```
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']
```

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

```
type(dataset)
```

```
pandas.core.frame.DataFrame
```

```
dataset.shape
```

```
(303, 14)
```

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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	th
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	

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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
291	58	1	0	114	318	0	2	140	0	4.4	0	3
224	54	1	0	110	239	0	1	126	1	2.8	1	1

```
dataset.describe()
```

	age	sex	cp	trestbps	chol	fbs	reste
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.5280
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.5258
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.0000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.0000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.0000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.0000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.0000

```
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
```

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

```
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: n
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
```

```
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
```

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

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

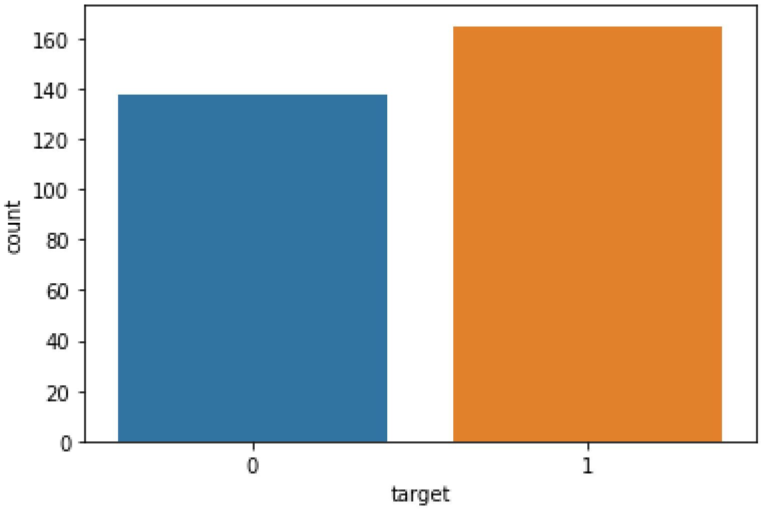
```
print(dataset.corr()["target"].abs().sort_values(ascending=False))
```

```
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
```

```
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])
```

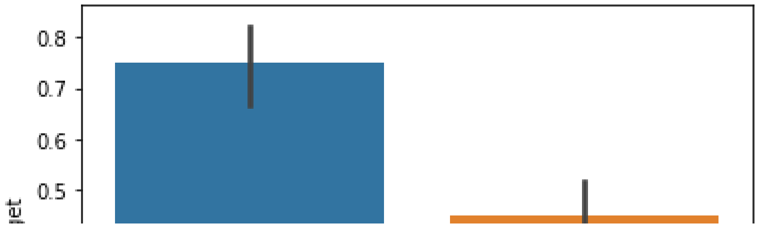
```
Percentage of patience without heart problems: 45.54
Percentage of patience with heart problems: 54.46
```

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

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

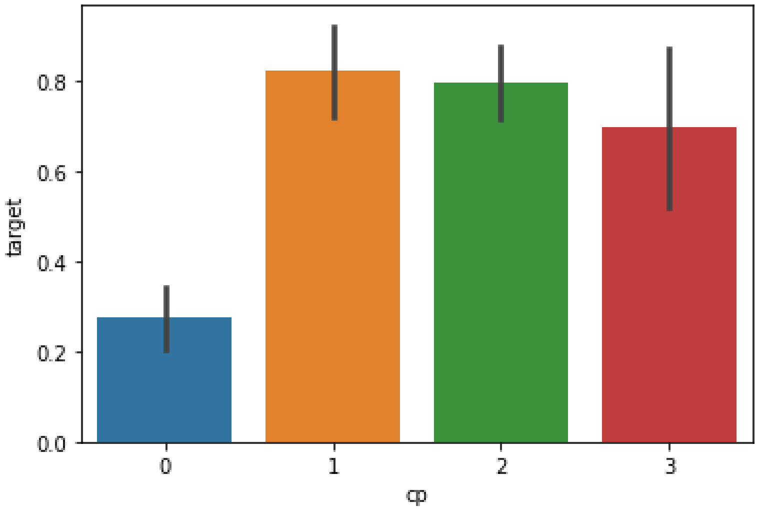
```
sns.barplot(dataset["sex"],y)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f859a1c8690>
```



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

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

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

    <matplotlib.axes._subplots.AxesSubplot at 0x7f859a152dd0>
```

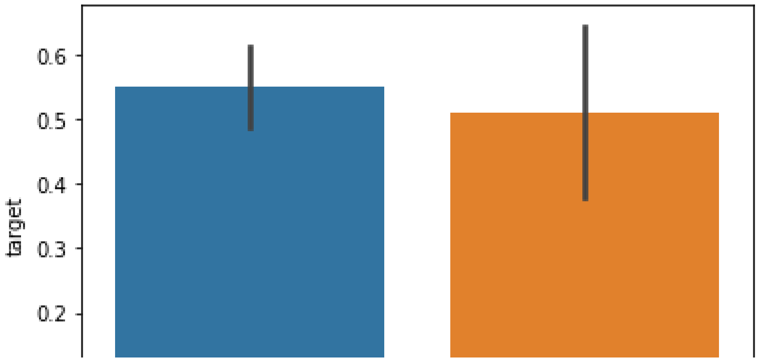


```
dataset["fbs"].describe()

    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
```

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

    array([1, 0])
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f859a0cff10>
```

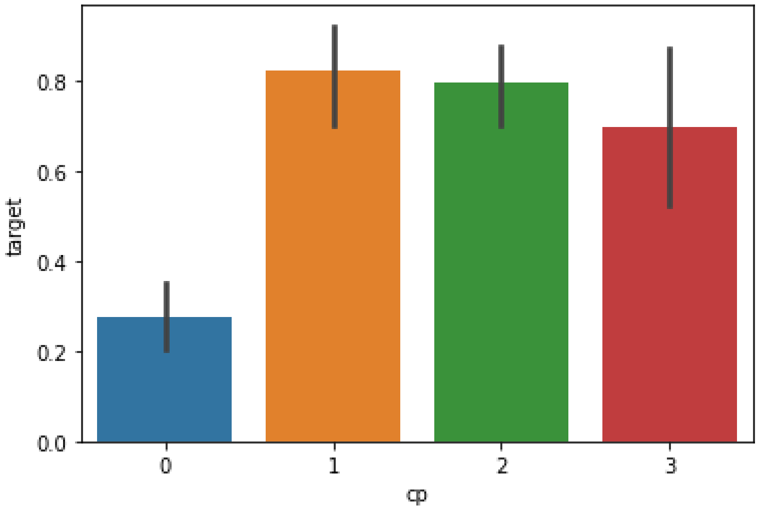


```
dataset["cp"].unique()
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f859a0b1fd0>
```



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

```
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
```

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

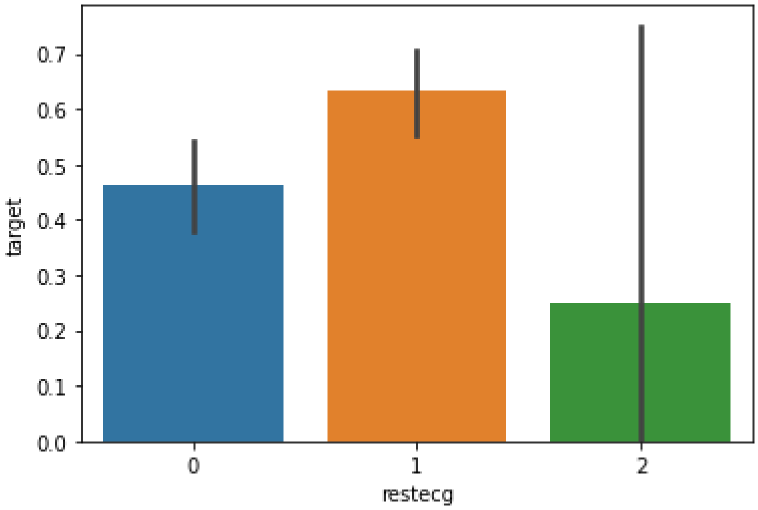
```
array([1, 0])
```

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

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

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

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f859a02e3d0>
```

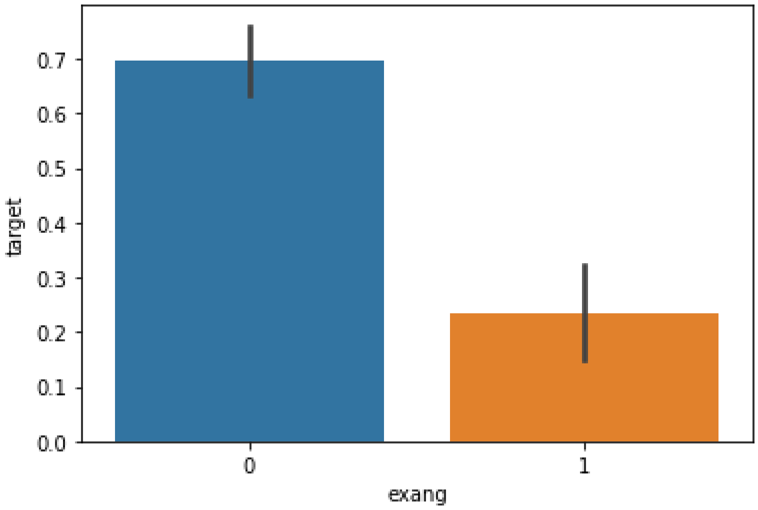


```
dataset["exang"].unique()
```

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

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

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f8599f9e310>
```

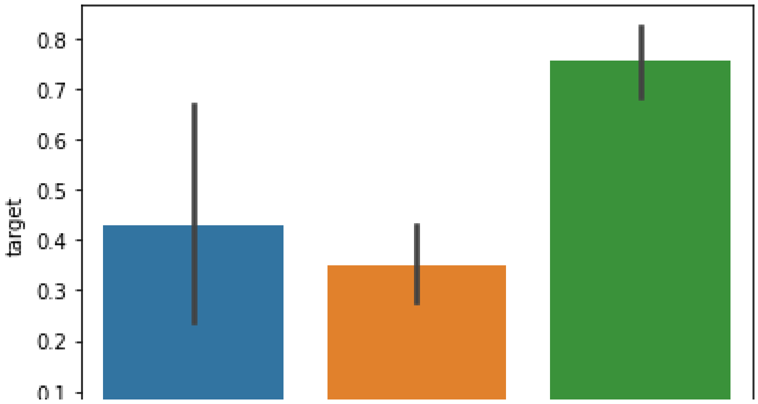


```
dataset["slope"].unique()
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8599f82e10>
```

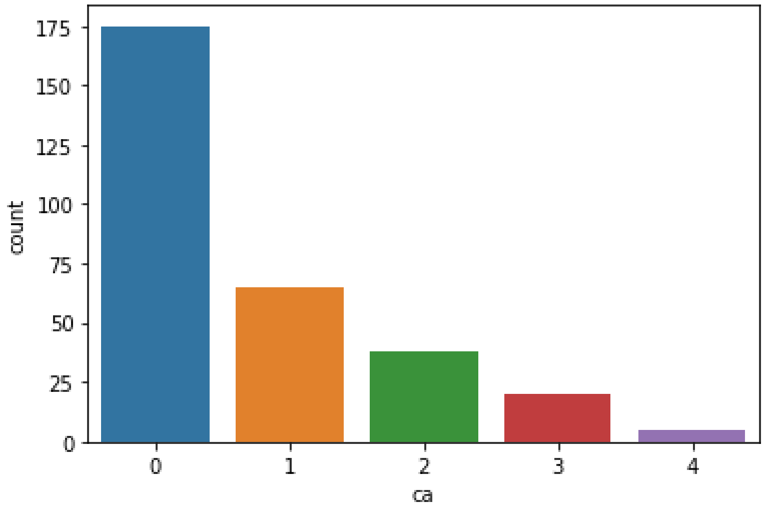


```
dataset["ca"].unique()
```

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

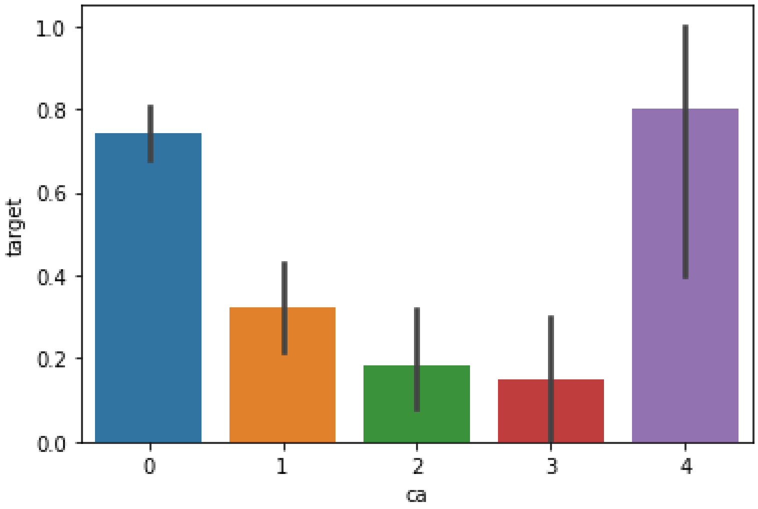
```
sns.countplot(dataset["ca"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8599f00490>
```



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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8599e78250>
```

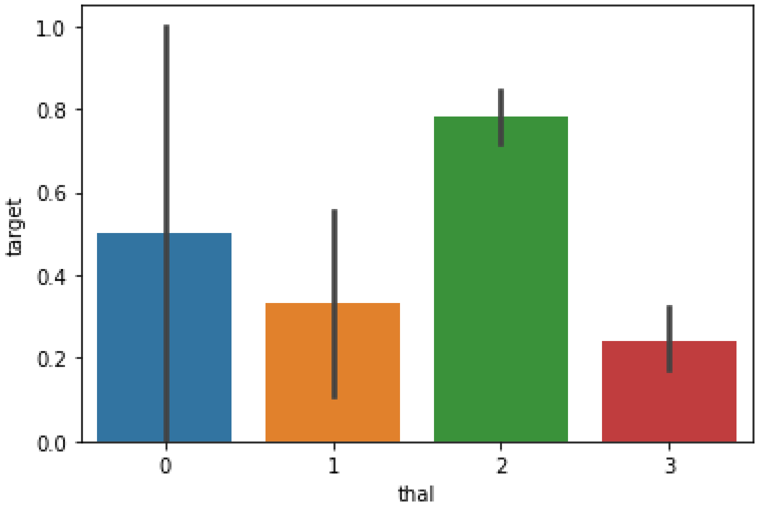


```
dataset["thal"].unique()
```

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

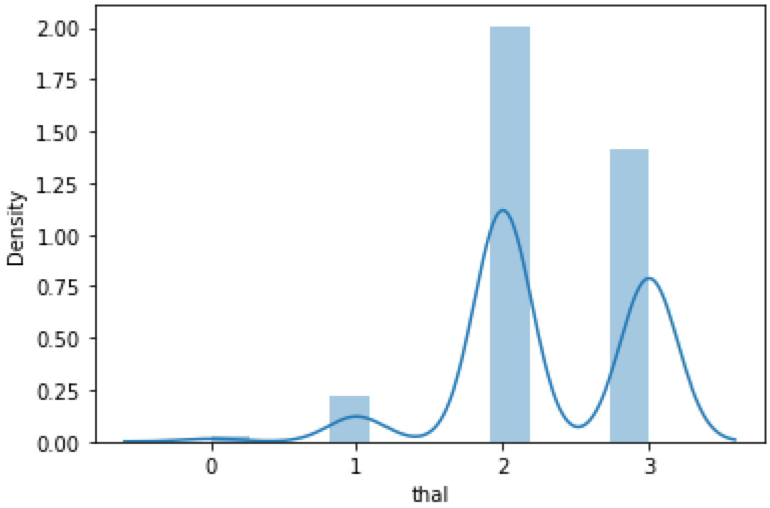
```
sns.barplot(dataset["thal"],y)
```

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f8599df4b50>
```



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

```
    <matplotlib.axes._subplots.AxesSubplot at 0x7f8599d66510>
```



```
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_s
```

```
X_train.shape
```

```
    (242, 13)
```

```
X_test.shape
```

```
     (61, 13)
```

```
Y_train.shape
```

```
     (242,)
```

```
Y_test.shape
```

```
     (61,)
```

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

```
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 %
```

```
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 %
```

```
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 %
```

```
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 %
```

```
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 %
```

```
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)
```

```
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: 90.16 %
```

```
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: 85.25 %
```

```
import tensorflow as tf
from tensorflow.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)
```

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

```
Y_pred_nn.shape
```

```
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
```

```
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 Neig

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

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

sns.barplot(algorithms,scores)


input_data = (62,0,0,140,268,0,0,160,0,3.6,0,2,2)

# change the input data to a numpy array
input_data_as_numpy_array= np.asarray(input_data)

# reshape the numpy array as we are predicting for only on instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]== 0):
  print('The Person does not have a Heart Disease')
else:
  print('The Person has Heart Disease')
```