OBJECTIVE:

preliminary prognosis of Hypertension/hypotension, based on the level of hemoglobin and genetic history of the individual.

In [2]:
```
import pandas as pd
import numpy as np
df = pd.read_csv("data.csv")
```

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

Out[3]:
```
(2000, 15)
```

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

Out[4]:

	Patient_Number	Blood_Pressure_Abnormality	Level_of_Hemoglobin	Genetic_Pedigree_Coefficient
0	1	1	11.28	0.90
1	2	0	9.75	0.23
2	3	1	10.79	0.91
3	4	0	11.00	0.43
4	5	1	14.17	0.83

In [5]:
```
df.describe(include='all')
```

Out[5]:

	Patient_Number	Blood_Pressure_Abnormality	Level_of_Hemoglobin	Genetic_Pedigree_Coeffi
count	2000.000000	2000.000000	2000.000000	1908.00
mean	1000.500000	0.493500	11.710035	0.49
std	577.494589	0.500083	2.186701	0.29
min	1.000000	0.000000	8.100000	0.00
25%	500.750000	0.000000	10.147500	0.24
50%	1000.500000	0.000000	11.330000	0.49
75%	1500.250000	1.000000	12.945000	0.74
max	2000.000000	1.000000	17.560000	1.00

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 15 columns):
Patient_Number                    2000 non-null int64
Blood_Pressure_Abnormality        2000 non-null int64
Level_of_Hemoglobin               2000 non-null float64
Genetic_Pedigree_Coefficient      1908 non-null float64
Age                               2000 non-null int64
BMI                               2000 non-null int64
Sex                               2000 non-null int64
Pregnancy                         442 non-null float64
Smoking                           2000 non-null int64
Physical_activity                 2000 non-null int64
salt_content_in_the_diet          2000 non-null int64
alcohol_consumption_per_day       1758 non-null float64
Level_of_Stress                   2000 non-null int64
Chronic_kidney_disease            2000 non-null int64
Adrenal_and_thyroid_disorders     2000 non-null int64
dtypes: float64(4), int64(11)
memory usage: 234.5 KB
```

Missing Value Treatment

In [7]:
```
df.fillna(0)
```

Out[7]:

	Patient_Number	Blood_Pressure_Abnormality	Level_of_Hemoglobin	Genetic_Pedigree_Coeffici
0	1	1	11.28	0
1	2	0	9.75	0
2	3	1	10.79	0
3	4	0	11.00	0
4	5	1	14.17	0
5	6	0	11.64	0
6	7	1	11.69	0
7	8	0	12.70	0
8	9	0	10.88	0
9	10	1	14.56	0
10	11	1	8.58	0
11	12	1	12.77	0
12	13	1	16.40	0
13	14	0	16.42	0
14	15	0	11.97	0
15	16	1	10.96	0
16	17	1	11.98	0
17	18	1	11.60	0
18	19	1	8.99	0
19	20	1	16.55	0
20	21	1	16.95	0
21	22	0	10.85	0
22	23	0	11.19	0
23	24	0	10.61	0
24	25	0	10.26	0
25	26	1	13.29	0
26	27	0	10.40	0
27	28	0	10.55	0
28	29	1	9.37	0
29	30	0	11.17	0
...	...	...	...	
1970	1971	1	11.26	0
1971	1972	0	12.53	0
1972	1973	0	9.82	0

	Patient_Number	Blood_Pressure_Abnormality	Level_of_Hemoglobin	Genetic_Pedigree_Coeffici
1973	1974	1	11.45	0
1974	1975	1	8.47	0
1975	1976	1	8.75	0
1976	1977	0	12.07	0
1977	1978	0	11.16	0
1978	1979	0	11.57	0
1979	1980	0	11.70	0
1980	1981	0	12.34	0
1981	1982	1	13.49	0
1982	1983	0	9.08	0
1983	1984	1	14.56	0
1984	1985	0	10.81	0
1985	1986	0	10.44	0
1986	1987	1	13.63	0
1987	1988	1	11.79	0
1988	1989	1	17.17	0
1989	1990	1	10.84	0
1990	1991	1	11.21	0
1991	1992	1	15.53	0
1992	1993	1	9.38	0
1993	1994	0	9.69	1
1994	1995	0	11.07	0
1995	1996	1	10.14	0
1996	1997	1	11.77	1
1997	1998	1	16.91	0
1998	1999	0	11.15	0
1999	2000	1	11.36	0

2000 rows × 15 columns

In [8]:
```
df.isnull().any(axis=0)
```

Out[8]:
```
Patient_Number                  False
Blood_Pressure_Abnormality      False
Level_of_Hemoglobin             False
Genetic_Pedigree_Coefficient     True
Age                             False
BMI                             False
Sex                             False
Pregnancy                        True
Smoking                         False
Physical_activity               False
salt_content_in_the_diet        False
alcohol_consumption_per_day      True
Level_of_Stress                 False
Chronic_kidney_disease          False
Adrenal_and_thyroid_disorders   False
dtype: bool
```

In [9]:
```
df['Genetic_Pedigree_Coefficient'].isnull().value_counts()
```

Out[9]:
```
False    1908
True       92
Name: Genetic_Pedigree_Coefficient, dtype: int64
```

In [10]:
```
#median imputation
median = df['Genetic_Pedigree_Coefficient'].median()
df['Genetic_Pedigree_Coefficient'] = df['Genetic_Pedigree_Coefficient'].fillna(m
```

In [11]:
```
#checking for hidden missing values
print("# rows in dataframe {0}".format(len(df)))
print("# rows missing in Level_of_Hemoglobin: {0}".format(df['Level_of_Hemoglobi
print("# rows missing in Genetic_Pedigree_Coefficient: {0}".format(df['Genetic_P
```

```
# rows in dataframe 2000
# rows missing in Level_of_Hemoglobin: 0
# rows missing in Genetic_Pedigree_Coefficient: 0
```

In [12]:
```
#check True/False ratio
num_false = len(df[df['Blood_Pressure_Abnormality'] == 0])
num_true = len(df[df['Blood_Pressure_Abnormality'] == 1])
print("no. of false cases: {0} ({1:2.2f}%)".format(num_false, (num_false*100/(nu
print("no. of true cases: {0} ({1:2.2f}%)".format(num_true, (num_true*100/(num_f
#both cases have fair amount of cases standard predictions can be used
#Good distribution of true and false cases. No special work needed.
```

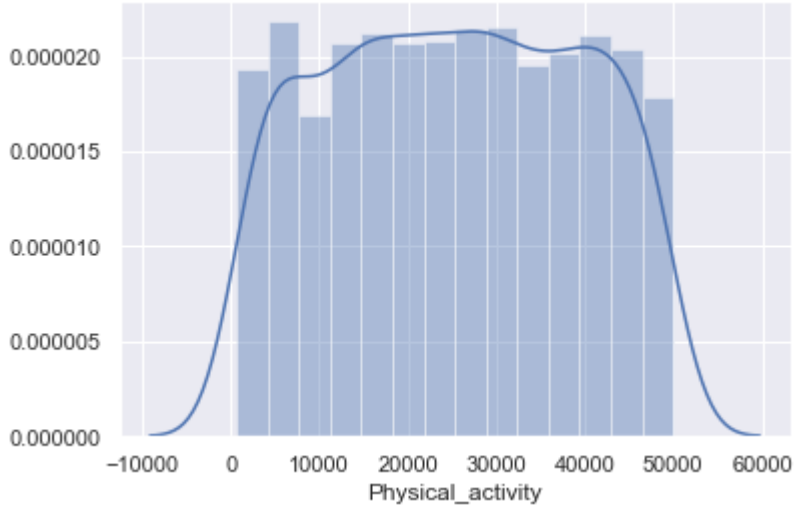
```
no. of false cases: 1013 (50.65%)
no. of true cases: 987 (49.35%)
```

In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.distplot(df['BMI'])
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x23a4e4b8b70>

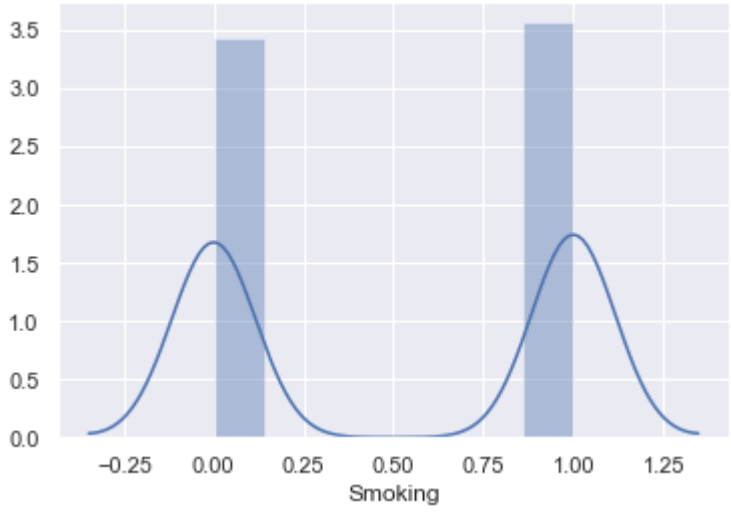
In [14]:
```
sns.distplot(df['Physical_activity'])
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x23a4fb7e860>



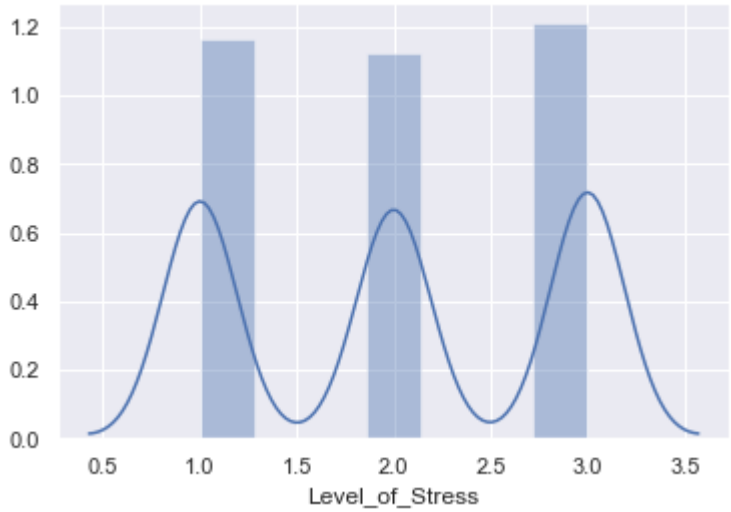
In [15]:
```
sns.distplot(df['Smoking'])
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x23a4fc299b0>



In [16]:
```
sns.distplot(df['Level_of_Stress'])
```

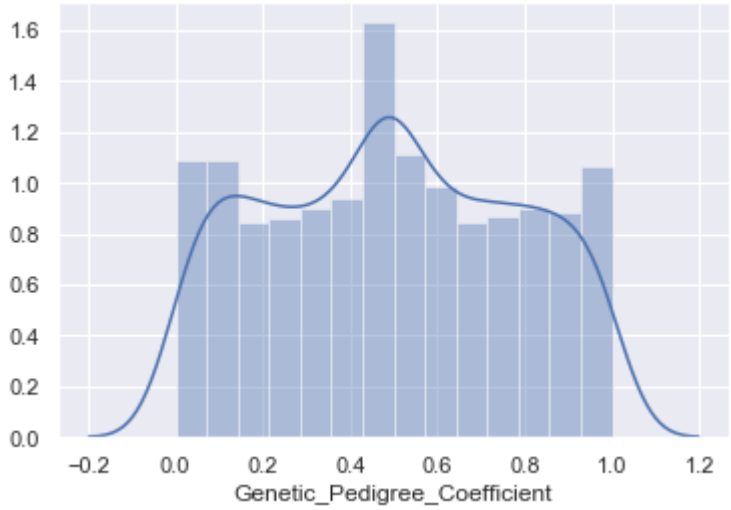
Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a4ffcebe0>
```



In [17]:
```
df['Genetic_Pedigree_Coefficient'].isnull().value_counts()
```

Out[17]:
```
False    2000
Name: Genetic_Pedigree_Coefficient, dtype: int64
```

In [18]:
```
sns.distplot(df['Genetic_Pedigree_Coefficient']);
```



In [19]:
```
#print("Male Patient: {}".format(df[df["Sex"]== 0]['Sex'].count()))
#print("Female Patient: {}".format(df[df["Sex"]== 1]['Sex'].count()))
```

In [20]:
```
#df[df["Sex"]== 1]['Pregnancy'].isnull().value_counts()
```

In [21]:
```
#df[df["Sex"]== 0]['Pregnancy'].isnull().value_counts()
```

In [22]:
```
df["Pregnancy"]=df["Pregnancy"].replace(df[df["Sex"]== 0]['Pregnancy'] ,0 )
```

In [23]:
```
df["Pregnancy"]=df["Pregnancy"].replace(np.nan,0 )
df["Pregnancy"].isnull().value_counts()
```

Out[23]:
```
False    2000
Name: Pregnancy, dtype: int64
```

Split Dataset(70,30)

In [25]:
```
from sklearn.model_selection import train_test_split
feature_col_names = ['Genetic_Pedigree_Coefficient','BMI','Physical_activity','L
predicted_class_names = ['Blood_Pressure_Abnormality']
X= df[feature_col_names].values
y= df[predicted_class_names].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [26]:
```
#check to ensure we have the the desired 70% train, 30% test split of the data
print("{0:0.2f}% in training set".format((len(X_train)/len(df.index)) * 100))
print("{0:0.2f}% in test set".format((len(X_test)/len(df.index)) * 100))
```

```
70.00% in training set
30.00% in test set
```

In [27]:
```
#Verifying predicted value was split correctly

print("Original True  : {0} ({1:0.2f}%)".format(len(df.loc[df['Blood_Pressure_Abn
print("Original False : {0} ({1:0.2f}%)".format(len(df.loc[df['Blood_Pressure_Abn
print("")
print("Training True  : {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 1]), (
print("Training False : {0} ({1:0.2f}%)".format(len(y_train[y_train[:] == 0]), (
print("")
print("Test True      : {0} ({1:0.2f}%)".format(len(y_test[y_test[:] == 1]), (len
print("Test False     : {0} ({1:0.2f}%)".format(len(y_test[y_test[:] == 0]), (len
```

```
Original True  : 987 (49.35%)
Original False : 1013 (50.65%)

Training True  : 707 (50.50%)
Training False : 693 (49.50%)

Test True      : 280 (46.67%)
Test False     : 320 (53.33%)
```

NAIVE BAYES

In [28]:
```
#Training the model
from sklearn.naive_bayes import GaussianNB

# create Gaussian Naive Bayes model object and train it with the data
nb_model = GaussianNB()

nb_model.fit(X_train, y_train.ravel())
```

Out[28]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [29]:
```
# predict values using the training data
nb_predict_train = nb_model.predict(X_train)

# import the performance metrics library
from sklearn import metrics

# Accuracy
print("Accuracy on Training data: {0:.4f}".format(metrics.accuracy_score(y_train,
print()
```

```
Accuracy on Training data: 0.5893
```

In [30]:
```
# predict values using the testing data
nb_predict_test = nb_model.predict(X_test)

# Accuracy
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
Accuracy on Test data: 0.5983
```

In [31]:
```
score_nb1=metrics.accuracy_score(y_test,nb_predict_test)
print(score_nb1)
#print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
0.5983333333333334
Accuracy on Test data: 0.5983
```

In [32]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,nb_predict_test)
```

Out[32]:
```
array([[254,  66],
       [175, 105]], dtype=int64)
```

In [33]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,nb_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.59      0.79      0.68       320
           1       0.61      0.38      0.47       280

   micro avg       0.60      0.60      0.60       600
   macro avg       0.60      0.58      0.57       600
weighted avg       0.60      0.60      0.58       600
```

SUPPORT VECTOR MACHINE

In [34]:
```
from sklearn.svm import SVC
model=SVC(gamma='auto')
```

In [35]:
```
model.fit(X_train,y_train.ravel())
```

Out[35]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [36]:
```
prediction_test=model.predict(X_test)
```

In [37]:
```
print("accuracy on training data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on training data :0.4617
```

In [38]:
```
prediction_test=model.predict(X_test)
score_svm1=metrics.accuracy_score(y_test,prediction_test)
print(score_svm1)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.46166666666666667
Accuracy= 0.46166666666666667
```

In [39]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[39]:
```
array([[  3, 317],
       [  6, 274]], dtype=int64)
```

In [40]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.33      0.01      0.02       320
           1       0.46      0.98      0.63       280

   micro avg       0.46      0.46      0.46       600
   macro avg       0.40      0.49      0.32       600
weighted avg       0.39      0.46      0.30       600
```

DECISION TREE

In [41]:
```
from sklearn import tree
model=tree.DecisionTreeClassifier()
```

In [42]:
```
model.fit(X_train,y_train.ravel())
```

Out[42]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [43]:
```
prediction_test=model.predict(X_test)
```

In [44]:
```
prediction_test=model.predict(X_test)
score_dt1=metrics.accuracy_score(y_test,prediction_test)
print(score_dt1)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.6466666666666666
Accuracy= 0.6466666666666666
```

In [45]:
```
print("accuracy on training data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on training data :0.6467
```

In [46]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[46]:
```
array([[212, 108],
       [104, 176]], dtype=int64)
```

In [47]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.67      0.66      0.67       320
           1       0.62      0.63      0.62       280

   micro avg       0.65      0.65      0.65       600
   macro avg       0.65      0.65      0.65       600
weighted avg       0.65      0.65      0.65       600
```

RANDOM FOREST

In [48]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
score_rf1=(metrics.accuracy_score(y_test,prediction_test))
print(score_rf1)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.6933333333333334
Accuracy= 0.6933333333333334
```

In [49]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[49]:
```
array([[243,  77],
       [107, 173]], dtype=int64)
```

In [50]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.69      0.76      0.73       320
           1       0.69      0.62      0.65       280

   micro avg       0.69      0.69      0.69       600
   macro avg       0.69      0.69      0.69       600
weighted avg       0.69      0.69      0.69       600
```

GRADIENT BOOSTING

In [51]:
```
from sklearn.ensemble import GradientBoostingClassifier
model=GradientBoostingClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
score_gb1=format(model.score(X_test,y_test))
print("accuracy on testing data :{0:.4f}",score_gb1)
#print("accuracy on testing data :{0:.4f}".format(model.score(X_test,y_test)))
print("accuracy on testing data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on testing data :{0:.4f} 0.7566666666666667
accuracy on testing data :0.7567
```

In [52]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[52]:
```
array([[292,  28],
       [118, 162]], dtype=int64)
```

In [53]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.71      0.91      0.80       320
           1       0.85      0.58      0.69       280

   micro avg       0.76      0.76      0.76       600
   macro avg       0.78      0.75      0.74       600
weighted avg       0.78      0.76      0.75       600
```

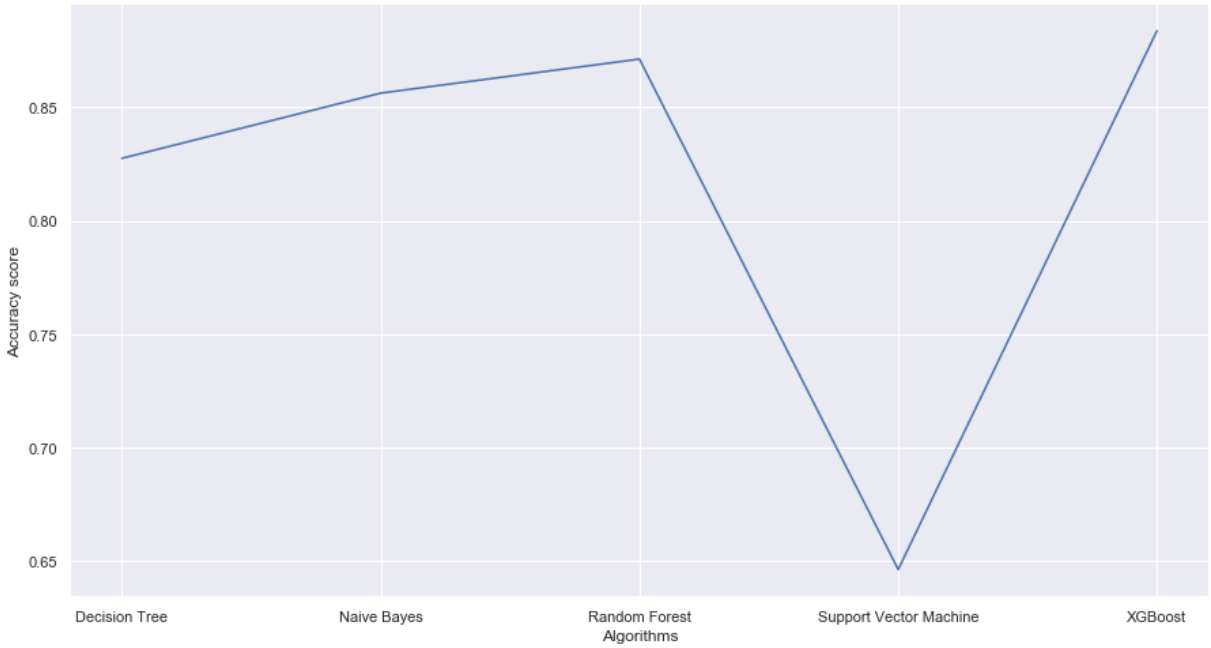
In [54]:
```
scores = [score_nb1,score_svm1,score_dt1,score_rf1,score_gb1]
algorithms = ["Naive Bayes","Support Vector Machine","Decision Tree","Random Fore

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

```
The accuracy score achieved using Naive Bayes is: 0.5983333333333334 %
The accuracy score achieved using Support Vector Machine is: 0.4616666666666666
7 %
The accuracy score achieved using Decision Tree is: 0.6466666666666666 %
The accuracy score achieved using Random Forest is: 0.6933333333333334 %
The accuracy score achieved using XGBoost is: 0.7566666666666667 %
```

In [98]:
```
scores = [score_nb,score_svm,score_dt,score_rf,score_gb]
algorithms = ["Naive Bayes","Support Vector Machine","Decision Tree","Random Fore
```

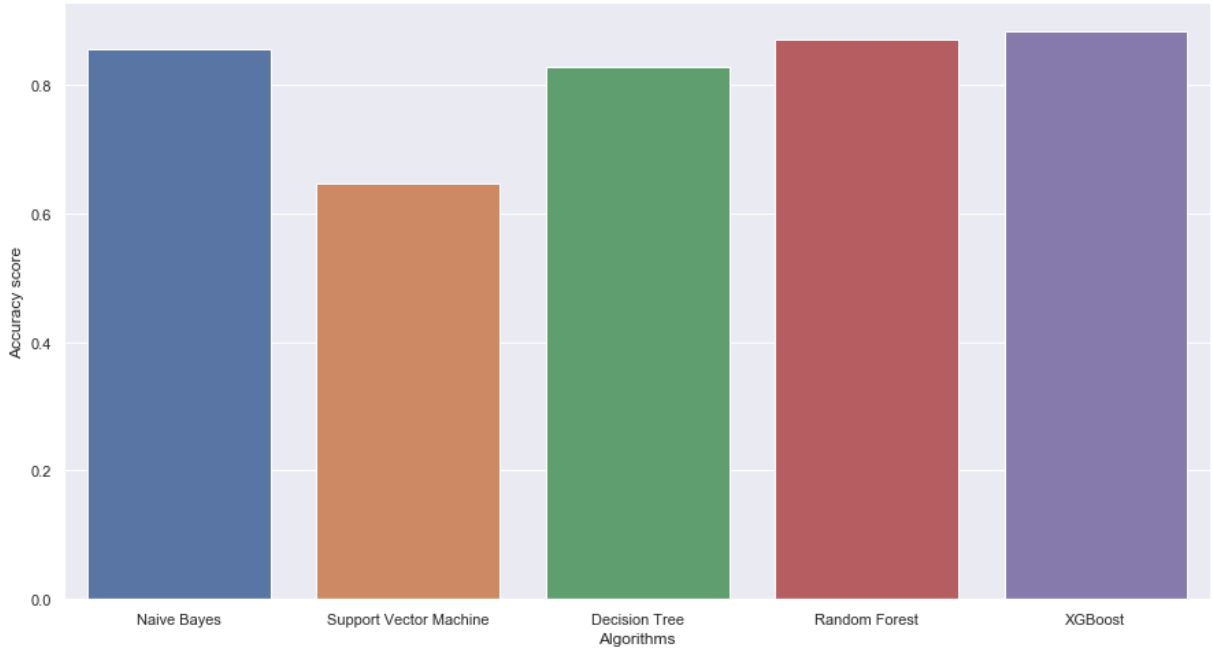
In [99]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.lineplot(algorithms,scores)
```

Out[99]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a50131278>
```



In [100]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms,scores)
```

Out[100]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a5085e240>
```



Split Dataset

In [55]:
```
from sklearn.model_selection import train_test_split
feature_col_names = ['Level_of_Hemoglobin', 'Genetic_Pedigree_Coefficient']
predicted_class_names = ['Blood_Pressure_Abnormality']
X= df[feature_col_names].values
y= df[predicted_class_names].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_
```

NAIVE BAYES

In [56]:
```
#Training the model
from sklearn.naive_bayes import GaussianNB

# create Gaussian Naive Bayes model object and train it with the data
nb_model = GaussianNB()

nb_model.fit(X_train, y_train.ravel())
```

Out[56]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [57]:
```
# predict values using the testing data
nb_predict_test = nb_model.predict(X_test)

# Accuracy
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
Accuracy on Test data: 0.8657
```

In [58]:
```
nb_predict_test = nb_model.predict(X_test)

# Accuracy
score_nb=metrics.accuracy_score(y_test,nb_predict_test)
print(score_nb)
#print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
0.8657142857142858
Accuracy on Test data: 0.8657
```

In [59]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,nb_predict_test)
```

Out[59]:
```
array([[322,  45],
       [ 49, 284]], dtype=int64)
```

In [60]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,nb_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       367
           1       0.86      0.85      0.86       333

   micro avg       0.87      0.87      0.87       700
   macro avg       0.87      0.87      0.87       700
weighted avg       0.87      0.87      0.87       700
```

SUPPORT VECTOR MACHINE

In [61]:
```
from sklearn.svm import SVC
model=SVC(gamma='auto')
```

In [62]:
```
model.fit(X_train,y_train.ravel())
```

Out[62]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [63]:
```
prediction_test=model.predict(X_test)
score_svm=metrics.accuracy_score(y_test,prediction_test)
print(score_svm)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.8628571428571429
Accuracy= 0.8628571428571429
```

In [64]:
```
print("accuracy on testing data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on testing data :0.8629
```

In [65]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[65]:
```
array([[335,  32],
       [ 64, 269]], dtype=int64)
```

In [66]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87       367
           1       0.89      0.81      0.85       333

   micro avg       0.86      0.86      0.86       700
   macro avg       0.87      0.86      0.86       700
weighted avg       0.87      0.86      0.86       700
```

DECISION TREE

In [67]:
```
from sklearn import tree
model=tree.DecisionTreeClassifier()
```

In [68]:
```
model.fit(X_train,y_train.ravel())
```

Out[68]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [69]:
```
prediction_test=model.predict(X_test)
score_dt=metrics.accuracy_score(y_test,prediction_test)
print(score_dt)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.8514285714285714
Accuracy= 0.8514285714285714
```

In [70]:
```
print("accuracy on testing data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on testing data :0.8514
```

In [71]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[71]:
```
array([[322,  45],
       [ 59, 274]], dtype=int64)
```

In [72]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.88      0.86       367
           1       0.86      0.82      0.84       333

   micro avg       0.85      0.85      0.85       700
   macro avg       0.85      0.85      0.85       700
weighted avg       0.85      0.85      0.85       700
```

RANDOM FOREST

In [73]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
scores_rf=(metrics.accuracy_score(y_test,prediction_test))
print("Accuracy on test data=",scores_rf)
```

```
Accuracy on test data= 0.9057142857142857
```

In [74]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[74]:
```
array([[342,  25],
       [ 41, 292]], dtype=int64)
```

In [75]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       367
           1       0.92      0.88      0.90       333

   micro avg       0.91      0.91      0.91       700
   macro avg       0.91      0.90      0.91       700
weighted avg       0.91      0.91      0.91       700
```

GRADIENT BOOSTING

In [76]:
```
from sklearn.ensemble import GradientBoostingClassifier
model=GradientBoostingClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
score_gb=format(model.score(X_test,y_test))
print("accuracy on testing data :{0:.4f}",score_gb)
print("accuracy on testing data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on testing data :{0:.4f} 0.9
accuracy on testing data :0.9000
```

In [77]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[77]:
```
array([[342,  25],
       [ 45, 288]], dtype=int64)
```

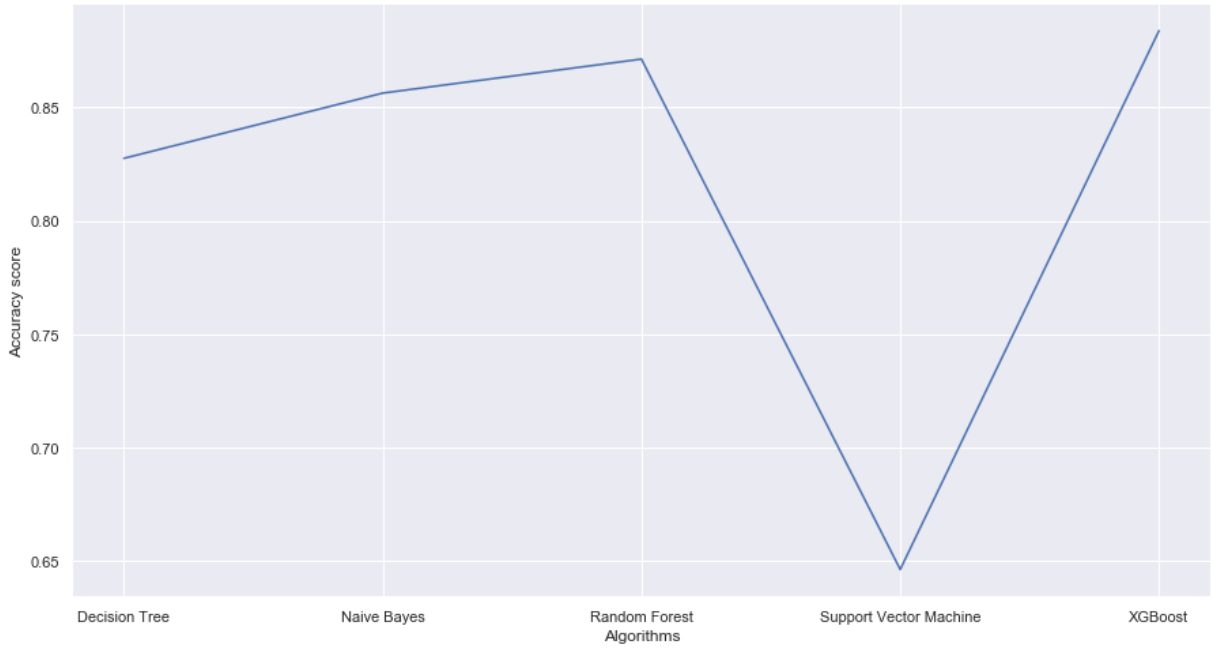
In [78]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.91       367
           1       0.92      0.86      0.89       333

   micro avg       0.90      0.90      0.90       700
   macro avg       0.90      0.90      0.90       700
weighted avg       0.90      0.90      0.90       700
```

In [101]:
```
scores = [score_nb,score_svm,score_dt,score_rf,score_gb]
algorithms = ["Naive Bayes","Support Vector Machine","Decision Tree","Random Fore
```

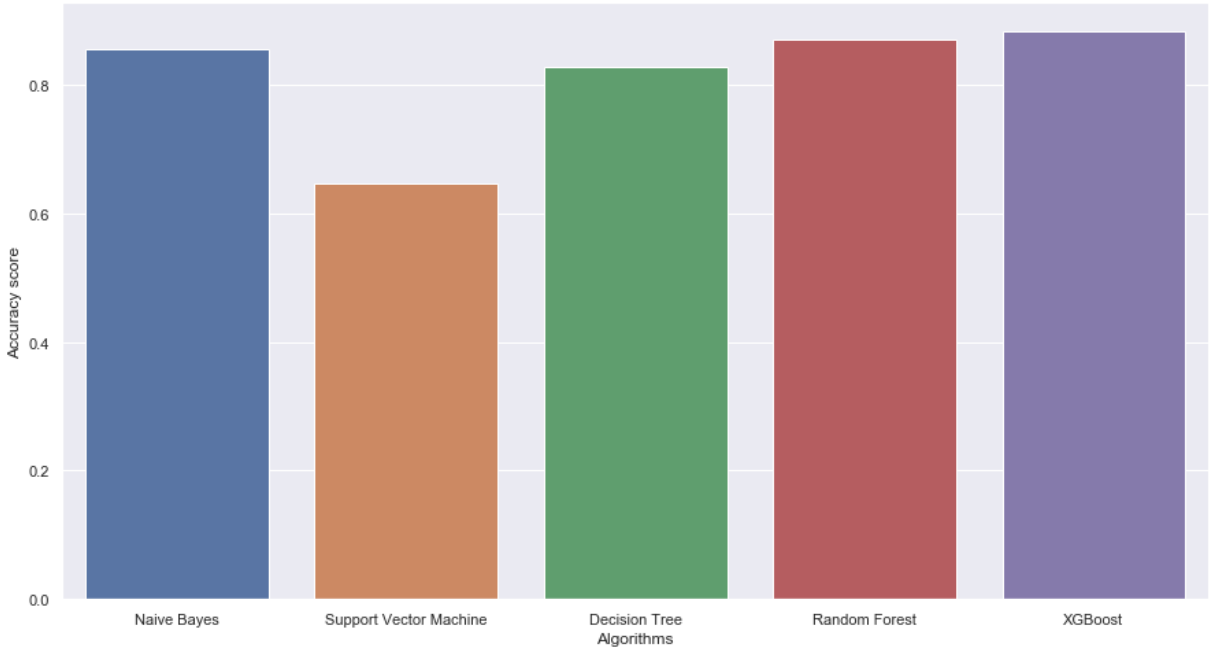
In [102]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.lineplot(algorithms,scores)
```

Out[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a51900ac8>
```



In [103]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms,scores)
```

Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a52213f60>
```



splitdata

In [79]:
```
from sklearn.model_selection import train_test_split
feature_col_names = ['Level_of_Hemoglobin', 'Genetic_Pedigree_Coefficient','Age'
predicted_class_names = ['Blood_Pressure_Abnormality']
X= df[feature_col_names].values
y= df[predicted_class_names].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_
```

NAIVE BAYES

In [80]:
```
#Training the model
from sklearn.naive_bayes import GaussianNB

# create Gaussian Naive Bayes model object and train it with the data
nb_model = GaussianNB()

nb_model.fit(X_train, y_train.ravel())
```

Out[80]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [81]:
```
# predict values using the testing data
nb_predict_test = nb_model.predict(X_test)

# Accuracy
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
Accuracy on Test data: 0.8562
```

In [82]:
```
nb_predict_test = nb_model.predict(X_test)

# Accuracy
score_nb=metrics.accuracy_score(y_test,nb_predict_test)
print(score_nb)
#print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
print("Accuracy on Test data: {0:.4f}".format(metrics.accuracy_score(y_test, nb_
print()
```

```
0.85625
Accuracy on Test data: 0.8562
```

In [83]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,nb_predict_test)
```

Out[83]:
```
array([[368,  58],
       [ 57, 317]], dtype=int64)
```

In [84]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,nb_predict_test))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86       426
           1       0.85      0.85      0.85       374

   micro avg       0.86      0.86      0.86       800
   macro avg       0.86      0.86      0.86       800
weighted avg       0.86      0.86      0.86       800
```

SUPPORT VECTOR MACHINE

In [85]:
```
from sklearn.svm import SVC
model=SVC(gamma='auto')
model.fit(X_train,y_train.ravel())
```

Out[85]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [86]:
```
prediction_test=model.predict(X_test)
```

In [87]:
```
score_svm=metrics.accuracy_score(y_test,prediction_test)
print(score_svm)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.64625
Accuracy= 0.64625
```

In [88]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,prediction_test)
```

Out[88]:
```
array([[267, 159],
       [124, 250]], dtype=int64)
```

DECISION TREE

In [89]:
```
from sklearn import tree
model=tree.DecisionTreeClassifier()
```

In [90]:
```
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
```

In [91]:
```
score_dt=metrics.accuracy_score(y_test,prediction_test)
print(score_dt)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.8275
Accuracy= 0.8275
```

In [92]:
```
print("accuracy on training data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
accuracy on training data :0.8275
```

RANDOM FOREST

In [93]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
score_rf=metrics.accuracy_score(y_test,prediction_test)
print(score_rf)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
```

```
0.87125
Accuracy= 0.87125
```

GRADIENT BOOSTING

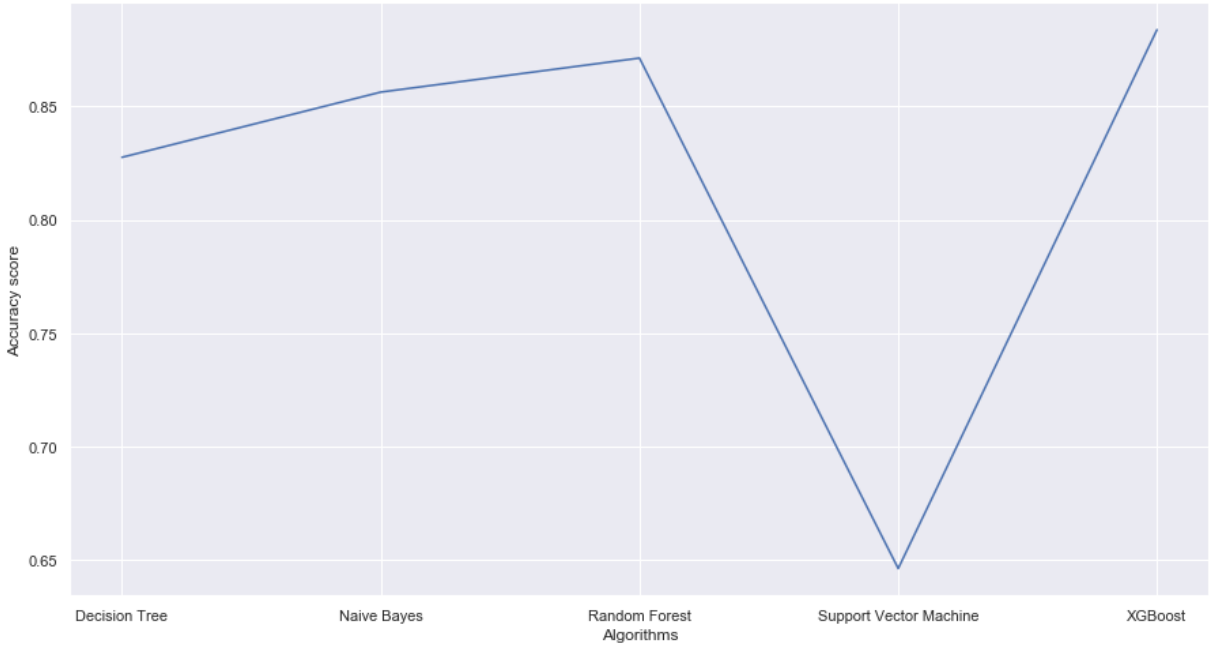
In [94]:
```
from sklearn.ensemble import GradientBoostingClassifier
model=GradientBoostingClassifier(n_estimators = 10,random_state=30)
model.fit(X_train,y_train.ravel())
prediction_test=model.predict(X_test)
from sklearn import metrics
score_gb=metrics.accuracy_score(y_test,prediction_test)
print(score_gb)
print("Accuracy=",metrics.accuracy_score(y_test,prediction_test))
print("accuracy on training data :{0:.4f}".format(model.score(X_test,y_test)))
```

```
0.88375
Accuracy= 0.88375
accuracy on training data :0.8838
```

In [95]:
```
scores = [score_nb,score_svm,score_dt,score_rf,score_gb]
algorithms = ["Naive Bayes","Support Vector Machine","Decision Tree","Random For
```

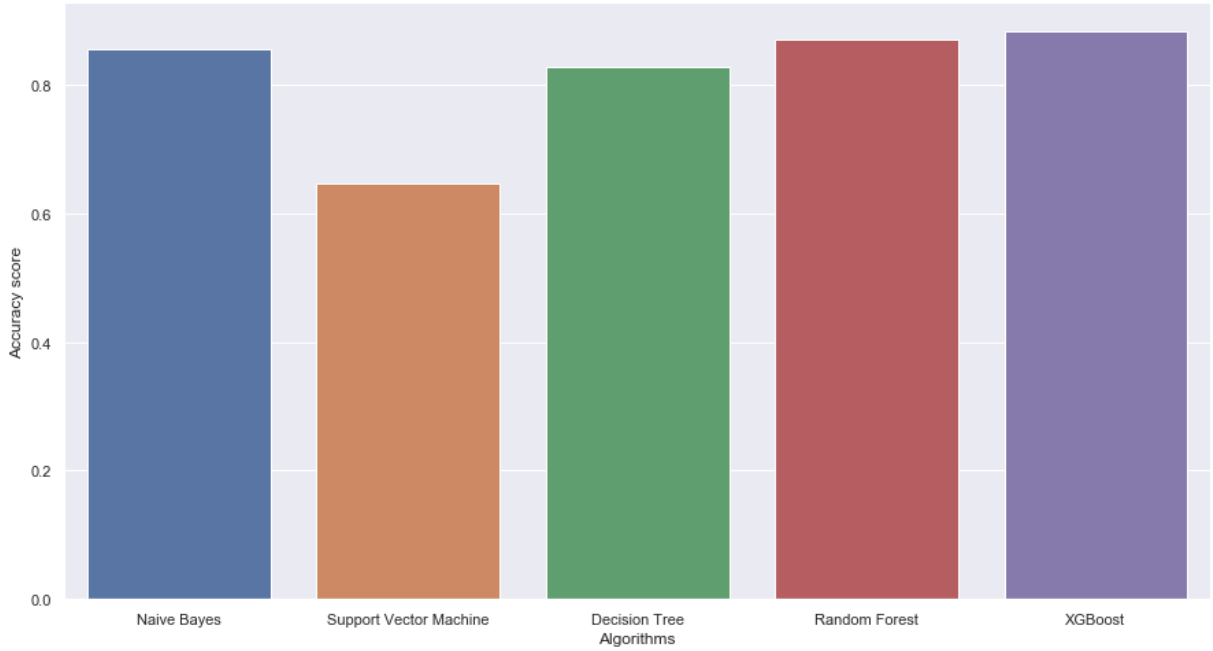
In [96]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.lineplot(algorithms,scores)
```

Out[96]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23a519d67f0>
```



In [97]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")

sns.barplot(algorithms,scores)
```

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x23a501a3dd8>



In [ ]: