```

```
array([0, 1, 2])
```

```
sns.barplot(data=dataset, x="restecg", y=y,palette="Set1")
```

```
<Axes: xlabel='restecg', ylabel='target'>
```



We realize that people with restecg '1' and '0' are much more likely to have a heart disease than with restecg '2'

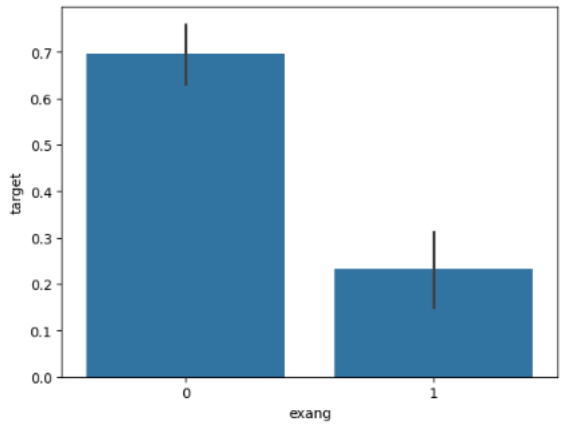
FEATURE ‘EXANG’

```
dataset["exang"].unique()
```

```
array([0, 1])
```

```
[ ] sns.barplot(data=dataset, x="exang", y=y)
```

```
<Axes: xlabel='exang', ylabel='target'>
```



People with exang=1 i.e. Exercise induced angina are much less likely to have heart problems

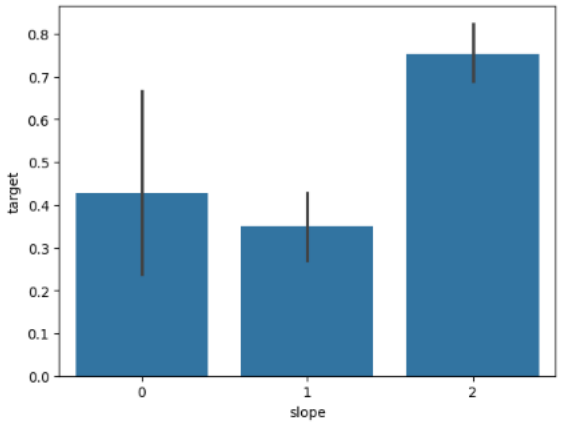
FEATURE ‘SLOPE’

```
[ ] dataset["slope"].unique()
```

```
array([0, 2, 1])
```

```
sns.barplot(data=dataset, x="slope", y=y)
```

```
<Axes: xlabel='slope', ylabel='target'>
```



We observe, that Slope '2' causes heart pain much more than Slope '0' and '1'

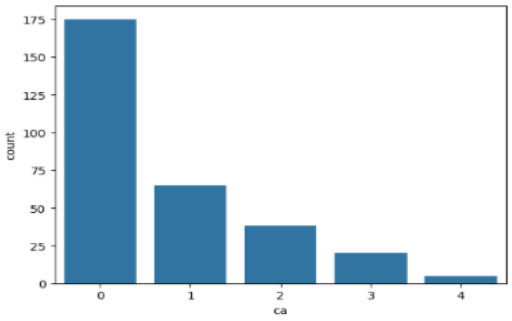
FEATURE ‘CA’

```
[ ] dataset["ca"].unique()
```

```
array([0, 2, 1, 3, 4])
```

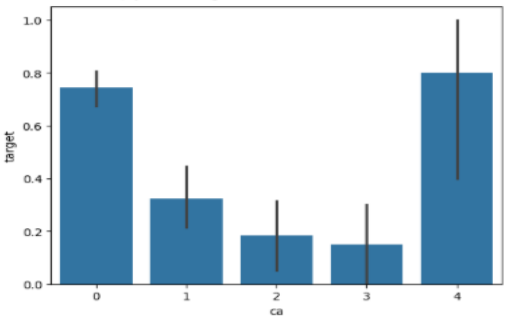
```
[ ] sns.countplot(data=dataset, x="ca")
```

```
<Axes: xlabel='ca', ylabel='count'>
```




```
sns.barplot(data=dataset, x="ca", y=y)
```

```
<Axes: xlabel='ca', ylabel='target'>
```

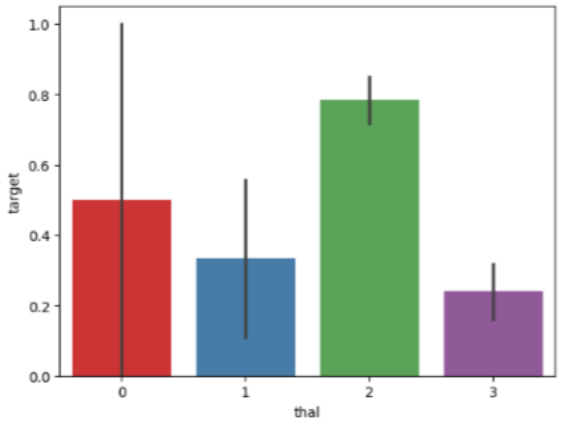


ca=4 has astonishingly large number of heart patients

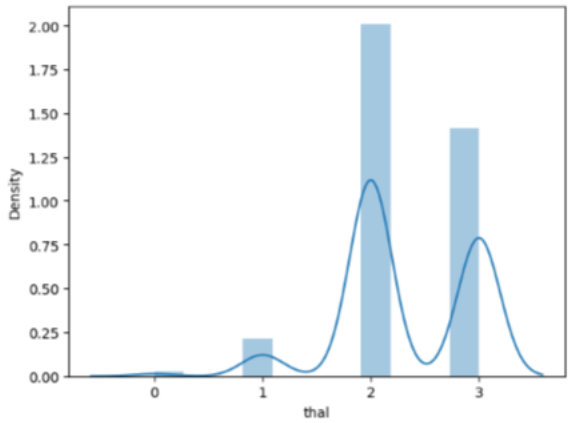
FEATURE ‘THAL’

```
[ ] dataset["thal"].unique()

    array([1, 2, 3, 0])
```

```
    sns.barplot(data=dataset, x="thal", y=y ,palette="Set1")

    <Axes: xlabel='thal', ylabel='target'>
```




```
[ ] sns.distplot(dataset["thal"])

    <Axes: xlabel='thal', ylabel='Density'>
```



STEP 8: TRAIN, TEST, & SPLIT

```
[ ] from sklearn.model_selection import train_test_split

    predictors = dataset.drop("target",axis=1)
    target = dataset["target"]

    X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.20,random_state=0)
```

```
    X_train.shape

    (242, 13)
```

```
[ ] X_test.shape

    (61, 13)
```

```
[ ] Y_train.shape

    (242,)
```

```
[ ] Y_test.shape

    (61,)
```

STEP 9: MODEL FITTING

```
from sklearn.metrics import accuracy_score
```

STEP 10: EXPLORING ALGORITHMS

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train,Y_train)

Y_pred_lr = lr.predict(X_test)
```

```
Y_pred_lr.shape
```

```
(61,)
```

```
score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 85.25 %
```

NAÏVE BAYES

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
```

```
Y_pred_nb.shape
```

```
(61,)
```

```
score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

```
The accuracy score achieved using Naive Bayes is: 85.25 %
```

SUPPORT VECTOR MACHINE(SVM)

```
[ ] from sklearn import svm

    sv = svm.SVC(kernel='linear')

    sv.fit(X_train, Y_train)

    Y_pred_svm = sv.predict(X_test)
```

```
Y_pred_svm.shape
```

```
(61,)
```

```
[ ] score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

    print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 81.97 %
```

K NEAREST NEIGHBOURS

```
[ ] from sklearn.neighbors import KNeighborsClassifier

    knn = KNeighborsClassifier(n_neighbors=7)
    knn.fit(X_train,Y_train)
    Y_pred_knn=knn.predict(X_test)
```

```
[ ] Y_pred_knn.shape

    (61,)
```

```
[ ] score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)

    print("The accuracy score achieved using KNN is: "+str(score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 67.21 %
```

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0


for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

```
[ ] print(Y_pred_dt.shape)
```

```
(61,)
```

```
[ ] score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

    print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 81.97 %
```

RANDOM FOREST

```
[ ] from sklearn.ensemble import RandomForestClassifier

    max_accuracy = 0


    for x in range(50):
        rf = RandomForestClassifier(random_state=x)
        rf.fit(X_train,Y_train)
        Y_pred_rf = rf.predict(X_test)
        current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
        if(current_accuracy>max_accuracy):
            max_accuracy = current_accuracy
            best_x = x

    #print(max_accuracy)
    #print(best_x)

    rf = RandomForestClassifier(random_state=best_x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
```

```
Y_pred_rf.shape
```

```
(61,)
```

```
[ ] score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)

    print("The accuracy score achieved using Decision Tree is: "+str(score_rf)+" %")
```

```
The accuracy score achieved using Decision Tree is: 88.52 %
```

XGBOOST

```
[ ] import xgboost as xgb

    xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
    xgb_model.fit(X_train, Y_train)

    Y_pred_xgb = xgb_model.predict(X_test)
```

```
Y_pred_xgb.shape
```

```
(61,)
```

```
[ ] score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

    print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 83.61 %
```

NEURAL NETWORK

```
[ ] from keras.models import Sequential
    from keras.layers import Dense
```

```
[ ] # https://stats.stackexchange.com/a/136542 helped a lot in avoiding overfitting

    model = Sequential()
    model.add(Dense(11,activation='relu',input_dim=13))
    model.add(Dense(1,activation='sigmoid'))

    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.fit(X_train,Y_train,epochs=300)
```

```
Epoch 1/300
8/8 [==============================] - 1s 4ms/step - loss: 71.0623 - accuracy: 0.4587
Epoch 2/300
8/8 [==============================] - 0s 3ms/step - loss: 62.7524 - accuracy: 0.4587
Epoch 3/300
8/8 [==============================] - 0s 3ms/step - loss: 54.4934 - accuracy: 0.4587
Epoch 4/300
8/8 [==============================] - 0s 3ms/step - loss: 45.9682 - accuracy: 0.4587
Epoch 5/300
8/8 [==============================] - 0s 3ms/step - loss: 37.1527 - accuracy: 0.4587
Epoch 6/300
8/8 [==============================] - 0s 4ms/step - loss: 28.0255 - accuracy: 0.4587
Epoch 7/300
8/8 [==============================] - 0s 3ms/step - loss: 19.0152 - accuracy: 0.4587
```

```
[ ] Y_pred_nn = model.predict(X_test)

    2/2 [==============================] - 0s 6ms/step
```

```
[ ] Y_pred_nn.shape

    (61, 1)
```

```
[ ] rounded = [round(x[0]) for x in Y_pred_nn]

    Y_pred_nn = rounded
```

```
    score_nn = round(accuracy_score(Y_pred_nn,Y_test)*100,2)

    print("The accuracy score achieved using Neural Network is: "+str(score_nn)+" %")

    #Note: Accuracy of 85% can be achieved on the test set, by setting epochs=2000, and number of nodes = 11.

    The accuracy score achieved using Neural Network is: 81.97 %
```

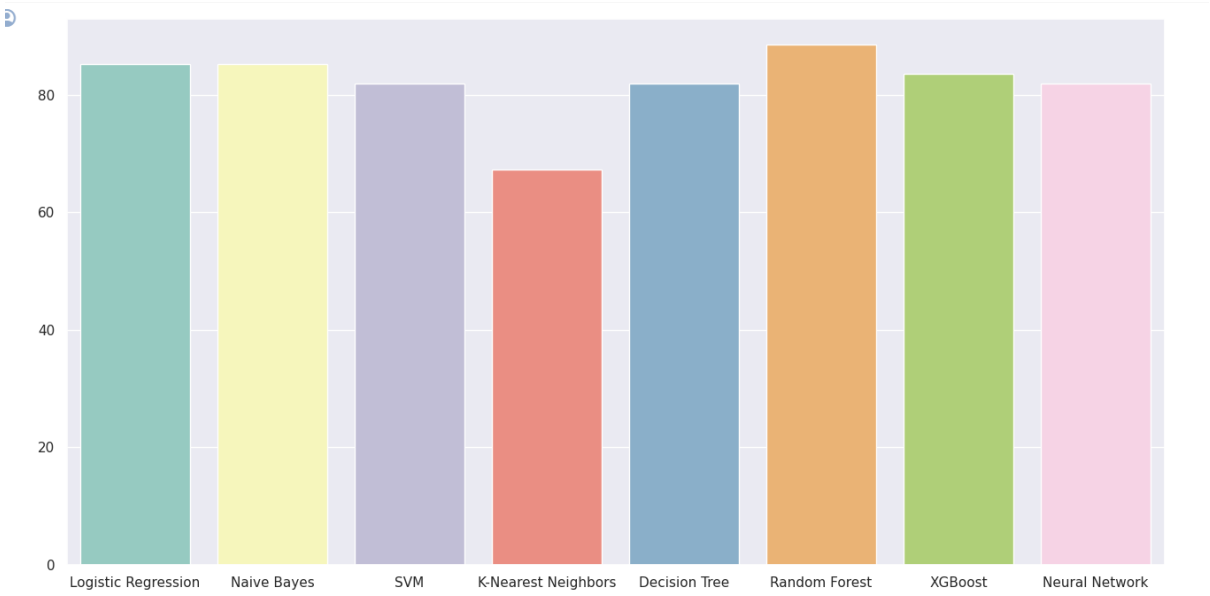
STEP 11: FINAL OUTPUT SCORE

```
    scores = [score_lr,score_nb,score_svm,score_knn,score_dt,score_rf,score_xgb,score_nn]
    algorithms = ["Logistic Regression","Naive Bayes","SVM","K-Nearest Neighbors","Decision Tree","Random Forest","XGBoost","Neural Network"]

    for i in range(len(algorithms)):
        print("The accuracy score achieved using "+algorithms[i]+" is: "+str(scores[i])+" %")
```

```
    The accuracy score achieved using Logistic Regression is: 85.25 %
    The accuracy score achieved using Naive Bayes is: 85.25 %
    The accuracy score achieved using SVM is: 81.97 %
    The accuracy score achieved using K-Nearest Neighbors is: 67.21 %
    The accuracy score achieved using Decision Tree is: 81.97 %
    The accuracy score achieved using Random Forest is: 88.52 %
    The accuracy score achieved using XGBoost is: 83.61 %
    The accuracy score achieved using Neural Network is: 81.97 %
```

```
[ ] print(len(dataset))
    print(len(algorithms))
    print(len(scores))

    303
    8
    8
```

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt
    plt.figure(figsize=(16, 8))  # Adjust the width and height as needed
    sns.barplot(x=algorithms, y=scores,palette='Set3')
    plt.show()
```



CONCLUSION:

In conclusion, this study highlights the effectiveness of machine learning algorithms in predicting heart disease risk based on diverse clinical parameters. By employing KNN, Decision Tree, Random Forest, SVM, Logistic Regression, we achieved accurate predictions and identified key predictors through feature selection. The models' performance metrics indicate their efficacy in facilitating early intervention and personalized healthcare strategies. This research underscores the potential of machine learning in mitigating the burden of heart disease and improving patient outcomes through proactive intervention. Future research could focus on refining models with larger datasets and incorporating emerging technologies for even more precise risk assessment.