Health Care

Cardiovascular diseases are the leading cause of death globally. It is therefore necessary to identify the causes and develop a system to predict heart attacks in an effective manner. The data below has the information about the factors that might have an impact on cardiovascular health.

datasets

Variable Description Age Age in years Sex 1 = male; 0 = female cp| Chest pain type trestbps Resting blood pressure (in mm Hg on admission to the hospital) chol Serum cholesterol in mg/dl fbs Fasting blood sugar > 120 mg/dl (1 = true; 0 = false) restecg Resting electrocardiographic results thalach Maximum heart rate achieved exang Exercise induced angina (1 = yes; 0 = no) oldpeak ST depression induced by exercise relative to rest slope Slope of the peak exercise ST segment

importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# import dataset
df = pd.read_excel(r'D:\family\eswar\simplilearn\ml\ml datasets health care.xlsx')
```

In [3]:
```
df.shape
# sahpe of dataset
```

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

In [4]:
```
df.head()
# printing head columns
```

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

In [67]:
```
df.tail()
# printing tail columns
```

Out[67]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

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

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	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
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	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
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	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
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

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
```

In [8]:
```
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: asympto
matic
trestbps:                       resting blood pressure
chol:                   Serum cholesterol 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:                        ST depression induced by exercise relative to rest
slope:                  Slope of the peak exercise ST segment
```

In [9]:
```
df["target"].describe()
# analyzing the target variables
```

Out[9]:
```
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 [10]:
```
df["target"].unique()
```

Out[10]:
```
array([1, 0], dtype=int64)
```

In [11]:
```
print(df.corr()["target"].abs().sort_values(ascending=False))
# checking correlation between columns
```

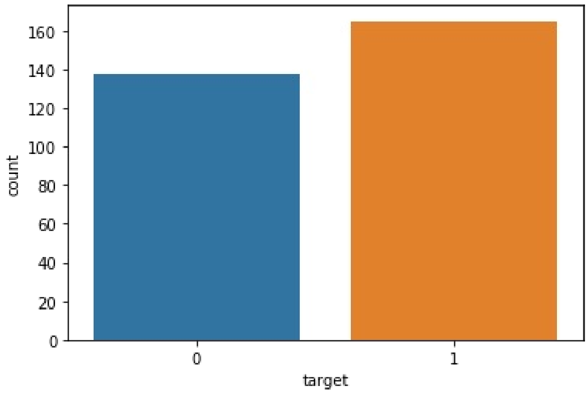
```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: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

fbs is weakly correlated

exploratory data analysis (EDA)

In [12]:
```
y = df["target"]
sns.countplot(y)
target_temp = df.target.value_counts()
print(target_temp)
# analyzing the target variables
```

```
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)))
```

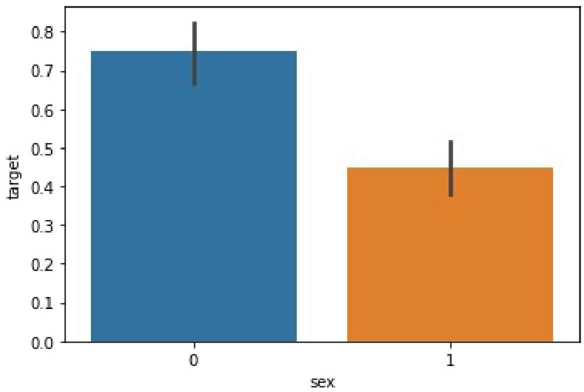
```
Percentage of patience without heart problems: 45.54
Percentage of patience with heart problems: 54.46
```

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'>
```

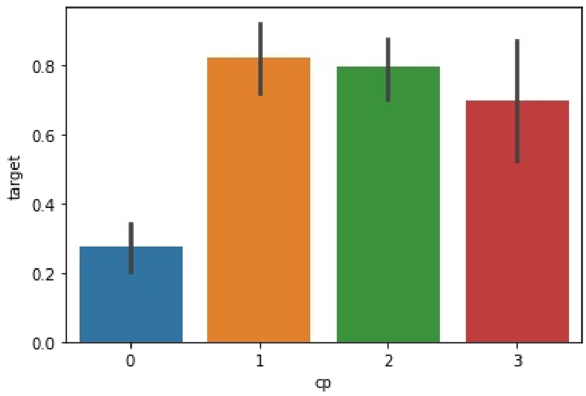


In [16]:
```
df["cp"].unique()
```

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

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

Out[17]:
```
<AxesSubplot:xlabel='cp', ylabel='target'>
```



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

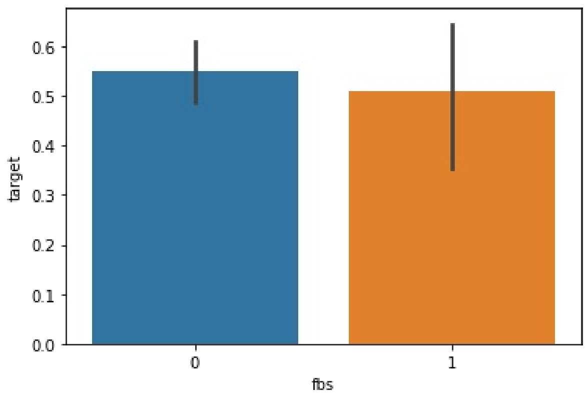
Out[18]:
```
count    303.000000
mean       0.148515
std        0.356198
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: fbs, dtype: float64
```

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'>
```

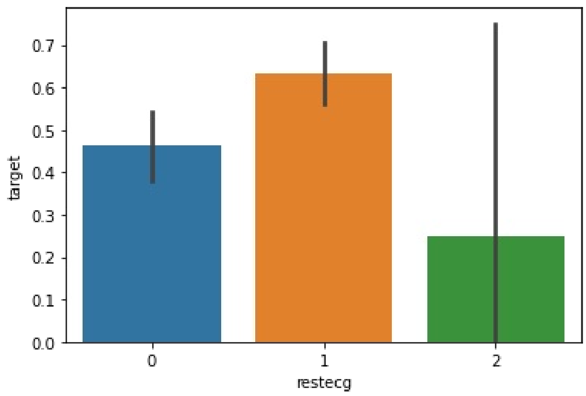


In [21]:
```
df["restecg"].unique()
```

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

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

Out[22]:
```
<AxesSubplot:xlabel='restecg', ylabel='target'>
```

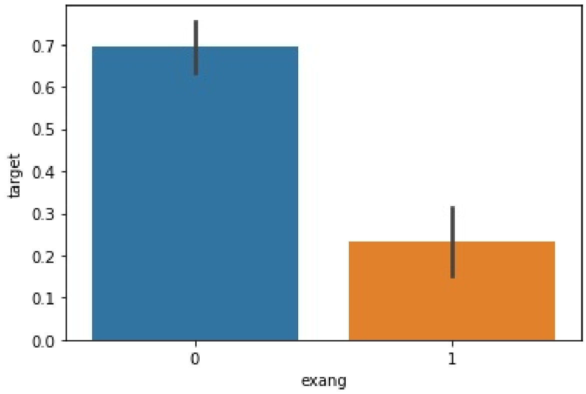


In [23]:
```
df["exang"].unique()
```

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

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

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

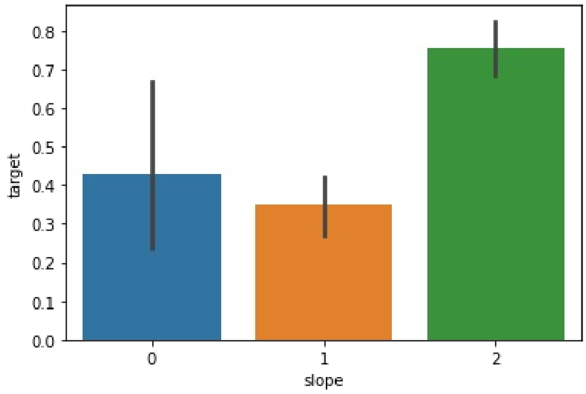


In [25]:
```python
df["slope"].unique()
```

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

In [26]:
```python
sns.barplot(df["slope"],y)
```

Out[26]: <AxesSubplot:xlabel='slope', ylabel='target'>

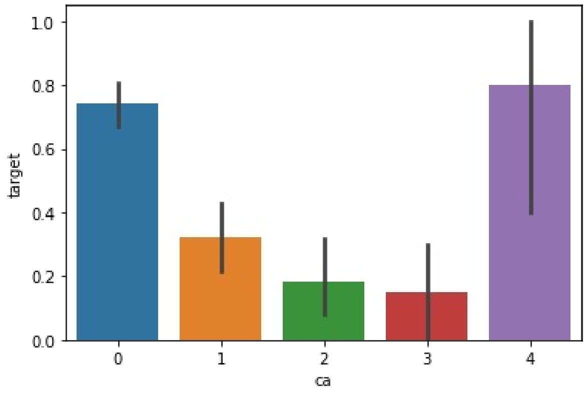


In [27]:
```python
df["ca"].unique()
```

Out[27]: array([0, 2, 1, 3, 4], dtype=int64)

In [28]:
```python
sns.barplot(df["ca"],y)
```

Out[28]: <AxesSubplot:xlabel='ca', ylabel='target'>

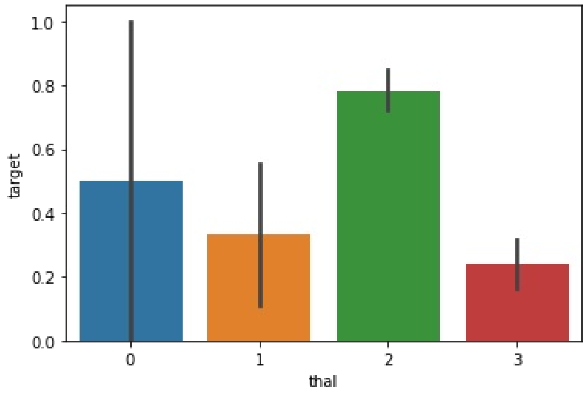


In [29]:
```python
df["thal"].unique()
```

Out[29]: array([1, 2, 3, 0], dtype=int64)

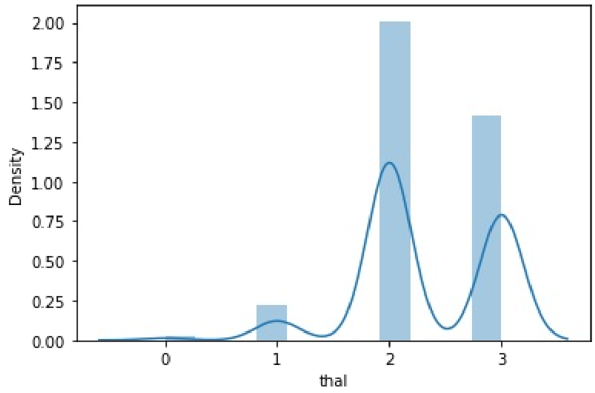
In [30]:
```python
sns.barplot(df["thal"],y)
```

Out[30]: <AxesSubplot:xlabel='thal', ylabel='target'>



In [31]:
```
sns.distplot(df["thal"])
```

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



TRAIN TEST SPLIT

In [32]:
```
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 [33]:
```
X_train.shape
```

Out[33]: (242, 13)

In [34]:
```
X_test.shape
```

Out[34]: (61, 13)

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

Out[35]: (242,)

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

Out[36]: (61,)

MODEL FITTING

In [37]:
```
from sklearn.metrics import accuracy_score
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,Y_train)
Y_pred_lr = lr.predict(X_test)
```

In [39]:
```
Y_pred_lr.shape
```
Out[39]:
```
(61,)
```
In [40]:
```
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 %
```
In [41]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train,Y_train)
Y_pred_nb = nb.predict(X_test)
```
In [42]:
```
Y_pred_nb.shape
```
Out[42]:
```
(61,)
```
In [43]:
```
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 %
```
In [44]:
```
from sklearn import svm
sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred_svm = sv.predict(X_test)
```
In [45]:
```
Y_pred_svm.shape
```
Out[45]:
```
(61,)
```
In [46]:
```
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 %
```
In [47]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
```
In [48]:
```
Y_pred_knn.shape
```
Out[48]:
```
(61,)
```
In [49]:
```
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 [50]:
```
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 [51]:
```
print(Y_pred_dt.shape)
```
```
(61,)
```
In [52]:
```
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 [53]:
```
from sklearn.ensemble import RandomForestClassifier
```

```python
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 [54]:
```python
Y_pred_rf.shape
```

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

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

xg boost

In [61]:
```python
pip install xgboost
```

```
Collecting xgboost
  Using cached xgboost-1.6.2-py3-none-win_amd64.whl (125.4 MB)
Requirement already satisfied: scipy in c:\users\eswar\anaconda3\lib\site-packages (from xgboost) (1.7.3)
Requirement already satisfied: numpy in c:\users\eswar\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Installing collected packages: xgboost
Successfully installed xgboost-1.6.2
Note: you may need to restart the kernel to use updated packages.
```

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

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

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

In [64]:
```python
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)
print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

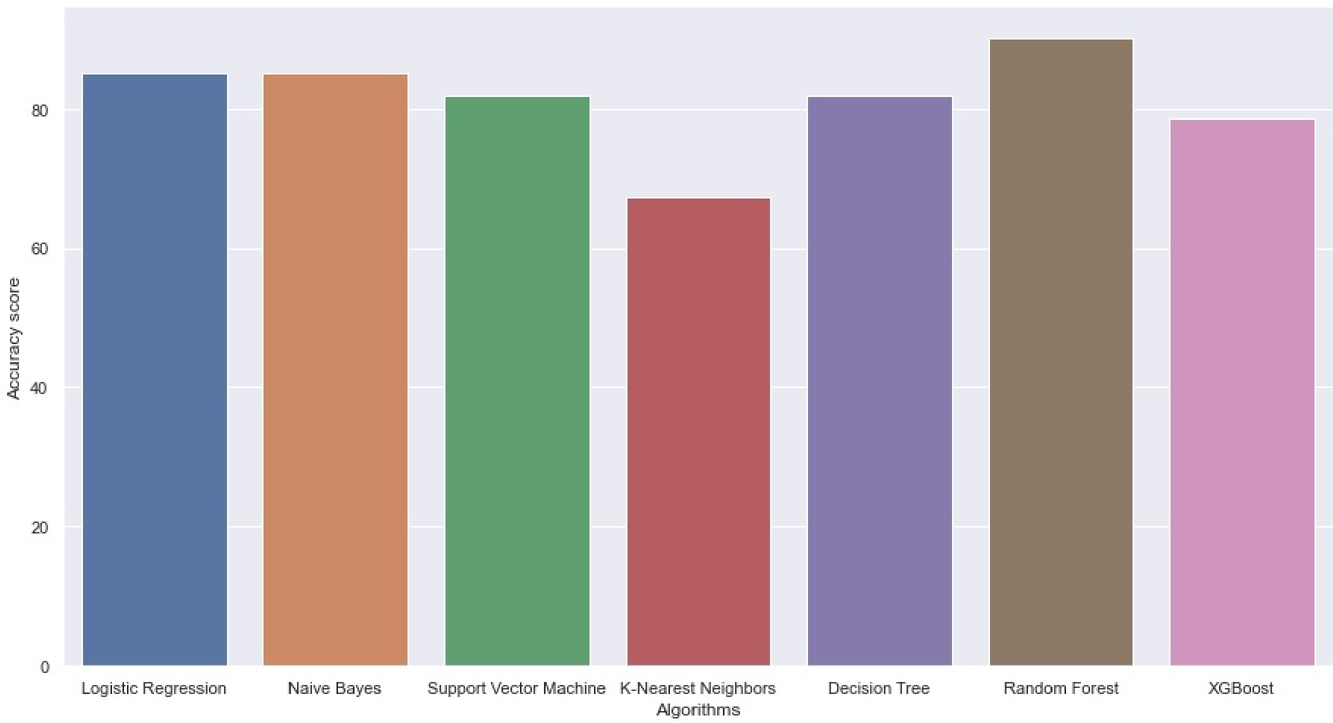
```
The accuracy score achieved using XGBoost is: 78.69 %
```

In [65]:
```python
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
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 [66]:
```python
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")
sns.barplot(algorithms,scores)
```

Out[66]:
```
<AxesSubplot:xlabel='Algorithms', ylabel='Accuracy score'>
```




```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js