Ram Lal Anand College

(University of Delhi)

DATA MINING PROJECT

Department of Computer Science

Heart Disease Prediction

Name of Course	**:**	**B.Sc. (H) Computer Science**
Semester	**:**	**6^{th}**
Name of the Paper	**:**	**Data Mining**
Paper Code	**:**	**32347611**
Name of the Student	**:**	**Abhishek Kumar**
Examination Roll No	**:**	**20058570051**

About the Project:

Heart-Disease-Prediction:

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.

The project involved analysis of the heart disease patient dataset with proper data processing. Then, different models were trained and predictions are made with different algorithms KNN, Decision Tree, Random Forest, SVM, Logistic Regression etc This is the jupyter notebook code and dataset I've used for my Kaggle kernel 'Binary Classification with Sklearn and Keras'

I've 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:

1. Logistic Regression (Scikit-learn)
2. Naive Bayes (Scikit-learn)
3. Support Vector Machine (Linear) (Scikit-learn)
4. K-Nearest Neighbours (Scikit-learn)
5. Decision Tree (Scikit-learn)
6. Random Forest (Scikit-learn)
7. XGBoost (Scikit-learn)
8. Artificial Neural Network with 1 Hidden layer (Keras)

Accuracy achieved: 100% (Decision Tree, Random Forest & XGBoost)

Dataset used: https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

Heart Disease Prediction By Abhishek Kumar

I. Importing essential 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']
```

II. Importing and understanding our dataset

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

Verifying it as a 'dataframe' object in pandas

```
type(dataset)
```

```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None,
dtype: Dtype | None=None, copy: bool | None=None) -> None
```

```
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure
```

Shape of dataset

```
dataset.shape
```

```
(1025, 14)
```

Printing out a few columns

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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	52	1	0	125	212	0	1	168	0	1.0	2	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	1	
4	62	0	0	138	294	1	1	106	0	1.9	1	3	

```
dataset.sample(5)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	t
720	64	1	2	140	335	0	1	158	0	0.0	2	0	
477	57	1	2	128	229	0	0	150	0	0.4	1	1	
115	61	0	0	145	307	0	0	146	1	1.0	1	0	
373	58	1	1	120	284	0	0	160	0	1.8	1	0	
826	42	1	2	130	180	0	1	150	0	0.0	2	0	

Description

```
dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	102
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

###Luckily, we have no missing values

Let's understand our columns better:

```
info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angin
```

```
for i in range(len(info)):
    print(dataset.columns[i]+":\t\t\t"+info[i])
```

```
age:                    age
sex:                    1: male, 0: female
cp:                     chest pain type, 1: typical angina, 2: atypical angina, 3: nc
trestbps:                       resting blood pressure
chol:                    serum cholestoral in mg/dl
fbs:                    fasting blood sugar > 120 mg/dl
restecg:                        resting electrocardiographic results (values 0,1,2)
thalach:                         maximum heart rate achieved
exang:                  exercise induced angina
oldpeak:                        oldpeak = ST depression induced by exercise relative
slope:                  the slope of the peak exercise ST segment
ca:                     number of major vessels (0-3) colored by flourosopy
thal:                   thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

Analysing the 'target' variable

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

	target
count	1025.000000
mean	0.513171
std	0.500070
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000

dtype: float64

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

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

Clearly, this is a classification problem, with the target variable having values '0' and '1'

Checking correlation between columns

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

```
target      1.000000
oldpeak     0.438441
exang       0.438029
cp          0.434854
thalach     0.422895
ca          0.382085
slope       0.345512
thal        0.337838
sex         0.279501
age         0.229324
trestbps    0.138772
restecg     0.134468
chol        0.099966
fbs         0.041164
Name: target, dtype: float64
```

```
#This shows that most columns are moderately correlated with target, but 'fbs' is very we
```

Exploratory Data Analysis (EDA)

First, analysing the target variable:

```
sns.countplot(x=dataset["target"].astype(str), palette={"1": "orange", "0": "steelblue"})

plt.title("Distribution of Target Values")
plt.xlabel("Target")
plt.ylabel("Count")
plt.show()

# Print value counts
target_temp = dataset["target"].value_counts()
print(target_temp)
```


```
target
1    526
0    499
Name: count, dtype: int64
```

```
print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/1025
print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/1025,2)

#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: 48.68
Percentage of patience with heart problems: 51.32
```

We'll analyse 'sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca' and 'thal' features

Analysing the 'Sex' feature

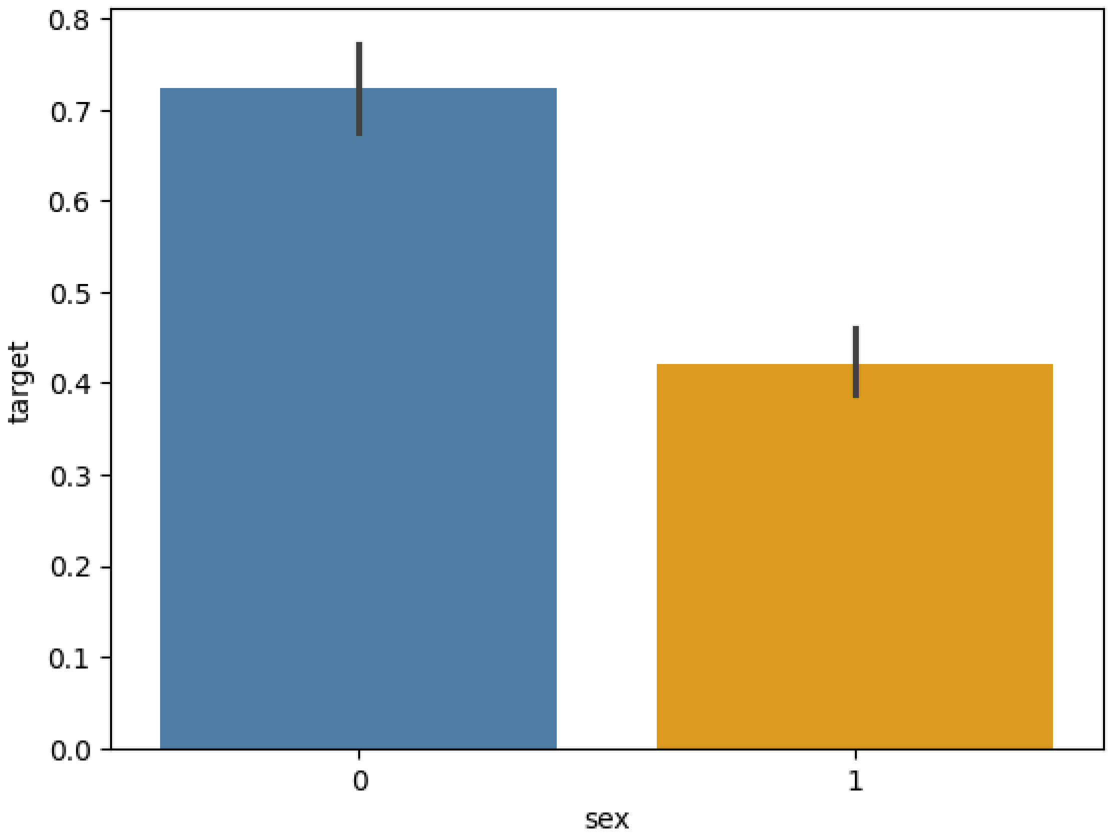
```
dataset["sex"].unique()
```

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

We notice, that as expected, the 'sex' feature has 2 unique features

```
sns.barplot(x="sex", y="target", data=dataset, palette=["steelblue", "orange"])
```

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



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

Analysing the 'Chest Pain Type' feature

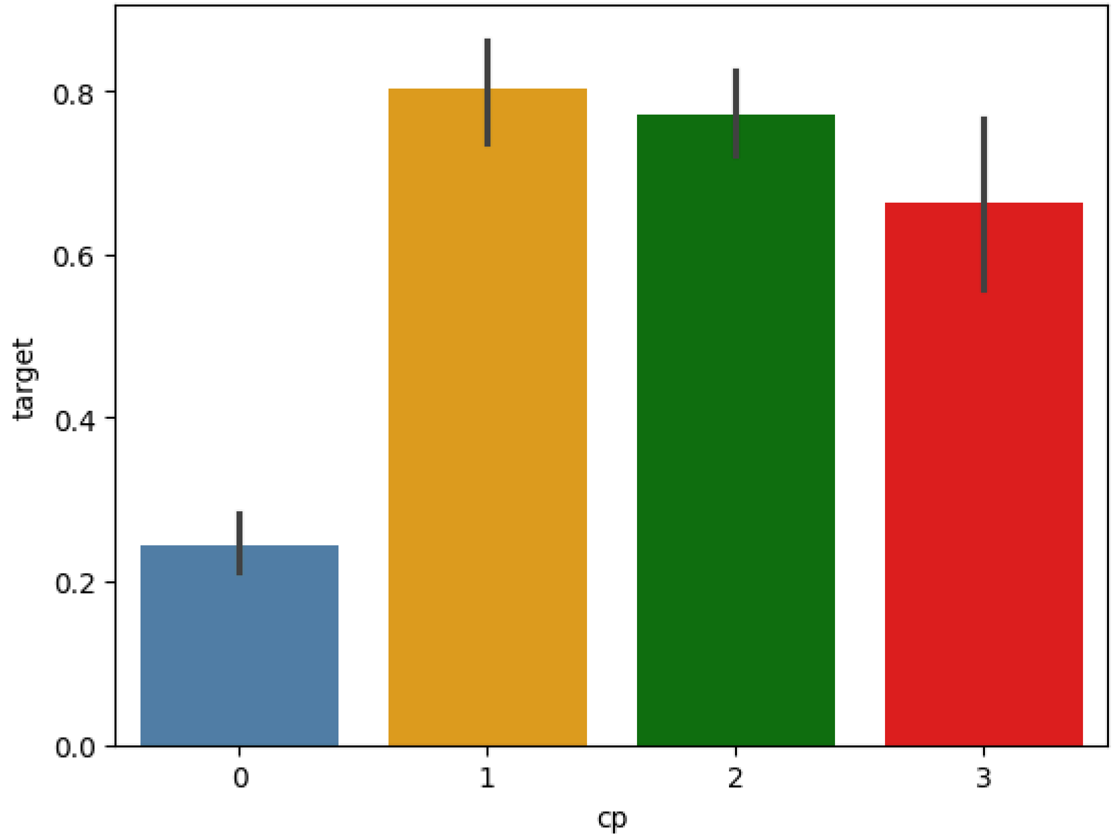
```
dataset["cp"].unique()
```

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

As expected, the CP feature has values from 0 to 3

```
sns.barplot(x="cp", y="target", data=dataset, palette=["steelblue", "orange", "green", "r
```

```
<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

Analysing the FBS feature

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

	fbs
count	1025.000000
mean	0.149268
std	0.356527
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

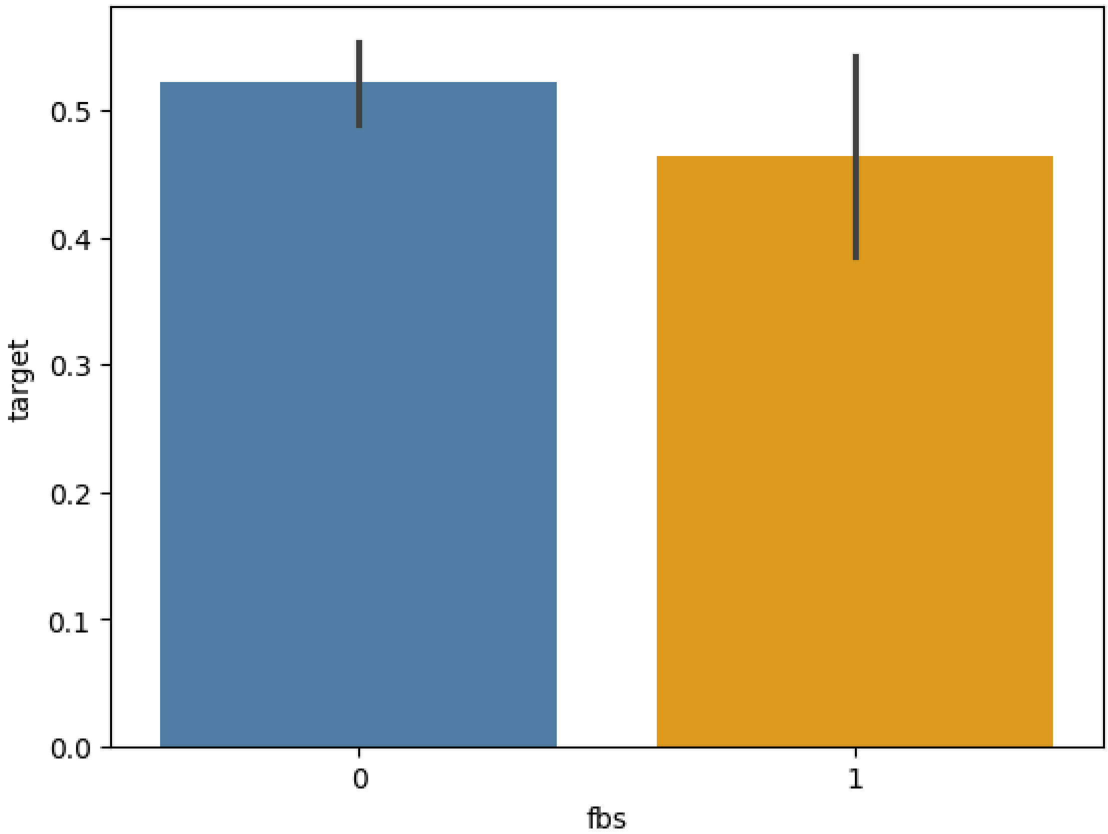
dtype: float64

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

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

```
sns.barplot(x="fbs", y="target", data=dataset, palette=["steelblue", "orange"])
```

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



Nothing extraordinary here

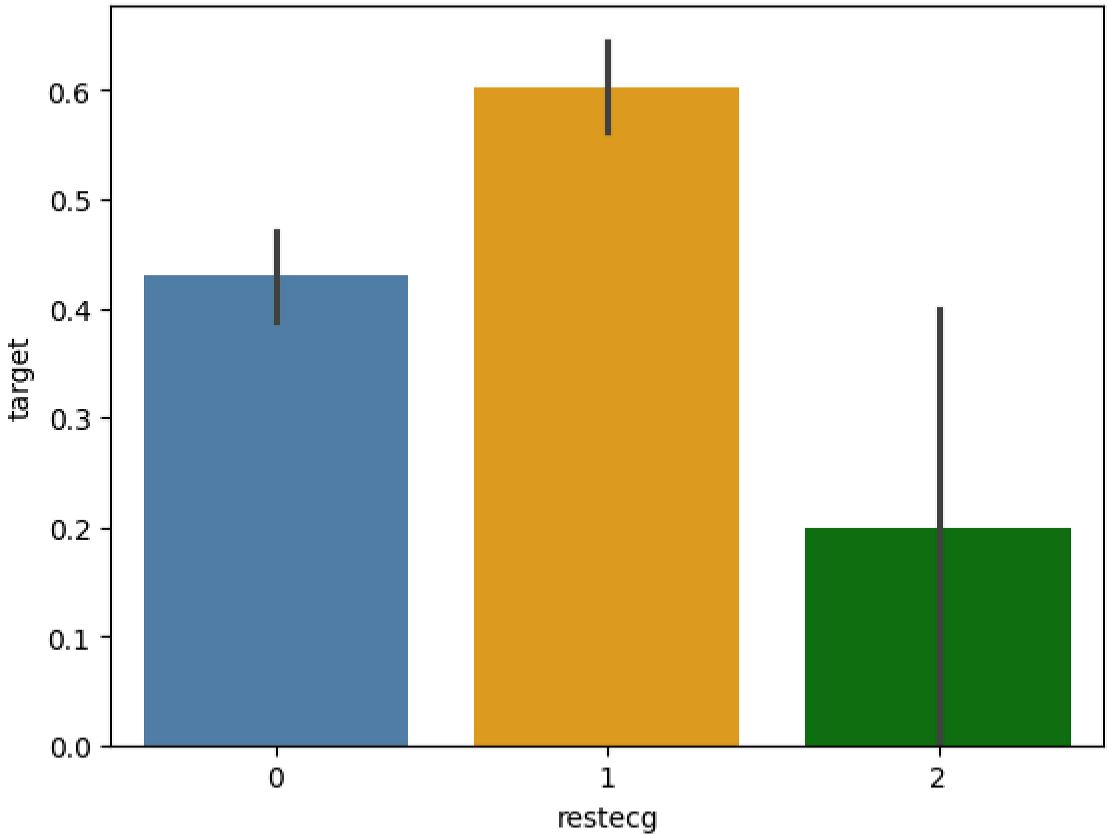
Analysing the restecg feature

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

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

```
sns.barplot(x="restecg", y="target", data=dataset, palette=["steelblue", "orange", "green
```

```
<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'

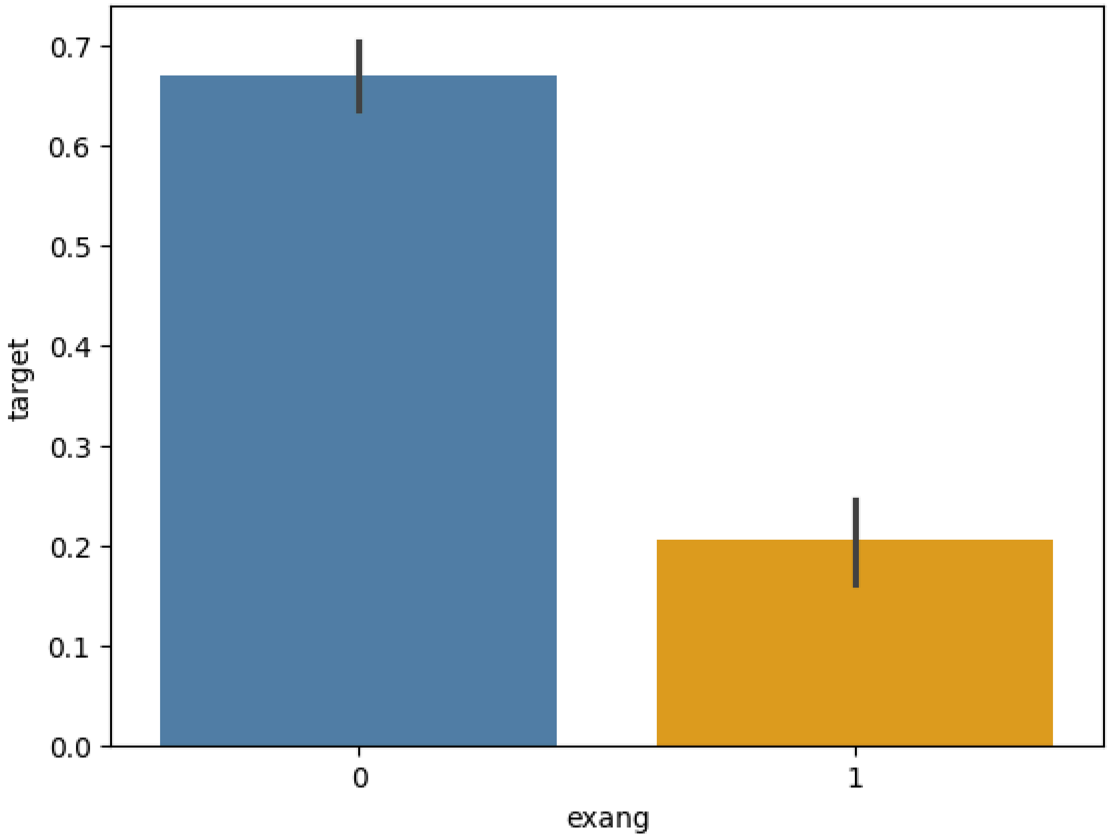
Analysing the 'exang' feature

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

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

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

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



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

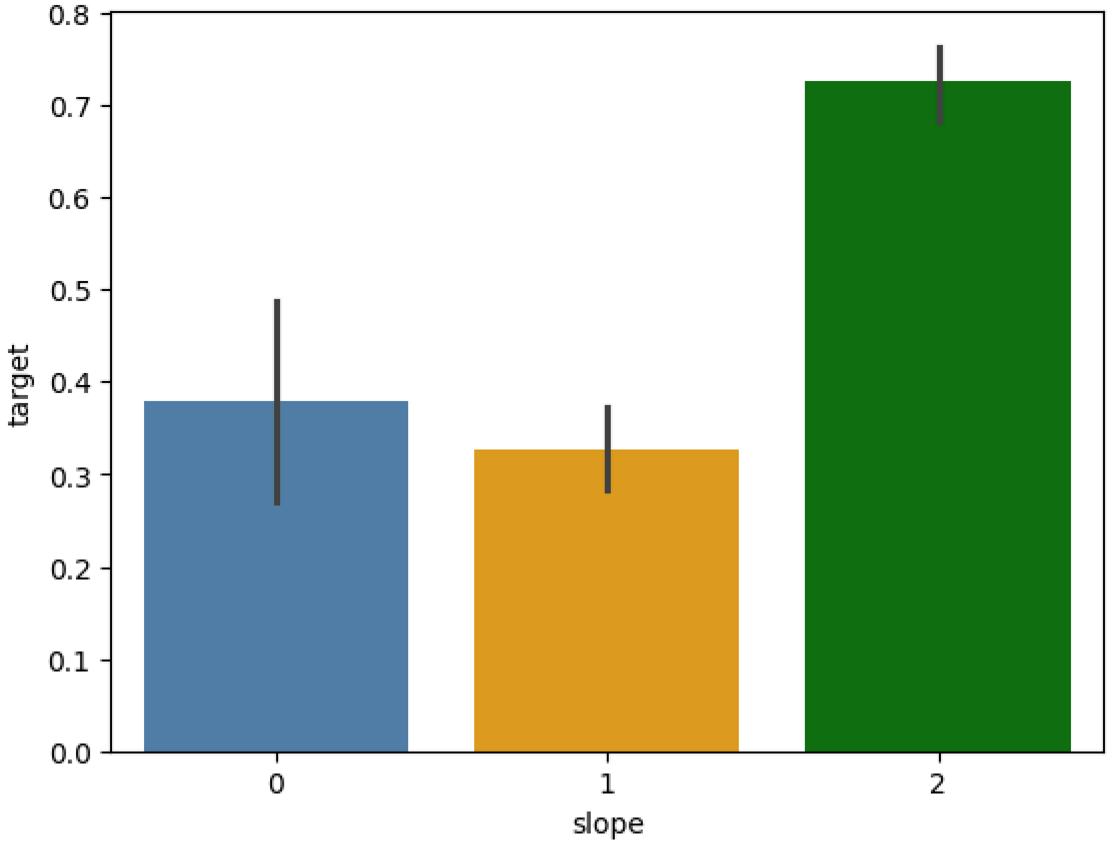
Analysing the Slope feature

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

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

```
sns.barplot(x="slope", y="target", data=dataset, palette=["steelblue", "orange", "green"]
```

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



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

Analysing the 'ca' feature

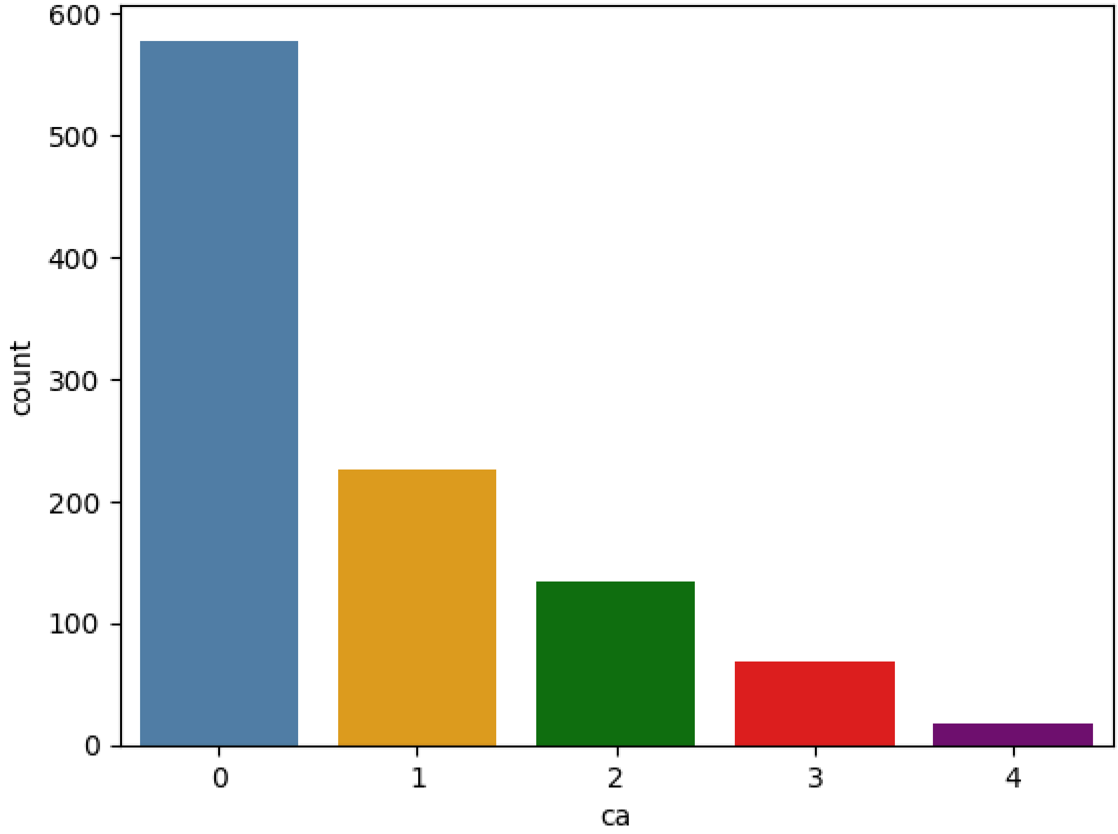
```
#number of major vessels (0-3) colored by flourosopy
```

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

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

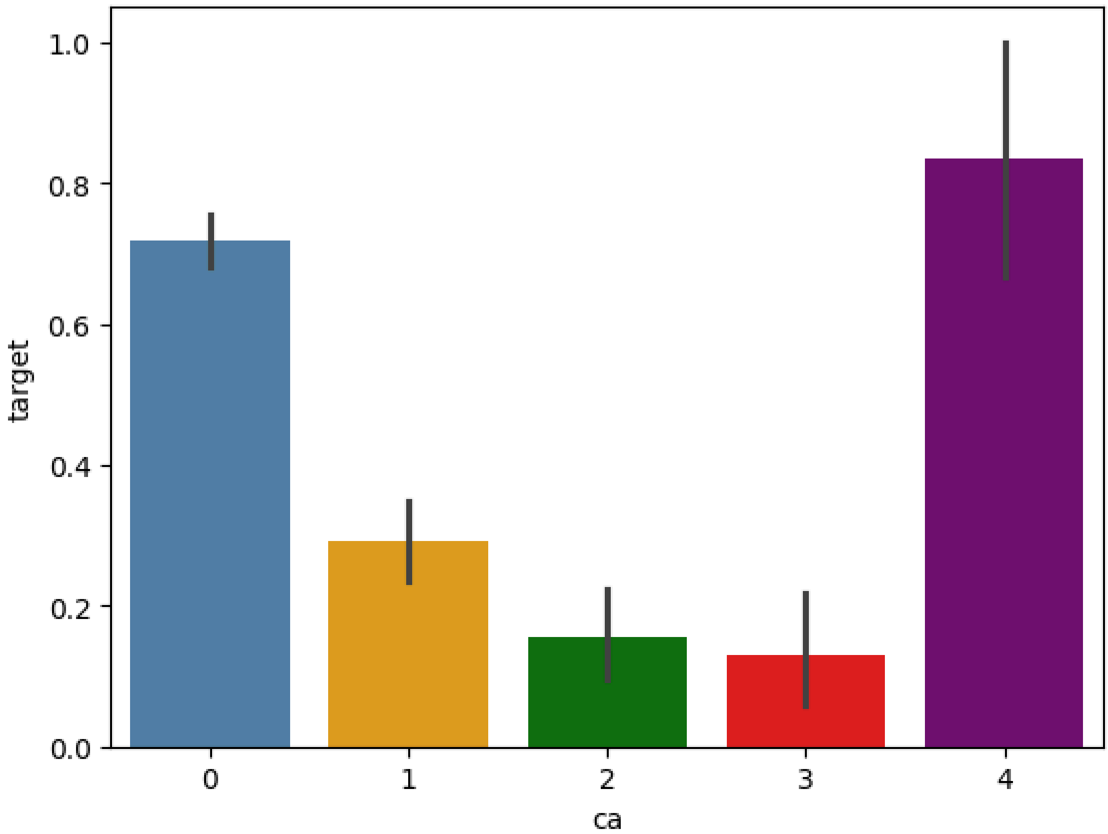
```
sns.countplot(x="ca", data=dataset, palette=["steelblue", "orange", "green", "red", "purp
```

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




```
sns.barplot(x="ca", y="target", data=dataset, palette=["steelblue", "orange", "green", "r
```

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



ca=4 has astonishingly large number of heart patients

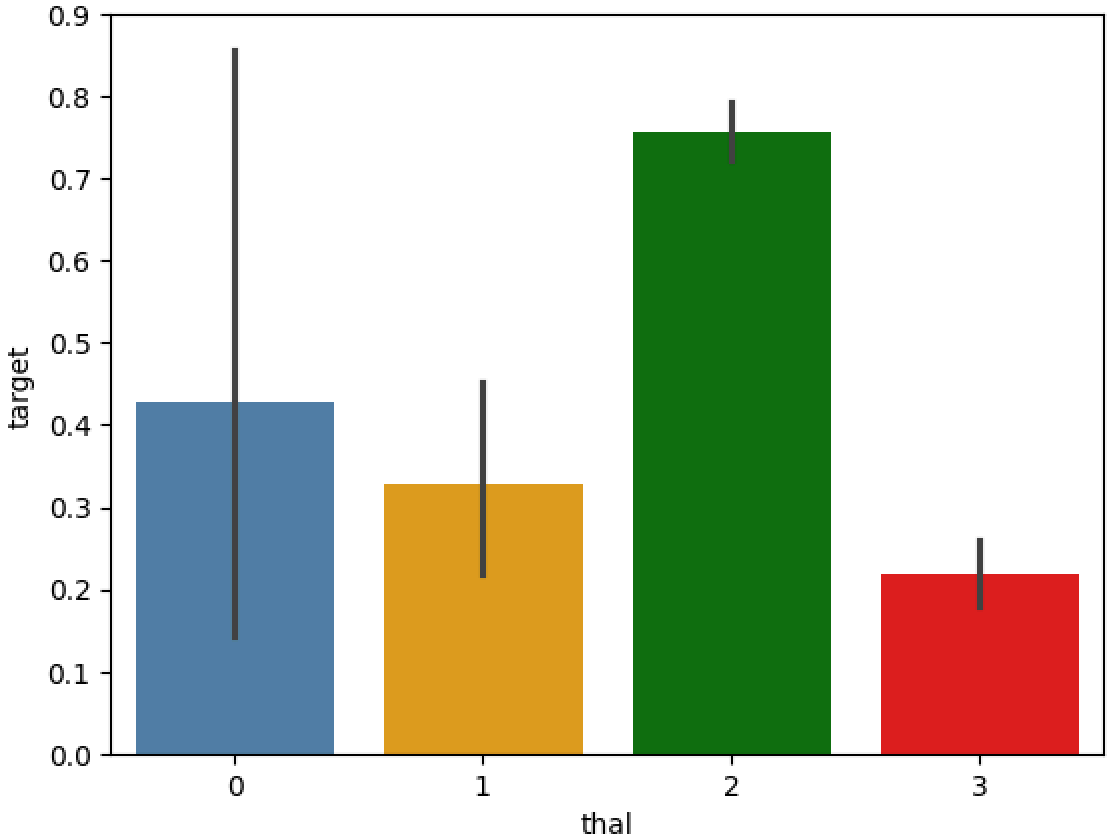
```
### Analysing the 'thal' feature
```

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

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

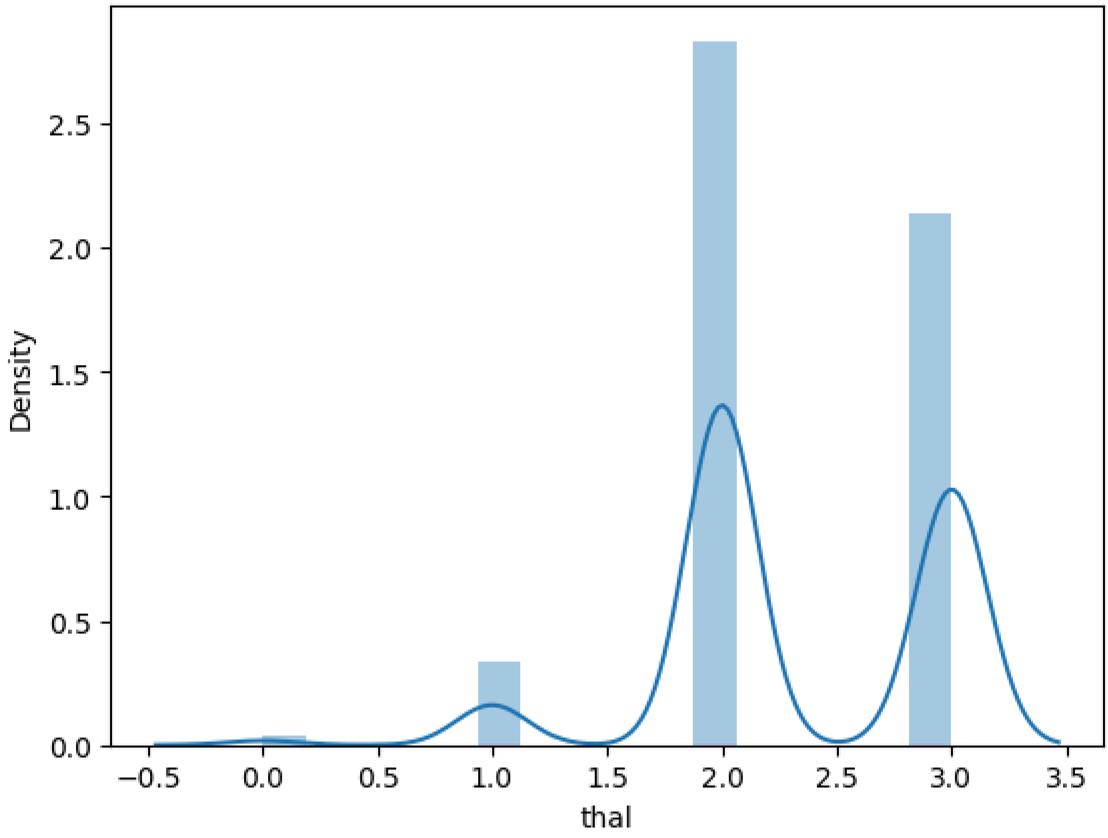
```
sns.barplot(x="thal", y="target", data=dataset, palette=["steelblue", "orange", "green",
```

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




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

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



IV. 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_
```

```
X_train.shape
```

```
(820, 13)
```

```
X_test.shape
```

```
(205, 13)
```

```
Y_train.shape
```

```
(820,)
```

```
Y_test.shape
```

```
(205,)
```

V. Model Fitting

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

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
```

```
(205,)
```

```
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: 86.34 %
```

Naive 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
```

```
(205,)
```

```
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: 85.37 %
```

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
```

```
(205,)
```

```
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: 83.9 %
```

K Nearest Neighbors

```
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
```

```
(205,)
```

```
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: 72.2 %
```

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)
```

```
(205,)
```

```
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: 100.0 %
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0


for x in range(1050):
    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
```

```
(205,)
```

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

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

```
The accuracy score achieved using Random Forest is: 100.0 %
```

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
```

```
(205,)
```

```
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: 100.0 %
```

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=1050)
```

```
Epoch 1/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.5345 - loss: 120.7349
Epoch 2/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5200 - loss: 98.6464
Epoch 3/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5324 - loss: 72.5548
Epoch 4/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4776 - loss: 57.6612
Epoch 5/1050
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5142 - loss: 33.5769
Epoch 6/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5233 - loss: 15.0681
Epoch 7/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6316 - loss: 3.5013
Epoch 8/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6387 - loss: 3.4180
Epoch 9/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6496 - loss: 3.0750
Epoch 10/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6398 - loss: 2.9235
Epoch 11/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6376 - loss: 3.0607
Epoch 12/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6438 - loss: 2.8937
Epoch 13/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6710 - loss: 2.6831
Epoch 14/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6660 - loss: 2.4999
Epoch 15/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6916 - loss: 2.4233
Epoch 16/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6562 - loss: 2.4815
Epoch 17/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6711 - loss: 2.2748
Epoch 18/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6422 - loss: 2.3381
Epoch 19/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6866 - loss: 2.1865
Epoch 20/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6935 - loss: 2.0112
Epoch 21/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6865 - loss: 2.0908
Epoch 22/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6878 - loss: 1.9713
Epoch 23/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6808 - loss: 1.8509
Epoch 24/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6818 - loss: 1.8981
Epoch 25/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6860 - loss: 1.8239
Epoch 26/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7208 - loss: 1.5875
Epoch 27/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6636 - loss: 1.8953
Epoch 28/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6718 - loss: 1.7511
Epoch 29/1050
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7283 - loss: 1.3876
```

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

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
```

```
Y_pred_nn.shape
```

```
(205, 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 numbe
```

```
The accuracy score achieved using Neural Network is: 85.85 %
```

VI. Output final score

```
scores = [score_lr,score_nb,score_svm,score_knn,score_dt,score_rf,score_xgb,score_nn]
algorithms = ["Logistic Regression","Naive Bayes","Support Vector Machine","K-Nearest Nei

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

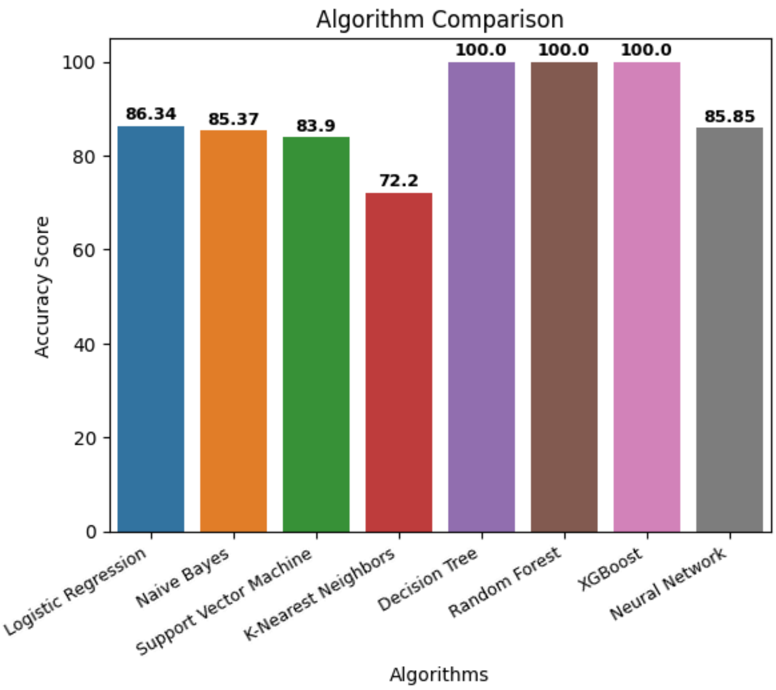
```
The accuracy score achieved using Logistic Regression is: 86.34 %
The accuracy score achieved using Naive Bayes is: 85.37 %
The accuracy score achieved using Support Vector Machine is: 83.9 %
The accuracy score achieved using K-Nearest Neighbors is: 72.2 %
The accuracy score achieved using Decision Tree is: 100.0 %
The accuracy score achieved using Random Forest is: 100.0 %
The accuracy score achieved using XGBoost is: 100.0 %
The accuracy score achieved using Neural Network is: 85.85 %
```

```
# Convert to DataFrame
results = pd.DataFrame({"Algorithm": algorithms, "Score": scores})

# Create barplot with 8 unique colors
sns.barplot(x="Algorithm", y="Score", data=results, palette="tab10")

# Add labels above bars
for i, score in enumerate(scores):
    plt.text(i, score + 0.5, str(round(score, 2)),
             ha='center', va='bottom', fontsize=9, fontweight='bold')

# Improve x-axis labels
plt.xticks(rotation=30, ha="right", fontsize=9)  # rotate so full names are visible
plt.xlabel("Algorithms")
plt.ylabel("Accuracy Score")
plt.title("Algorithm Comparison")
plt.ylim(0, max(scores) + 5)  # leave space for labels
plt.show()
```



Hey Abhishek!! Here Decision Tree, Random Rorest & XGBoost has good result as compare to other algorithms with the accuracy of 100%