Heart Disease Prediction By Kshitij Agarwal, Kushagra Mishra, Kumar Ayush Sinha and Shivam Shreyans

I. Importing essential libraries

In [1]:
```
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')
```

```
['.ipynb_checkpoints', 'heart.csv', 'Heart_dise
ase_prediction.ipynb', 'Heart_disease_predictio
n.py', 'README.md']
```

II. Importing and understanding our dataset

In [2]:
```
dataset = pd.read_csv("heart.csv")
```

Verifying it as a 'dataframe' object in pandas

In [3]:
```
type(dataset)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

Shape of dataset

In [4]:
```
dataset.shape
```

Out[4]:
```
(303, 14)
```

Printing out a few columns

In [5]:
```
dataset.head(5)
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
0	63	1	3	145	233	1	0	150	0	2.3	0	0
1	37	1	2	130	250	0	1	187	0	3.5	0	0
2	41	0	1	130	204	0	0	172	0	1.4	2	0
3	56	1	1	120	236	0	1	178	0	0.8	2	0
4	57	0	0	120	354	0	1	163	1	0.6	2	0

In [6]:
```
dataset.sample(5)
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
91	57	1	0	132	207	0	1	168	1	0.0	2	0
11	48	0	2	130	275	0	1	139	0	0.2	2	0
219	48	1	0	130	256	1	0	150	1	0.0	2	2
106	69	1	3	160	234	1	0	131	0	0.1	1	1
95	53	1	0	142	226	0	0	111	1	0.0	2	0

Description

In [7]:
```
dataset.describe()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.

In [8]:
```
dataset.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 [9]:
```
###Luckily, we have no missing values
```

Let's understand our columns better:

In [10]:
```
info = ["age","1: male, 0: female","chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic","resting blood pressure"," serum cholestoral in mg/dl","fasting blood sugar > 120 mg/dl","resting electrocardiographic results (values 0,1,2)"," maximum heart rate achieved","exercise induced angina","oldpeak = ST depression induced by exercise relative to rest","the slope of the peak exercise ST segment","number of major vessels (0-3) colored by flourosopy","thal: 3 = normal; 6 = fixed defect; 7 = reversable defect"]



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: typ
ical angina, 2: atypical angina, 3: non-anginal
pain, 4: asymptomatic
trestbps:                       resting blood p
ressure
chol:                    serum cholestoral in m
g/dl
fbs:                    fasting blood sugar > 1
20 mg/dl
restecg:                        resting electro
cardiographic results (values 0,1,2)
thalach:                         maximum heart
rate achieved
exang:                  exercise induced angina
oldpeak:                        oldpeak = ST de
pression induced by exercise relative to rest
slope:                  the slope of the peak e
xercise ST segment
ca:                     number of major vessels
(0-3) colored by flourosopy
thal:                   thal: 3 = normal; 6 = f
ixed defect; 7 = reversable defect
```

Analysing the 'target' variable

In [11]:
```
dataset["target"].describe()
```

Out[11]:
```
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 [12]:
```
dataset["target"].unique()
```

Out[12]:
```
array([1, 0], dtype=int64)
```

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

Checking correlation between columns

In [13]:
```
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
```

In [14]:
```
#This shows that most columns are moderately correlated with target, but 'fbs' is very weakly correlated.
```

Exploratory Data Analysis (EDA)

First, analysing the target variable:

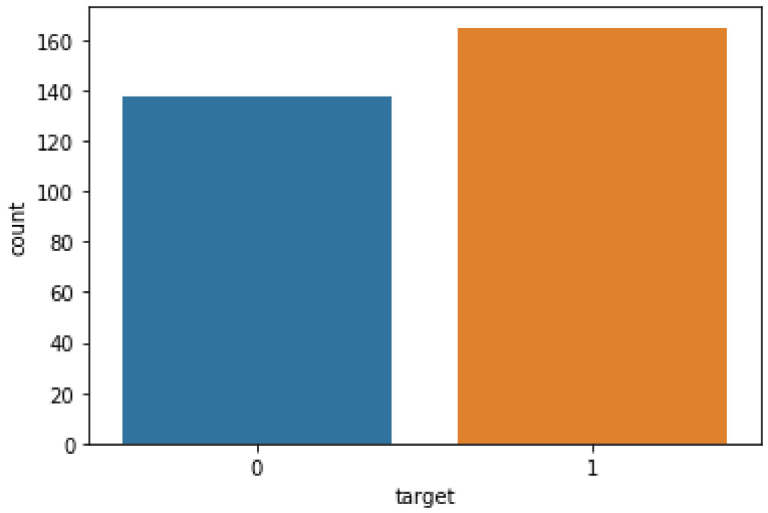
In [15]:
```
y = dataset["target"]

sns.countplot(y)


target_temp = dataset.target.value_counts()

print(target_temp)
```

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



In [16]:
```
print("Percentage of patience without heart proble
ms: "+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 problem
s: "+str(y.where(y==1).count()*100/303))
# print("Percentage of patience with heart problem
s: "+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
```

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

Analysing the 'Sex' feature

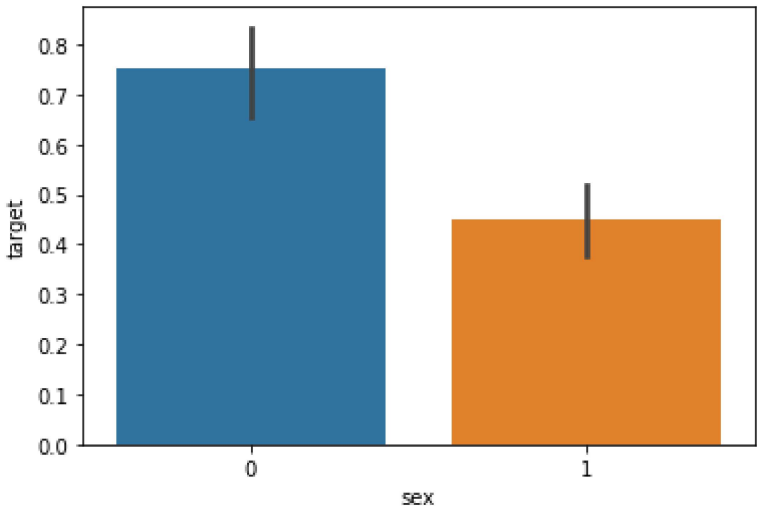
In [17]:
```
dataset["sex"].unique()
```

Out[17]:
```
array([1, 0], dtype=int64)
```

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

In [18]:
```
sns.barplot(dataset["sex"],y)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d9074b708>
```



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

Analysing the 'Chest Pain Type' feature

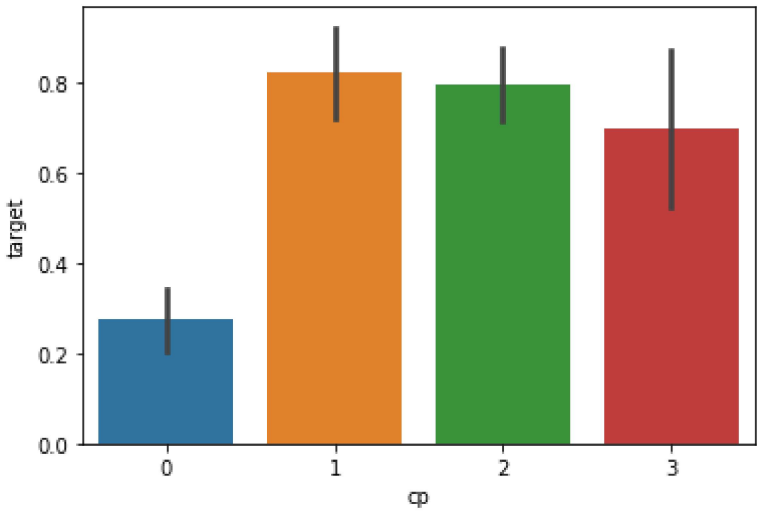
In [19]:
```
dataset["cp"].unique()
```

Out[19]:
```
array([3, 2, 1, 0], dtype=int64)
```

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

In [20]:
```
sns.barplot(dataset["cp"],y)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d907b88c8>
```



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 [21]:
```
dataset["fbs"].describe()
```

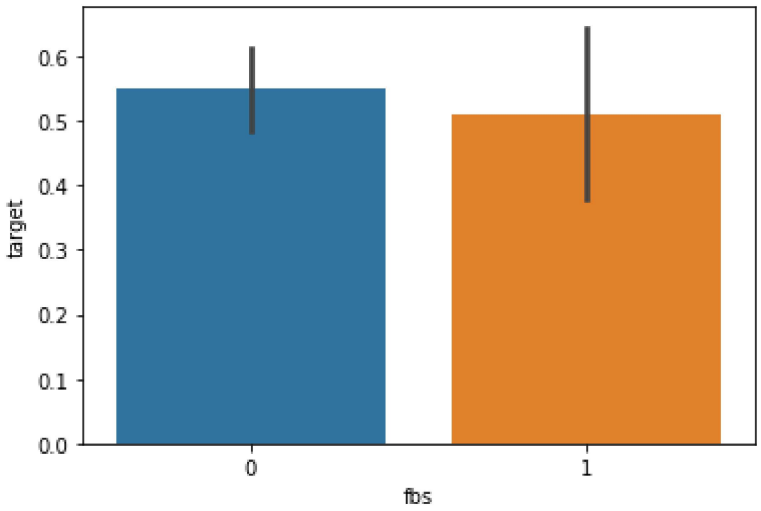
Out[21]:
```
count    303.000000
mean       0.148515
std        0.356198
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: fbs, dtype: float64
```

In [22]:
```
dataset["fbs"].unique()
```

Out[22]:
```
array([1, 0], dtype=int64)
```

In [23]:
```
sns.barplot(dataset["fbs"],y)
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d90827788>
```



Nothing extraordinary here

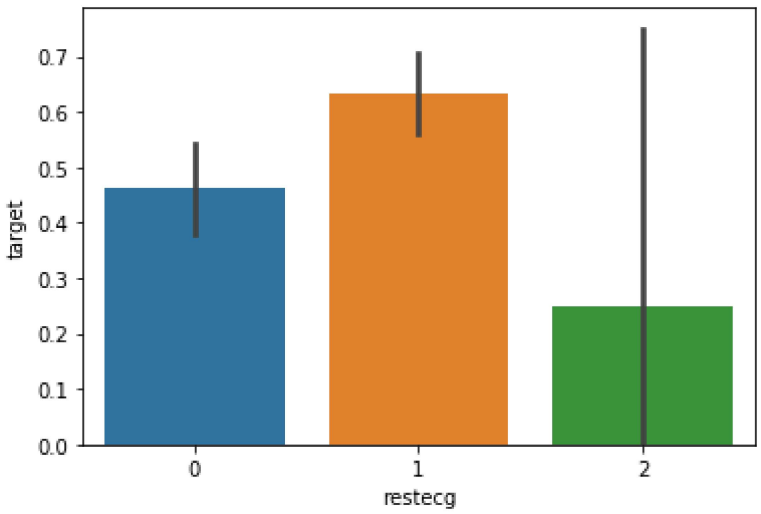
Analysing the restecg feature

In [24]:
```
dataset["restecg"].unique()
```

Out[24]:
```
array([0, 1, 2], dtype=int64)
```

In [25]:
```
sns.barplot(dataset["restecg"],y)
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d90884988>
```



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

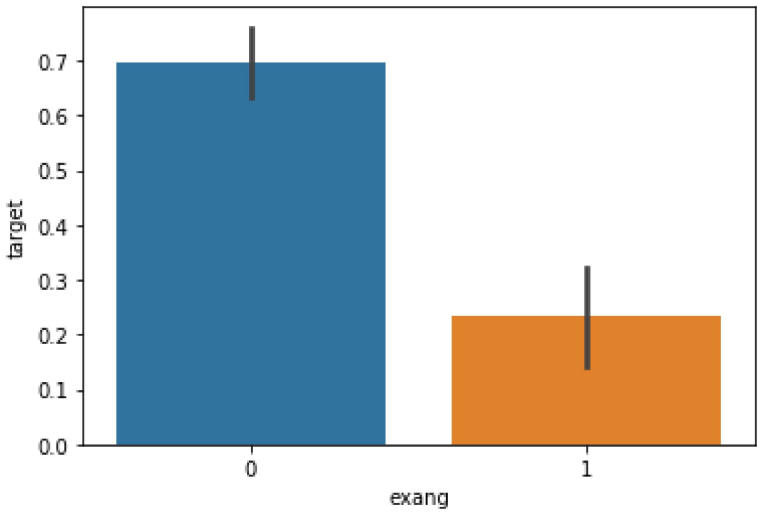
Analysing the 'exang' feature

In [26]:
```
dataset["exang"].unique()
```

Out[26]:
```
array([0, 1], dtype=int64)
```

In [27]:
```
sns.barplot(dataset["exang"],y)
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d908f90c8>
```



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

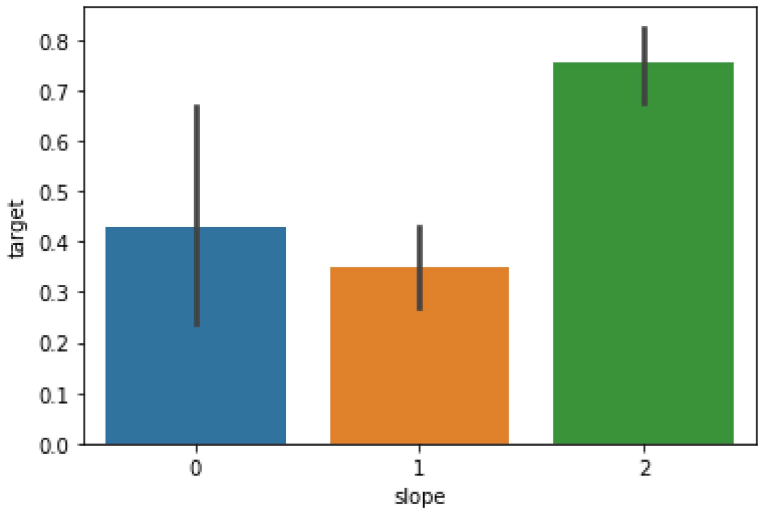
Analysing the Slope feature

In [28]:
```
dataset["slope"].unique()
```

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

In [29]:
```
sns.barplot(dataset["slope"],y)
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d9088c708>
```



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

Analysing the 'ca' feature

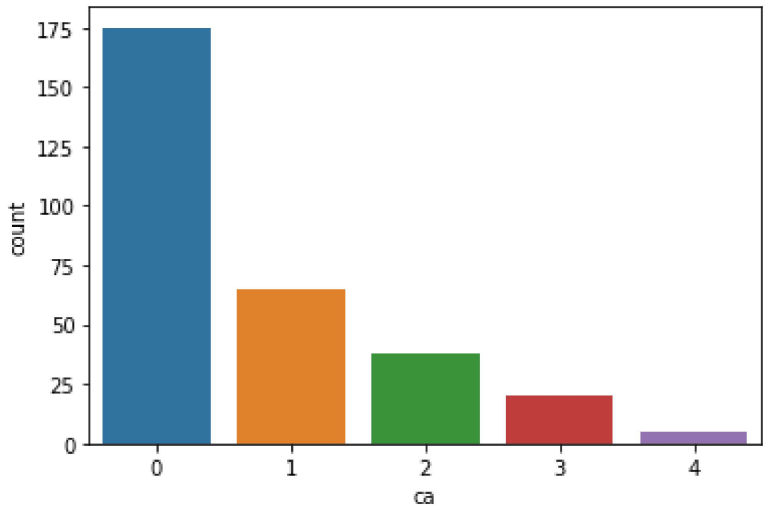
In [30]:
```
#number of major vessels (0-3) colored by flouroso
py
```

In [31]:
```
dataset["ca"].unique()
```

Out[31]:
```
array([0, 2, 1, 3, 4], dtype=int64)
```

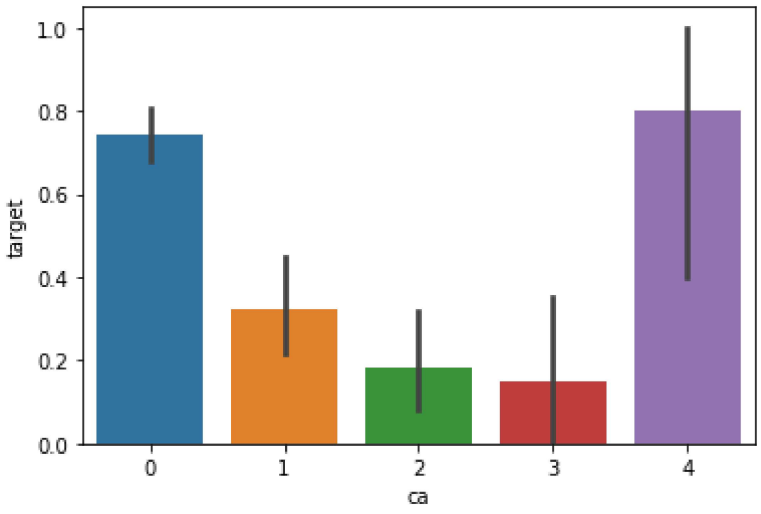
In [32]:
```
sns.countplot(dataset["ca"])
```

Out[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d909ced88>
```



In [33]:
```
sns.barplot(dataset["ca"],y)
```

Out[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d90a29548>
```



ca=4 has astonishingly large number of heart patients

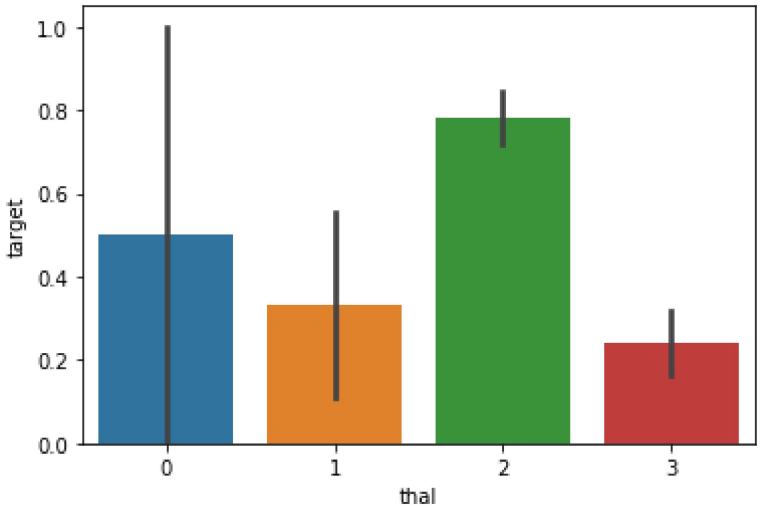
In [34]:
```
### Analysing the 'thal' feature
```

In [35]:
```
dataset["thal"].unique()
```

Out[35]:
```
array([1, 2, 3, 0], dtype=int64)
```

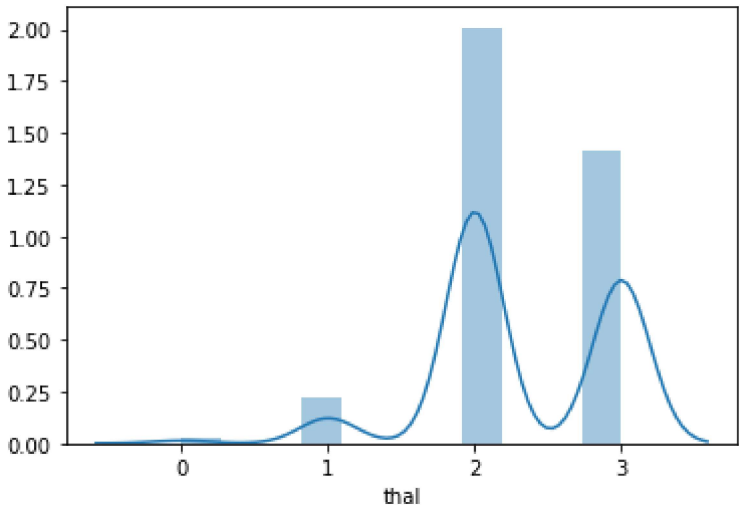
In [36]:
```
sns.barplot(dataset["thal"],y)
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d90ab9748>
```



In [37]:
```
sns.distplot(dataset["thal"])
```

Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22d90a2e5c8>
```



IV. Train Test split

In [38]:
```
from sklearn.model_selection import train_test_spl
it

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

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

In [39]:
```
X_train.shape
```

Out[39]:
```
(242, 13)
```

In [40]:
```
X_test.shape
```

Out[40]:
```
(61, 13)
```

In [41]:
```
Y_train.shape
```

Out[41]:
```
(242,)
```

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

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

V. Model Fitting

In [43]:
```
from sklearn.metrics import accuracy_score
```

Logistic Regression

In [44]:
```
from sklearn.linear_model import LogisticRegressio
n

lr = LogisticRegression()

lr.fit(X_train,Y_train)

Y_pred_lr = lr.predict(X_test)
```

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

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

In [46]:
```
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 Regr
ession is: 85.25 %
```

Naive Bayes

In [47]:
```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
```

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

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

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

SVM

In [50]:
```
from sklearn import svm

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

sv.fit(X_train, Y_train)

Y_pred_svm = sv.predict(X_test)
```

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

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

In [52]:
```
score_svm = round(accuracy_score(Y_pred_svm,Y_test
)*100,2)

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

```
The accuracy score achieved using Linear SVM i
s: 81.97 %
```

K Nearest Neighbors

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier

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

In [54]:
```
Y_pred_knn.shape
```

Out[54]:
```
(61,)
```

In [55]:
```
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

In [56]:
```
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)
```

In [57]:
```
print(Y_pred_dt.shape)
```

```
(61,)
```

In [58]:
```
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

In [59]:
```
from sklearn.ensemble import RandomForestClassifie
r

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 [60]:
```
Y_pred_rf.shape
```

Out[60]:
```
(61,)
```

In [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: 90.16 %
```

XGBoost

In [62]:
```
import xgboost as xgb

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

Y_pred_xgb = xgb_model.predict(X_test)
```

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 i
s: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 8
5.25 %
```

Neural Network

In [65]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

In [66]:
```
# 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'])
```

In [ ]:
```
model.fit(X_train,Y_train,epochs=300)
```

```
Epoch 1/300
242/242 [==============================] - 1s 5
ms/step - loss: 20.9709 - accuracy: 0.5413
Epoch 2/300
242/242 [==============================] - 0s 2
1us/step - loss: 14.3212 - accuracy: 0.5413
Epoch 3/300
242/242 [==============================] - 0s 6
5us/step - loss: 7.3186 - accuracy: 0.5413
Epoch 4/300
242/242 [==============================] - 0s 0
us/step - loss: 1.8330 - accuracy: 0.5455
Epoch 5/300
242/242 [==============================] - 0s 6
5us/step - loss: 2.4037 - accuracy: 0.4545
Epoch 6/300
242/242 [==============================] - 0s 6
5us/step - loss: 1.0642 - accuracy: 0.5992
Epoch 7/300
242/242 [==============================] - 0s 0
us/step - loss: 1.1605 - accuracy: 0.5826
Epoch 8/300
242/242 [==============================] - 0s 0
us/step - loss: 0.9678 - accuracy: 0.5372
Epoch 9/300
242/242 [==============================] - 0s 7
1us/step - loss: 0.8868 - accuracy: 0.6074
Epoch 10/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.8260 - accuracy: 0.6446
Epoch 11/300
242/242 [==============================] - 0s 0
us/step - loss: 0.8217 - accuracy: 0.5909
```

```
Epoch 12/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.7991 - accuracy: 0.6653
Epoch 13/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.7801 - accuracy: 0.6570
Epoch 14/300
242/242 [==============================] - 0s 0
us/step - loss: 0.7633 - accuracy: 0.6570
Epoch 15/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.7459 - accuracy: 0.6901
Epoch 16/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.7364 - accuracy: 0.6942
Epoch 17/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.7169 - accuracy: 0.7025
Epoch 18/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.7012 - accuracy: 0.7025
Epoch 19/300
242/242 [==============================] - ETA:
0s - loss: 0.7761 - accuracy: 0.62 - 0s 0us/ste
p - loss: 0.6942 - accuracy: 0.6942
Epoch 20/300
242/242 [==============================] - 0s 0
us/step - loss: 0.6771 - accuracy: 0.7231
Epoch 21/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.6751 - accuracy: 0.7107
Epoch 22/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.6690 - accuracy: 0.7231
Epoch 23/300
242/242 [==============================] - 0s 0
us/step - loss: 0.6575 - accuracy: 0.7149
Epoch 24/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.6690 - accuracy: 0.7066
Epoch 25/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.6558 - accuracy: 0.7025
Epoch 26/300
242/242 [==============================] - 0s 0
us/step - loss: 0.6425 - accuracy: 0.7438
Epoch 27/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.6300 - accuracy: 0.7149
Epoch 28/300
242/242 [==============================] - 0s 0
us/step - loss: 0.6237 - accuracy: 0.7438
Epoch 29/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5998 - accuracy: 0.7438
Epoch 30/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5907 - accuracy: 0.7645
Epoch 31/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5835 - accuracy: 0.7603
Epoch 32/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5797 - accuracy: 0.7397
Epoch 33/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.5721 - accuracy: 0.7727
Epoch 34/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5723 - accuracy: 0.7686
Epoch 35/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5702 - accuracy: 0.7562
Epoch 36/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.5557 - accuracy: 0.7769
Epoch 37/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5559 - accuracy: 0.7438
Epoch 38/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5500 - accuracy: 0.7645
Epoch 39/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5433 - accuracy: 0.7603
Epoch 40/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5418 - accuracy: 0.7727
Epoch 41/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5352 - accuracy: 0.7603
Epoch 42/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5301 - accuracy: 0.7686
Epoch 43/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5369 - accuracy: 0.7645
Epoch 44/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.5237 - accuracy: 0.7603
Epoch 45/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.5217 - accuracy: 0.7645
Epoch 46/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5230 - accuracy: 0.7686
Epoch 47/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5156 - accuracy: 0.7686
Epoch 48/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5142 - accuracy: 0.7686
Epoch 49/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5075 - accuracy: 0.7769
Epoch 50/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5017 - accuracy: 0.7934
Epoch 51/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.5099 - accuracy: 0.7810
Epoch 52/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4997 - accuracy: 0.7893
Epoch 53/300
242/242 [==============================] - 0s 0
us/step - loss: 0.5068 - accuracy: 0.7810
Epoch 54/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.4941 - accuracy: 0.7851
Epoch 55/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.4922 - accuracy: 0.7810
Epoch 56/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4866 - accuracy: 0.7810
Epoch 57/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4811 - accuracy: 0.8099
Epoch 58/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4843 - accuracy: 0.8058
Epoch 59/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4809 - accuracy: 0.7934
Epoch 60/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4905 - accuracy: 0.8058
Epoch 61/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.4859 - accuracy: 0.7851
Epoch 62/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4896 - accuracy: 0.7893
Epoch 63/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4800 - accuracy: 0.7934
Epoch 64/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4606 - accuracy: 0.8099
Epoch 65/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4746 - accuracy: 0.8058
Epoch 66/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.4934 - accuracy: 0.7851
Epoch 67/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4890 - accuracy: 0.7934
Epoch 68/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4410 - accuracy: 0.8140
Epoch 69/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4908 - accuracy: 0.8017
Epoch 70/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4449 - accuracy: 0.8140
Epoch 71/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4469 - accuracy: 0.8099
Epoch 72/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4646 - accuracy: 0.8017
Epoch 73/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4568 - accuracy: 0.8099
Epoch 74/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4321 - accuracy: 0.8264
Epoch 75/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4374 - accuracy: 0.8223
Epoch 76/300
242/242 [==============================] - 0s 4
5us/step - loss: 0.4282 - accuracy: 0.8140
Epoch 77/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.4354 - accuracy: 0.8099
Epoch 78/300
242/242 [==============================] - 0s 4
5us/step - loss: 0.4374 - accuracy: 0.8306
Epoch 79/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4287 - accuracy: 0.8223
Epoch 80/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4292 - accuracy: 0.8182
Epoch 81/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4239 - accuracy: 0.8554
Epoch 82/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4224 - accuracy: 0.8058
Epoch 83/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4215 - accuracy: 0.8017
Epoch 84/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4506 - accuracy: 0.8140
Epoch 85/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4406 - accuracy: 0.8140
Epoch 86/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4310 - accuracy: 0.8140
Epoch 87/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.4247 - accuracy: 0.8264
Epoch 88/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.4190 - accuracy: 0.8264
Epoch 89/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4142 - accuracy: 0.8264
Epoch 90/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4085 - accuracy: 0.8388
Epoch 91/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4085 - accuracy: 0.8223
Epoch 92/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4045 - accuracy: 0.8306
Epoch 93/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4085 - accuracy: 0.8223
Epoch 94/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4069 - accuracy: 0.8182
Epoch 95/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4165 - accuracy: 0.8182
Epoch 96/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3998 - accuracy: 0.8223
Epoch 97/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4248 - accuracy: 0.8264
Epoch 98/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3887 - accuracy: 0.8430
Epoch 99/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.4535 - accuracy: 0.7975
Epoch 100/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4546 - accuracy: 0.8058
Epoch 101/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4428 - accuracy: 0.8140
Epoch 102/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4027 - accuracy: 0.8306
Epoch 103/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4048 - accuracy: 0.8264
Epoch 104/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3910 - accuracy: 0.8306
Epoch 105/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3945 - accuracy: 0.8430
Epoch 106/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3919 - accuracy: 0.8347
Epoch 107/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3914 - accuracy: 0.8264
Epoch 108/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3847 - accuracy: 0.8264
Epoch 109/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3844 - accuracy: 0.8264
Epoch 110/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.3890 - accuracy: 0.8388
Epoch 111/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4059 - accuracy: 0.8017
Epoch 112/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4309 - accuracy: 0.8099
Epoch 113/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.3864 - accuracy: 0.8347
Epoch 114/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3725 - accuracy: 0.8347
Epoch 115/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3880 - accuracy: 0.8347
Epoch 116/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3827 - accuracy: 0.8264
Epoch 117/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3773 - accuracy: 0.8306
Epoch 118/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3878 - accuracy: 0.8430
Epoch 119/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3797 - accuracy: 0.8388
Epoch 120/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3687 - accuracy: 0.8388
Epoch 121/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.3921 - accuracy: 0.8430
Epoch 122/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3928 - accuracy: 0.8223
Epoch 123/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3785 - accuracy: 0.8347
Epoch 124/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3767 - accuracy: 0.8306
Epoch 125/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3745 - accuracy: 0.8347
Epoch 126/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3852 - accuracy: 0.8223
Epoch 127/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4162 - accuracy: 0.8264
Epoch 128/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3861 - accuracy: 0.8347
Epoch 129/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3853 - accuracy: 0.8347
Epoch 130/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3665 - accuracy: 0.8347
Epoch 131/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3694 - accuracy: 0.8306
Epoch 132/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.3655 - accuracy: 0.8471
Epoch 133/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3670 - accuracy: 0.8471
Epoch 134/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3876 - accuracy: 0.8182
Epoch 135/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3725 - accuracy: 0.8388
Epoch 136/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3710 - accuracy: 0.8430
Epoch 137/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3729 - accuracy: 0.8223
Epoch 138/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3711 - accuracy: 0.8471
Epoch 139/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3655 - accuracy: 0.8554
Epoch 140/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3721 - accuracy: 0.8471
Epoch 141/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3722 - accuracy: 0.8388
Epoch 142/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3722 - accuracy: 0.8636
Epoch 143/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.3712 - accuracy: 0.8388
Epoch 144/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3720 - accuracy: 0.8223
Epoch 145/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3888 - accuracy: 0.8140
Epoch 146/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3620 - accuracy: 0.8430
Epoch 147/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.3675 - accuracy: 0.8512
Epoch 148/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3618 - accuracy: 0.8388
Epoch 149/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3734 - accuracy: 0.8512
Epoch 150/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3738 - accuracy: 0.8388
Epoch 151/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3647 - accuracy: 0.8388
Epoch 152/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3641 - accuracy: 0.8554
Epoch 153/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3658 - accuracy: 0.8388
Epoch 154/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.3574 - accuracy: 0.8636
Epoch 155/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3682 - accuracy: 0.8471
Epoch 156/300
242/242 [==============================] - 0s 9
1us/step - loss: 0.3880 - accuracy: 0.8264
Epoch 157/300
242/242 [==============================] - 0s 0
us/step - loss: 0.4038 - accuracy: 0.8347
Epoch 158/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3619 - accuracy: 0.8595
Epoch 159/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3654 - accuracy: 0.8388
Epoch 160/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3572 - accuracy: 0.8471
Epoch 161/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3675 - accuracy: 0.8347
Epoch 162/300
242/242 [==============================] - 0s 9
2us/step - loss: 0.3694 - accuracy: 0.8471
Epoch 163/300
242/242 [==============================] - 0s 8
us/step - loss: 0.3673 - accuracy: 0.8388
Epoch 164/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3650 - accuracy: 0.8430
Epoch 165/300
242/242 [==============================] - 0s 4
```

```
5us/step - loss: 0.3646 - accuracy: 0.8554
Epoch 166/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3609 - accuracy: 0.8512
Epoch 167/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3898 - accuracy: 0.8347
Epoch 168/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.4237 - accuracy: 0.8182
Epoch 169/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3958 - accuracy: 0.8223
Epoch 170/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3568 - accuracy: 0.8636
Epoch 171/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3620 - accuracy: 0.8430
Epoch 172/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3641 - accuracy: 0.8388
Epoch 173/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3654 - accuracy: 0.8388
Epoch 174/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3583 - accuracy: 0.8512
Epoch 175/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3683 - accuracy: 0.8388
Epoch 176/300
242/242 [==============================] - 0s 0
```

```
us/step - loss: 0.3607 - accuracy: 0.8512
Epoch 177/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3794 - accuracy: 0.8182
Epoch 178/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3468 - accuracy: 0.8512
Epoch 179/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3671 - accuracy: 0.8554
Epoch 180/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3535 - accuracy: 0.8554
Epoch 181/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3571 - accuracy: 0.8554
Epoch 182/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3630 - accuracy: 0.8471
Epoch 183/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3494 - accuracy: 0.8595
Epoch 184/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3629 - accuracy: 0.8512
Epoch 185/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3546 - accuracy: 0.8595
Epoch 186/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3533 - accuracy: 0.8512
Epoch 187/300
242/242 [==============================] - 0s 6
```

```
5us/step - loss: 0.3664 - accuracy: 0.8264
Epoch 188/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3821 - accuracy: 0.8430
Epoch 189/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3580 - accuracy: 0.8595
Epoch 190/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3747 - accuracy: 0.8471
Epoch 191/300
242/242 [==============================] - 0s 6
5us/step - loss: 0.3725 - accuracy: 0.8471
Epoch 192/300
242/242 [==============================] - 0s 2
7us/step - loss: 0.3782 - accuracy: 0.8388
Epoch 193/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3662 - accuracy: 0.8388
Epoch 194/300
242/242 [==============================] - 0s 0
us/step - loss: 0.3516 - accuracy: 0.8636
```

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

In [ ]:
```
Y_pred_nn.shape
```

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

Y_pred_nn = rounded
```

In [ ]:
```
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.
```

VI. Output final score

In [ ]:
```
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 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])+" %")
```

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

sns.barplot(algorithms,scores)
```

Random forest has good result as compare to other algorithms

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: