Name:- Jalendar Reddy Maligireddy

ID:- 11511290

RANDOM FOREST

1. About the model

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems.

Two types of methods:

1. Bagging– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.
2. Boosting– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST

1. Dataset

The data set containing descriptive attributes of digitized images of a process known as, fine needle aspirate (FNA) of breast mass.

We have a total of 29 features that were computed for each cell nucleus with an ID Number and the Diagnosis (Later converted to binary representations: Malignant = 1, Benign = 0).

In [1]:
```
#importing the pandas package
import pandas as pd
```

Loading the dataset with first column as index, named as dataframe

In [2]:
```
#Loading the dataset to dataframe
#By using pandas reading csv file
#index as first rows in dataset
dataframe = pd.read_csv("data-breastCancer.csv", index_col=0)
```

In [3]:
```
#using the head function getting first 5 rows in the dataframe
dataframe.head()
```

Out[3]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
842302	M	17.99	10.38	122.80	1001.0	0.11840	
842517	M	20.57	17.77	132.90	1326.0	0.08474	
84300903	M	19.69	21.25	130.00	1203.0	0.10960	
84348301	M	11.42	20.38	77.58	386.1	0.14250	
84358402	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 32 columns

In [4]:
```
#using the tail function getting last 5 rows in the dataframe
dataframe.tail()
```

Out[4]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compac
926424	M	21.56	22.39	142.00	1479.0	0.11100	
926682	M	20.13	28.25	131.20	1261.0	0.09780	
926954	M	16.60	28.08	108.30	858.1	0.08455	
927241	M	20.60	29.33	140.10	1265.0	0.11780	
92751	B	7.76	24.54	47.92	181.0	0.05263	

5 rows × 32 columns

1. Data pre-processing

In [5]:
```
#info gives details about dataset
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 842302 to 92751
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   diagnosis               569 non-null    object
 1   radius_mean             569 non-null    float64
 2   texture_mean            569 non-null    float64
 3   perimeter_mean          569 non-null    float64
 4   area_mean               569 non-null    float64
 5   smoothness_mean         569 non-null    float64
 6   compactness_mean        569 non-null    float64
 7   concavity_mean          569 non-null    float64
```

```
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
 31  Unnamed: 32              0 non-null      float64
dtypes: float64(31), object(1)
memory usage: 146.7+ KB
```

Finding the missing values in the dataset

In [6]:
```
dataframe.isnull().sum()
```

Out[6]:
```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
```

```
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

There are 569 missing values in the unamed: 32 ccolumn

To get the number of rows and columns in the dataset, by using shape function

In [7]:
```
#to get number of rows and columns using shape
dataframe.shape
```

Out[7]:
```
(569, 32)
```

To get the detail of the dataset, by using the describe function

from the describe function, we can have the

1. Nummber of values in column
2. Mean of the column
3. Standard deviation of the column
4. minimum & maximum of the column
5. 25%,50%,75% of the column

In [8]:
```
#to detail explain of dataset using describe
dataframe.describe()
```

Out[8]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400

8 rows × 31 columns

In [9]:
```
#getting the column names in the dataset
dataframe.columns
```

Out[9]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
```

```
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

To get number of columns in the dataframe

In [10]:
```
from numpy.ma.core import count
count(dataframe.columns)
```

Out[10]:
```
32
```

dropping the Unnamed: 32 column, and the new data is named as df

in the df we are using id as index

In [11]:
```
# dropping the unamed : 32 column and assigned the dataset to df
df= dataframe.drop(['Unnamed: 32'],axis=1)
```

In [12]:
```
df
```

Out[12]:

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
842302	M	17.99	10.38	122.80	1001.0	0.11840	
842517	M	20.57	17.77	132.90	1326.0	0.08474	
84300903	M	19.69	21.25	130.00	1203.0	0.10960	
84348301	M	11.42	20.38	77.58	386.1	0.14250	
84358402	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
926424	M	21.56	22.39	142.00	1479.0	0.11100	
926682	M	20.13	28.25	131.20	1261.0	0.09780	
926954	M	16.60	28.08	108.30	858.1	0.08455	
927241	M	20.60	29.33	140.10	1265.0	0.11780	
92751	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

In [13]:
```
df.shape
```

Out[13]:
```
(569, 31)
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

checking each column datatype

In [15]:
```
#getting the data type of all columns
df.dtypes
```

Out[15]:
```
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
dtype: object
```

converting object datatype into int datatype

In [16]:
```
# converint object type into int type (0,1)
# where in diagnosis M as 1 & B as 0
df['diagnosis_value'] = df['diagnosis'].map({'M':1, 'B':0})
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'diagnosis_value'],
      dtype='object')
```

dropping the diagnosis column, which is object type where we assigned new column diagnosis_value using diagnosis column to int type from diagnosis column M as 1 & B as 0

In [18]:
```
# dropping the object data type & assigned the data set to df
df = df.drop(['diagnosis'],axis=1)
```

In [19]:
```
df.dtypes
```

Out[19]:
```
radius_mean               float64
texture_mean              float64
perimeter_mean            float64
area_mean                 float64
smoothness_mean           float64
compactness_mean          float64
concavity_mean            float64
concave points_mean       float64
symmetry_mean             float64
fractal_dimension_mean    float64
radius_se                 float64
texture_se                float64
perimeter_se              float64
area_se                   float64
smoothness_se             float64
compactness_se            float64
concavity_se              float64
concave points_se         float64
symmetry_se               float64
fractal_dimension_se      float64
radius_worst              float64
texture_worst             float64
perimeter_worst           float64
area_worst                float64
smoothness_worst          float64
compactness_worst         float64
concavity_worst           float64
concave points_worst      float64
symmetry_worst            float64
fractal_dimension_worst   float64
diagnosis_value             int64
dtype: object
```

In [20]:
```
df.shape
```

```
(569, 31)
```

Out[20]:

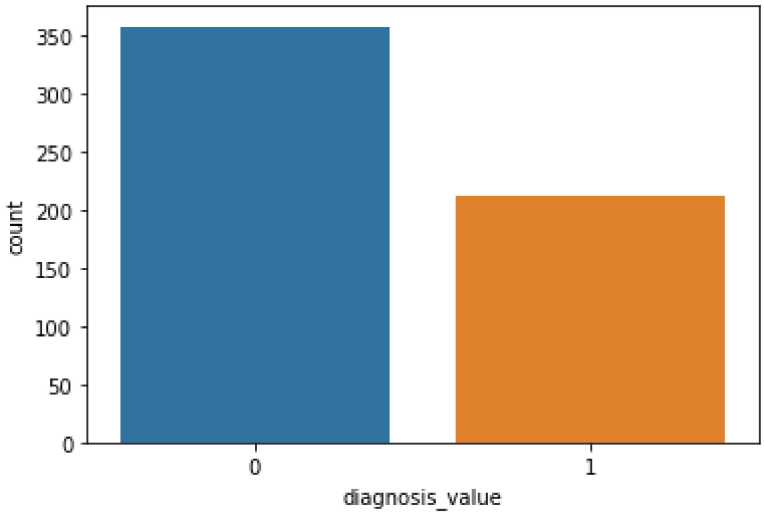
In [21]:
```
#checking any null values in the dataset
df.isnull().sum()
```

Out[21]:
```
radius_mean               0
texture_mean              0
perimeter_mean            0
area_mean                 0
smoothness_mean           0
compactness_mean          0
concavity_mean            0
concave points_mean       0
symmetry_mean             0
fractal_dimension_mean    0
radius_se                 0
texture_se                0
perimeter_se              0
area_se                   0
smoothness_se             0
compactness_se            0
concavity_se              0
concave points_se         0
symmetry_se               0
fractal_dimension_se      0
radius_worst              0
texture_worst             0
perimeter_worst           0
area_worst                0
smoothness_worst          0
compactness_worst         0
concavity_worst           0
concave points_worst      0
symmetry_worst            0
fractal_dimension_worst   0
diagnosis_value           0
dtype: int64
```

1. Data visualizations

In [22]:
```
#creating a plot for the number of value
import seaborn as sns
sns.countplot(x='diagnosis_value', data=df)
#getting number of different values count
print("Number of 0's & 1's in the diagnosis_value :\n", df['diagnosis_value'].value_cou
```

```
Number of 0's & 1's in the diagnosis_value :
 0    357
1    212
Name: diagnosis_value, dtype: int64
```



The "diagnosis_value" contains binary values: M as 1 (malignant) - 212 and B as 0 (benight) - 357

Declare feature vector and target variable

In [23]:
```
# Splitting data into x & y
# X as all the dataset columns except 'diagnosis_value'
# y as only 'diagnosis_value'
X = df.drop(labels='diagnosis_value', axis=1)
y = df['diagnosis_value']
```

1. Splitting the data

In [24]:
```
#importing the train_test_split model
from sklearn.model_selection import train_test_split
```

In [25]:
```
#creating test and train data using train_test_split model with train size 80% & test s
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

In [26]:
```
# getting the shapes of train & test
print ('Shapes of X_train, y_train: ', X_train.shape, y_train.shape)
print ('Shapes of X_test, y_test: ', X_test.shape, y_test.shape)
```

```
Shapes of X_train, y_train:  (455, 30) (455,)
Shapes of X_test, y_test:  (114, 30) (114,)
```

1. Applying the model

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
```

In [28]:
```
rand_frst_clasfr = RandomForestClassifier(random_state=42)
np.random.seed(42)
```

K-fold cross validation:

K-fold cv splits training data into K folds and iteratively fits the model k times, training on k-1 of the folds and assessing on the kth fold each time. Finally, the model's final validation metrics are calculated by averaging the performance on each of the folds.

In [29]:
```
from sklearn.model_selection import cross_validate
cv = cross_validate (rand_frst_clasfr, X_train, y_train, cv = 10)
```

In [30]:
```
print("Accuracy score of 10-fold cross validation: ", cv['test_score'])
print("Accuracy mean score of CV: ", round(cv['test_score'].mean(),3))
```

```
Accuracy score of 10-fold cross validation:  [1.         0.95652174 0.97826087 0.9130434
8 0.97826087 0.97777778
 0.95555556 0.93333333 0.97777778 0.95555556]
Accuracy mean score of CV:  0.963
```

With an accuracy rating of 0.96, the underlying version done brilliantly at the education set. However, through adjusting the hyperparameters, we will nonetheless enhance performance.

Before fitting a rf model we need to tune the parameters to get the best model. Below parameters are tuned in this project.

1. max_depth
2. max_features
3. criterion
4. bootstrap

In [31]:
```
prm_dist = {'max_depth': [2, 3, 4],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2', None],
              'criterion': ['gini', 'entropy']}

print(prm_dist)
```

```
{'max_depth': [2, 3, 4], 'bootstrap': [True, False], 'max_features': ['auto', 'sqrt', 'l
og2', None], 'criterion': ['gini', 'entropy']}
```

From the results :

max_depth: [2, 3, 4]

bootstrap: [True, False]

max_features: ['auto', 'sqrt', 'log2', None]

criterion: ['gini', 'entropy']

Before tuning the parameters, the below values are passed to the random forest classifier and got the best parameters using best_params of GridSearchCV from the sklearn.

In [32]:
```
from sklearn.model_selection import GridSearchCV
```

In [33]:

```
cv_rand_frst = GridSearchCV(rand_frst_clasfr , cv = 10,
                     param_grid=prm_dist,
                     n_jobs = 3)
```

In [34]:
```
cv_rand_frst.fit(X_train, y_train)
```

Out[34]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=42), n_jobs=3,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 3, 4],
                         'max_features': ['auto', 'sqrt', 'log2', None]})
```

In [35]:
```
print('Best Parameters using grid search: {0}'.format(cv_rand_frst.best_params_))
```

```
Best Parameters using grid search: {'bootstrap': True, 'criterion': 'entropy', 'max_dept
h': 4, 'max_features': 'log2'}
```

from the result the best parameters are

bootstrap: True

criterion: entropy

max_depth: 4

max_features: log2

Finally, the random forest model is fit by setting the parameter values after tuning.

Bagging Classifier

Bagging is an ensemble approach that fits multiple models to different subsets of a training dataset before combining the predictions of all the models. Random forest is a bagging extension that randomly selects a subset of the features used in each data sample. Both bagging forests and random forests have proven successful in a variety of predictive modeling problems.

While effective, they are not suitable for classification problems with a distorted class distribution. Nonetheless, some changes to the algorithms have been proposed, adjusting their behavior and making them more suitable for serious class imbalances.

In [36]:
```
from sklearn.ensemble import BaggingClassifier
```

In [37]:
```
bagging_clssfr = BaggingClassifier()
```

In [38]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [39]:
```
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
```

In [40]:
```
from sklearn.model_selection import cross_val_score
```

In [41]:
```
scrs = cross_val_score(bagging_clssfr, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
```

In [42]:
```
from numpy import mean
```

In [43]:
```
print('BaggingClassifier model Mean ROC AUC : %.3f' % mean(scrs))
```

```
BaggingClassifier model Mean ROC AUC : 0.983
```

In [44]:
```
cv = cross_validate (bagging_clssfr, X_train, y_train, cv = 10)
```

In [45]:
```
print("BaggingClassifier model Accuracy score of 10-fold cross validation: ", cv['test_
print("BaggingClassifier model Accuracy mean score of CV: ", cv['test_score'].mean())
```

```
BaggingClassifier model Accuracy score of 10-fold cross validation:  [0.95652174 0.93478
261 0.97826087 0.89130435 0.97826087 0.97777778
 1.         0.95555556 0.97777778 0.93333333]
BaggingClassifier model Accuracy mean score of CV:  0.9583574879227053
```

In [46]:
```
bagging_clssfr.fit(X_train, y_train)
```

Out[46]:
```
BaggingClassifier()
```

1. Prediction results and test scores

In [47]:
```
# making the prediction on the test set by using bagging classifer
predict_data = bagging_clssfr.predict(X_test)

# the estimating probability of the test set
predict_probability = bagging_clssfr.predict_proba (X_test)
```

In [48]:
```
#printing the predicted probability for class_0 & class_1
print('class_0', 'class_1')
print(predict_probability[:10])
```

```
class_0 class_1
[[1.  0. ]
 [0.  1. ]
 [0.  1. ]
 [1.  0. ]
 [1.  0. ]
 [0.  1. ]
 [0.  1. ]
 [0.1 0.9]
 [0.3 0.7]
 [1.  0. ]]
```

In [49]:
```
# finding the acurracy scores for the tet set
```

```
print('BaggingClassifier model Accuracy of the selected model in the test set: {:.3f}'.
```

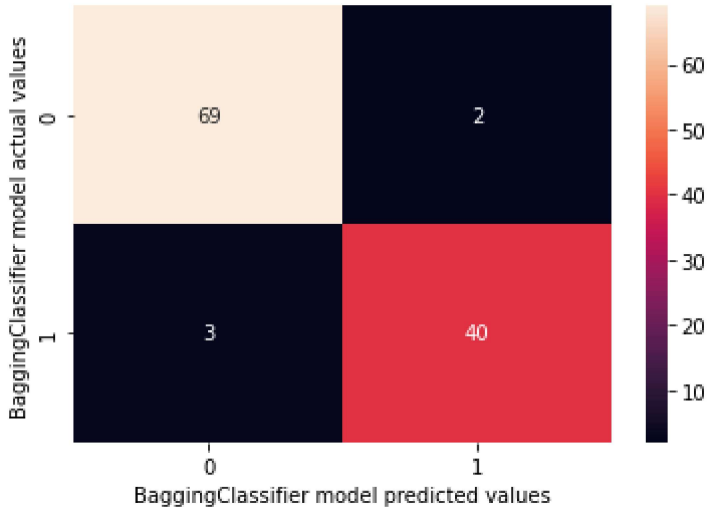
```
BaggingClassifier model Accuracy of the selected model in the test set: 0.956
```

In [50]:
```
import matplotlib.pyplot as plyt
```

In [51]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, predict_data )
```

In [52]:
```
# visualizing confusion matrix
sns.heatmap(conf_matrix, annot = True)
plyt.xlabel ('BaggingClassifier model predicted values')
plyt.ylabel ('BaggingClassifier model actual values')
```

Out[52]:
```
Text(33.0, 0.5, 'BaggingClassifier model actual values')
```



From the above confusion matrix below results can be inferred

In the test data set actually there are total 114 patient records among which 71 are actual 0s and 43 are actual 1s

The random forest model we developed has predicted 1 record as 1s from out of 71 0s and 5 records as 0s out of 43 1s.

1. Metrics

In [53]:
```
from sklearn.metrics import roc_curve, roc_auc_score
# the probability of the class_1 on the test set
predicted_probability_class_1 = predict_probability[:,1]

# True Positive Rate  and False Positive Rate
false_post_rate, true_post_rate, threshold = roc_curve (y_test,predicted_probability_cl

# the Area Under the ROC Curve (roc_auc)
roc_auc_score = roc_auc_score (y_test,predicted_probability_class_1)
print("BaggingClassifier model roc auc score : {:.3f}".format(roc_auc_score))
```

```
BaggingClassifier model roc auc score : 0.978
```

A ROC (Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model at all classification thresholds. The higher the AUC, the better the model behaves. The AUC is expected to be more than 0.5 above the upper left part of the baseline.

TPR is the probability that an right positive will test positive.

```
TPR = TP/P = TP/(TP+FN)
```

FPR is the model wrongly predicted the positive class

```
FPR = FP/N = FP/ (FP+TN)
```

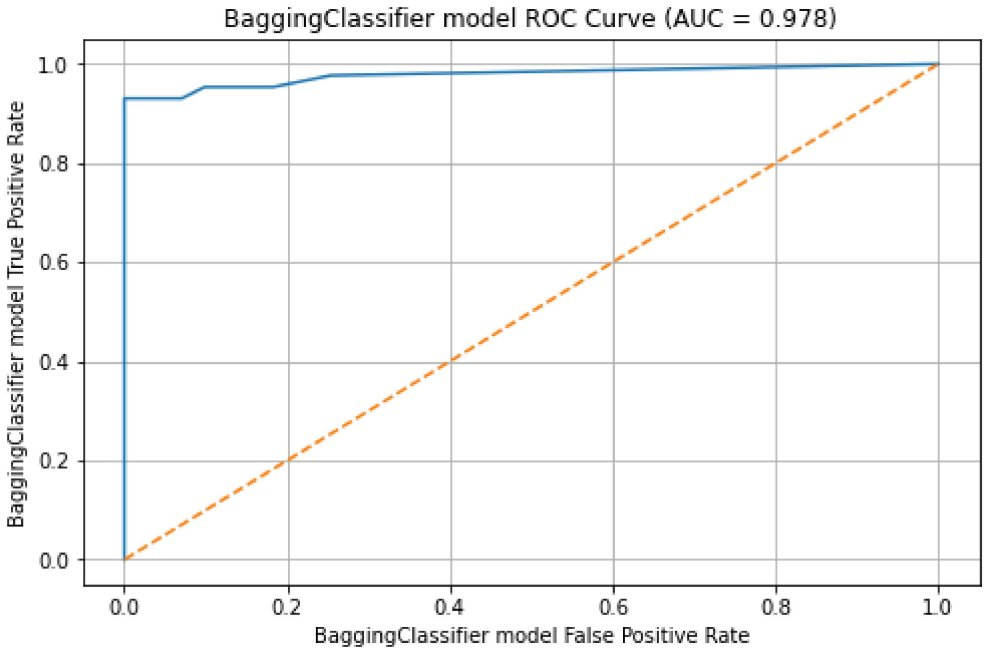
In [54]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, predict_data))
```

```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

In [55]:
```
# Visualizing the ROC Curve
plyt.subplots(figsize = (8,5))
plyt.plot( false_post_rate, true_post_rate, label='BaggingClassifier model Test AUC: %0
plyt.plot([0,1], '--')
plyt.xlabel('BaggingClassifier model False Positive Rate')
plyt.ylabel('BaggingClassifier model True Positive Rate')
plyt.title ("BaggingClassifier model ROC Curve (AUC = {:.3f})".format(roc_auc_score))
plyt.grid()
plyt.show()
```



From the above ROC curve, The accuracy of the BaggingClassifier model is 0.97%.

In [ ]:

In [ ]:

In [ ]:

In [56]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [57]:
```
random_grid = {'n_estimators' :[int(x) for x in np.linspace(start = 100, stop = 1000, n
               'max_features': ['auto', 'sqrt', 'log2'],
               'max_depth': [int(x) for x in np.linspace(start = 3, stop = 36, num=33, en
               'min_samples_split': [5, 10, 15],
               'min_samples_leaf': [3, 4, 5],
               'bootstrap': ['True']}
print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features':
['auto', 'sqrt', 'log2'], 'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36], 'min_sample
s_split': [5, 10, 15], 'min_samples_leaf': [3, 4, 5], 'bootstrap': ['True']}
```

In [58]:
```
random_grid ={'max_depth': [2, 3, 4],
              'bootstrap': [True, False],
              'max_features': ['auto', 'sqrt', 'log2', None],
              'criterion': ['gini', 'entropy']}
print(random_grid)
```

```
{'max_depth': [2, 3, 4], 'bootstrap': [True, False], 'max_features': ['auto', 'sqrt', 'l
og2', None], 'criterion': ['gini', 'entropy']}
```

In [59]:
```
# Creating a base randomforest model for tuning
tuning_randfrst_clssfr = RandomForestClassifier(random_state=42)

# Create a randomized search cross validation model for searching for the best hyperpar
random_search_rf = RandomizedSearchCV (estimator=tuning_randfrst_clssfr, param_distribu

# fit the randomized search CV model into the training set
random_search_rf.fit(X_train, y_train)

# Print the best parameters
print(random_search_rf.best_params_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:285: UserW
arning: The total space of parameters 48 is smaller than n_iter=100. Running 48 iteratio
ns. For exhaustive searches, use GridSearchCV.
  warnings.warn(
{'max_features': 'log2', 'max_depth': 4, 'criterion': 'entropy', 'bootstrap': True}
```

In [60]:
```
# Creating a tuned random forest model with the best parameters choosen by  the cv rand
tuned_randfrst_clssfr = RandomForestClassifier (n_estimators = 100, min_samples_split =

# Validating the model using k-fold cross validation (k=10)
tuned_cv = cross_validate (tuned_randfrst_clssfr, X_train, y_train, cv = 10)
print("RandomForestClassifier model Accuracy score: ", tuned_cv['test_score'])
print("RandomForestClassifier model Accuracy mean score of CV: ", tuned_cv['test_score'
```

```
RandomForestClassifier model Accuracy score:  [1.         0.95652174 0.95652174 0.913043
48 0.97826087 0.97777778
 0.93333333 0.93333333 0.97777778 0.95555556]
RandomForestClassifier model Accuracy mean score of CV:  0.9582125603864734
```

In [61]:
```
# Fit the selected model to the training set
tuned_randfrst_clssfr.fit(X_train, y_train)
```

Out[61]:
```
RandomForestClassifier(max_depth=26, min_samples_leaf=3, min_samples_split=5,
                       random_state=42)
```

In [62]:
```
# Applying the selected model to make prediction on the test set
pred = tuned_randfrst_clssfr.predict(X_test)

# Observing the estimate probability of classess in the test set
pred_prob = tuned_randfrst_clssfr.predict_proba (X_test)
print ('class_0', 'class_1')
print(pred_prob[:10])
```

```
class_0 class_1
[[0.97733333 0.02266667]
 [0.005      0.995     ]
 [0.01       0.99      ]
 [1.         0.        ]
 [1.         0.        ]
 [0.00533333 0.99466667]
 [0.         1.        ]
 [0.10590476 0.89409524]
 [0.28519048 0.71480952]
 [0.97100794 0.02899206]]
```

In [63]:
```
# finding the acurracy scores for the tet set
print('RandomForestClassifier model Accuracy of the selected model in the test set: {:.
```

```
RandomForestClassifier model Accuracy of the selected model in the test set: 0.96
```

In [64]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, pred )
# visualizing confusion matrix
sns.heatmap(conf_matrix, annot = True)
plyt.xlabel ('RandomForestClassifier model predicted values')
plyt.ylabel ('RandomForestClassifier model actual values')
```

Out[64]:
```
Text(33.0, 0.5, 'RandomForestClassifier model actual values')
```

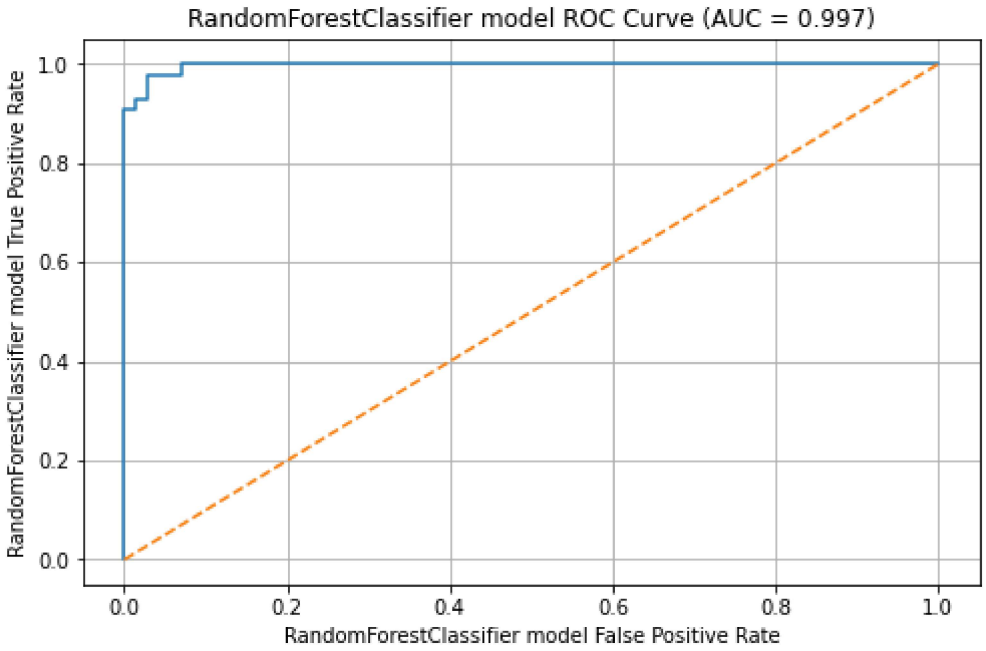
In [65]:
```
from sklearn.metrics import roc_curve, roc_auc_score
#the probability of the class_1 on the test set
pred_prob_c1 = pred_prob[:,1]

# True Positive Rate and False Positive Rate
false_post_rate, true_post_rate, threshold = roc_curve (y_test,pred_prob_c1, pos_label

# the Area Under the ROC Curve
roc_auc_score = roc_auc_score (y_test,pred_prob_c1)
print("RandomForestClassifier model Roc AUC Score : ",round(roc_auc_score,2))
```

```
RandomForestClassifier model Roc AUC Score :  1.0
```

In [66]:
```
# Visualizing the ROC Curve
plyt.subplots(figsize = (8,5))
plyt.plot( false_post_rate, true_post_rate, label='Test AUC: %0.3f'%roc_auc_score)
plyt.plot([0,1], '--')
plyt.xlabel('RandomForestClassifier model False Positive Rate')
plyt.ylabel('RandomForestClassifier model True Positive Rate')
plyt.title ("RandomForestClassifier model ROC Curve (AUC = {:.3f})".format(roc_auc_scor
plyt.grid()
plyt.show();
```



Conclusion

From the above ROC curve, The accuracy of the RandomForestClassifier model is 0.99%.

From the above ROC curve, The accuracy of the BaggingClassifier model is 0.97%.

from the both, we can conclude that RandomForestClassifier model is best with 0.99%

In [ ]: