HEART DISEASE PREDICTION BY NIKITA

I. IMPORTING THE ESSENTIAL LIBRARIES

In [61]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


%matplotlib inline



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

II. IMPORTING AND UNDERSTANDING OUR DATASET

In [2]:
```
df = pd.read_excel("1645792390_cep1_dataset.xlsx")
```

Shape of Dataset

In [3]:
```
df.shape
```

Out[3]:
```
(303, 14)
```

Printing few Columns

In [4]:
```
df.head()
```

Out[4]:

	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

Description

In [5]:
```
df.describe()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [7]:
```
info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic","resting blood pressure"," se



for i in range(len(info)):
    print(df.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: non-anginal pain, 4: asymptomatic
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 to rest
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

In [8]:
```
df["target"].describe()
```

Out[8]:
```
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
```

In [9]:
```
df["target"].unique()
```

Out[9]:
```
array([1, 0], dtype=int64)
```

Checking Corelation between columns

In [10]:
```
print(df.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
```

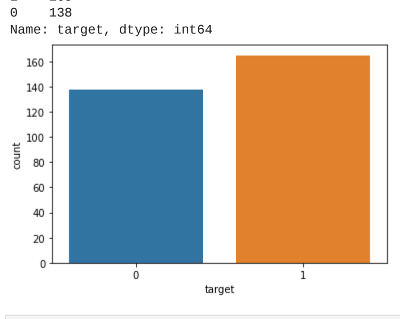
In [11]:
```
#'fbs' is weakely corelated
```

Eploratory Data Analytics (EDA)

Analysing the target Variable

In [12]:
```
y = df["target"]

sns.countplot(y)


target_temp = df.target.value_counts()

print(target_temp)
```

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



In [13]:
```
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)))
```

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

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

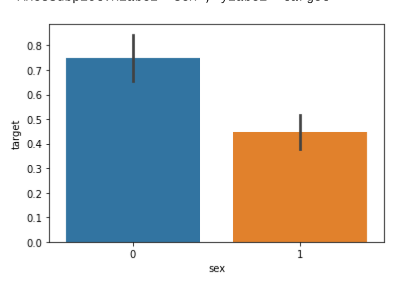
Analysing the 'Sex' feature

In [14]:
```
df["sex"].unique()
```

Out[14]:
```
array([1, 0], dtype=int64)
```

In [15]:
```
sns.barplot(df["sex"],y)
```

Out[15]:
```
<AxesSubplot:xlabel='sex', ylabel='target'>
```



We notice that Female have more chances of getting heart problems

Analysing the chest pain feature*

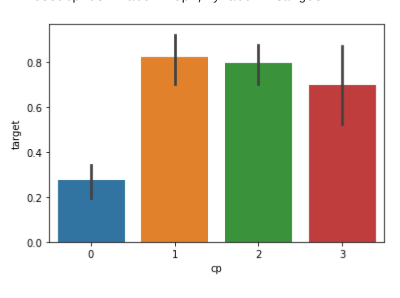
In [16]:
```
df["cp"].unique()
```

Out[16]:
```
array([3, 2, 1, 0], dtype=int64)
```

We notice here cp have values between o to 3

In [17]:
```
sns.barplot(df["cp"],y)
```

Out[17]:
```
<AxesSubplot: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

In [18]:
```
df["fbs"].describe()
```

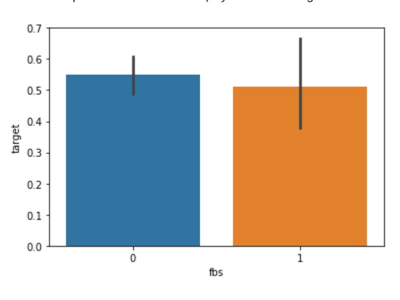
Out[18]:
```
count    303.000000
mean       0.148515
std        0.356198
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: fbs, dtype: float64
```

In [19]:
```
df["fbs"].unique()
```

Out[19]:
```
array([1, 0], dtype=int64)
```

In [20]:
```
sns.barplot(df["fbs"],y)
```

Out[20]:
```
<AxesSubplot:xlabel='fbs', ylabel='target'>
```



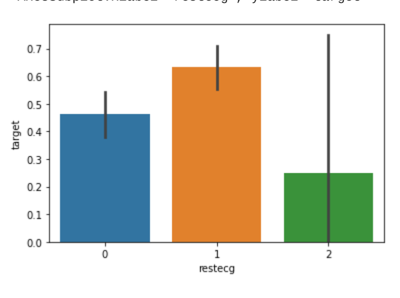
Analysing the restecg feature

In [21]:
```
df["restecg"].unique()
```

Out[21]:
```
array([0, 1, 2], dtype=int64)
```

In [22]:
```
sns.barplot(df["restecg"],y)
```

Out[22]:
```
<AxesSubplot: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'

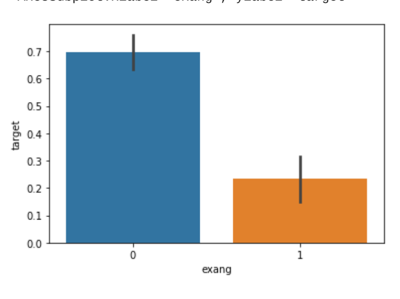
Analysing the exang feature

In [23]:
```
df["exang"].unique()
```

Out[23]:
```
array([0, 1], dtype=int64)
```

In [24]:
```
sns.barplot(df["exang"],y)
```

Out[24]:
```
<AxesSubplot:xlabel='exang', ylabel='target'>
```



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

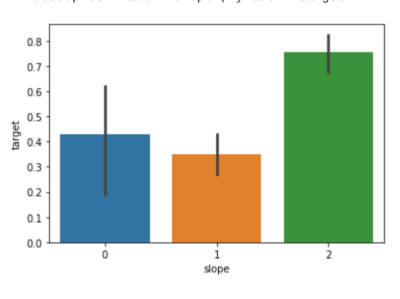
Analysing slope feature

In [25]:
```
df["slope"].unique()
```

Out[25]:
```
array([0, 2, 1], dtype=int64)
```

In [27]:
```
sns.barplot(df["slope"],y)
```

Out[27]:
```
<AxesSubplot:xlabel='slope', ylabel='target'>
```



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

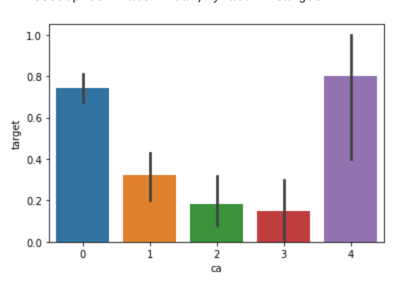
Analysing the 'ca' feature

In [28]:
```
df["ca"].unique()
```

Out[28]:
```
array([0, 2, 1, 3, 4], dtype=int64)
```

In [29]:
```
sns.barplot(df["ca"],y)
```

Out[29]:
```
<AxesSubplot:xlabel='ca', ylabel='target'>
```



ca=4 has astonishingly large number of heart patients

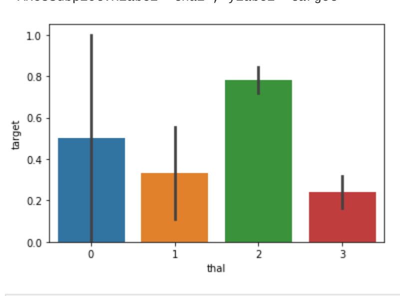
Analysing the 'thal' feature

In [30]:
```
df["thal"].unique()
```

Out[30]:
```
array([1, 2, 3, 0], dtype=int64)
```

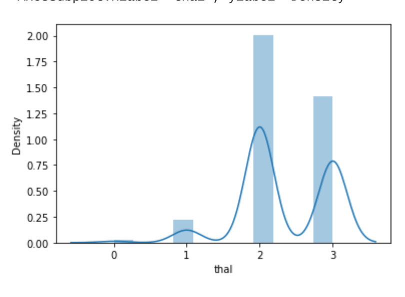
In [31]:
```
sns.barplot(df["thal"],y)
```

Out[31]:
```
<AxesSubplot:xlabel='thal', ylabel='target'>
```



In [32]:
```
sns.distplot(df["thal"])
```

Out[32]:
```
<AxesSubplot:xlabel='thal', ylabel='Density'>
```



IV. TRAIN TEST SPLIT

In [33]:
```
from sklearn.model_selection import train_test_split
predictors = df.drop("target" , axis=1)
target = df["target"]
X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.20,random_state=0)
```

In [35]:
```
X_train.shape
```

Out[35]:
```
(242, 13)
```

In [36]:
```
X_test.shape
```

Out[36]:
```
(61, 13)
```

In [37]:
```
Y_train.shape
```

Out[37]:
```
(242,)
```

In [38]:
```
Y_test.shape
```

Out[38]:
```
(61,)
```

V. MODEL FITTING

In [40]:
```
from sklearn.metrics import accuracy_score
```

In [41]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train,Y_train)

Y_pred_lr = lr.predict(X_test)
```

In [42]:
```
Y_pred_lr.shape
```

Out[42]:
```
(61,)
```

In [43]:
```
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 %
```

(A). Naive Bayes

In [44]:
```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
```

In [45]:
```
Y_pred_nb.shape
```

Out[45]:
```
(61,)
```

In [46]:
```
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 %
```

(B). SVM

In [47]:
```
from sklearn import svm

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

sv.fit(X_train, Y_train)

Y_pred_svm = sv.predict(X_test)
```

In [48]:
```
Y_pred_svm.shape
```

Out[48]:
```
(61,)
```

In [49]:
```
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 %
```

(C). K Nearest Neighbours

In [50]:
```
from sklearn.neighbors import KNeighborsClassifier

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

In [51]:
```
Y_pred_knn.shape
```

Out[51]:
```
(61,)
```

In [52]:
```
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 %
```

(D). Decision Tree

In [53]:
```
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


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

In [54]:
```
print(Y_pred_dt.shape)
```

```
(61,)
```

In [55]:
```
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 %
```

(E). Random Forest

In [56]:
```
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0


for x in range(2000):
    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)
```

In [57]:
```
Y_pred_rf.shape
```

Out[57]:
```
(61,)
```

In [58]:
```
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: 90.16 %
```

XGBoost

In [62]:
```
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)
```

```
[11:52:22] WARNING: D:\bld\xgboost-split_1645118015404\work\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the object
ive 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [63]:
```
Y_pred_xgb.shape
```

Out[63]:
```
(61,)
```

In [64]:
```
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

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

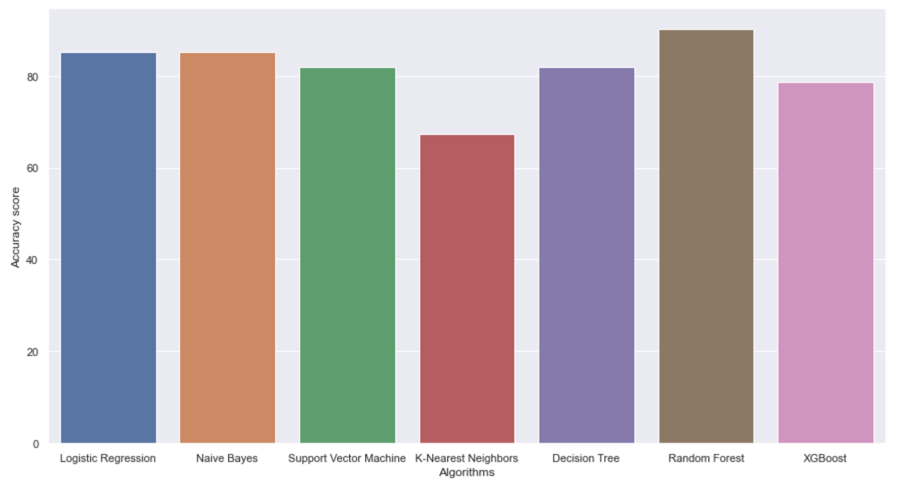
```
The accuracy score achieved using XGBoost is: 78.69 %
```

In [71]:
```
scores = [score_lr,score_nb,score_svm,score_knn,score_dt,score_rf,score_xgb]
algorithms = ["Logistic Regression","Naive Bayes","Support Vector Machine","K-Nearest Neighbors","Decision Tree","Random Forest","XGBoost"]

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 Support Vector Machine 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: 90.16 %
The accuracy score achieved using XGBoost is: 78.69 %
```

In [72]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms,scores)
```

Out[72]:
```
<AxesSubplot:xlabel='Algorithms', ylabel='Accuracy score'>
```



Here, Random Forest is giving Best accuracy

In [ ]: