Junior Data Scientist Challenge 2: Random Forest Algorithm

The Challenge:

For this Challenge, you will be using the Random Forest algorithm to create a model that can determine whether or not a banknote is authentic based on the given attributes.

Use this dataset: https://drive.google.com/file/d/13nw-uRXPY8XIZQxKRNZ3yYlho-CYm_Qt/view

Load the given data and prepare it for training Scale the data to normalize the range of features Train your random forests to perform the classification; experiment with the number of trees in your forest Evaluate the accuracy of your algorithm Create a visual representation of your results

Resources

https://builtin.com/data-science/random-forest-algorithm

Assessment Criteria

Your challenge will be assessed based on the following:

1 mark for loading and preparing the data 2 marks for scaling and normalizing the data 3 marks for training and testing your random forest 2 marks for the accuracy of your algorithms prediction and your explanation of the accuracy 1 mark for the visual representation of your results 1 bonus mark for extra work created

Heart Disease

In [1]:

```
# importing Libariries
# pandas
import pandas as pd

# numpy
import numpy as np

# finding mean (average) of rental rates
import statistics as stat

# matplotlib library for visualization
import matplotlib.pyplot as plt

#import seaborn library for visualization
import seaborn as sns
#%matplotlib inline
```

Step1) Load the given data and prepare it for training

In [2]:

```
# Loading data
df = pd.read_csv('bill_authentication.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Variance    0
Skewness    0
Curtosis    0
Entropy     0
Class       0
dtype: int64
```

In [5]:
```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
Variance	1372.0	0.433735	2.842763	-7.0421	-1.773000	0.49618	2.821475	6.8248
Skewness	1372.0	1.922353	5.869047	-13.7731	-1.708200	2.31965	6.814625	12.9516
Curtosis	1372.0	1.397627	4.310030	-5.2861	-1.574975	0.61663	3.179250	17.9274
Entropy	1372.0	-1.191657	2.101013	-8.5482	-2.413450	-0.58665	0.394810	2.4495
Class	1372.0	0.444606	0.497103	0.0000	0.000000	0.00000	1.000000	1.0000

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Variance  1372 non-null   float64
 1   Skewness  1372 non-null   float64
 2   Curtosis  1372 non-null   float64
 3   Entropy   1372 non-null   float64
 4   Class     1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

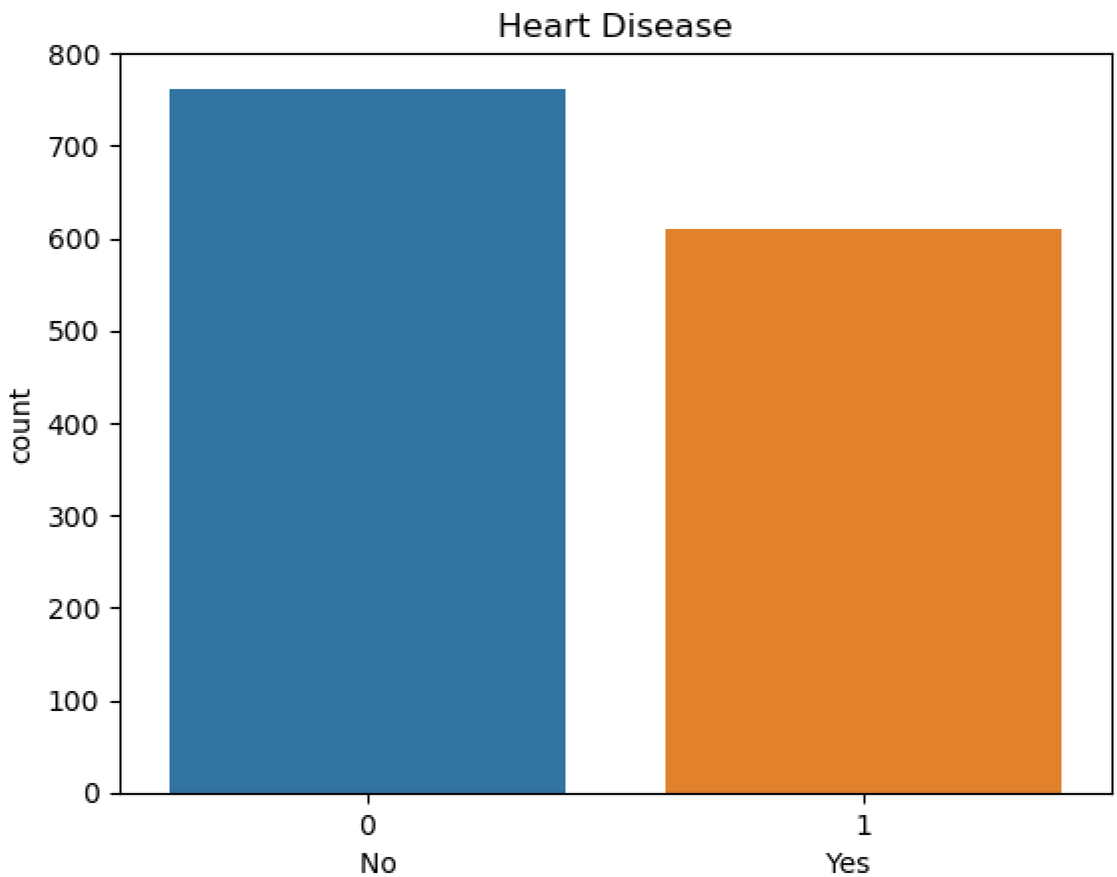
In [7]:
```
df.duplicated()
```

Out[7]:
```
0       False
1       False
2       False
3       False
4       False
        ...  
1367    False
1368    False
1369    False
1370    False
1371    False
Length: 1372, dtype: bool
```

In [8]:
```python
# finding how many data have heart disease (1) and
# how many data do not have heart disease (0)
df.Class.value_counts()
```

Out[8]:
```
0    762
1    610
Name: Class, dtype: int64
```

In [9]:
```python
sns.countplot(x=df.Class)
plt.title('Heart Disease')
plt.xlabel('No                                                  Yes')
plt.show()
```



In [10]:
```python
#prepare data for training

feature = df.drop('Class', axis=1)
target = df.Class
```

Step2) Scale the data to normalize the range of features

2 types 1) Normalization => minimum value=0 and maximum value=1 and rest values remain in the range, good to use when you want to normalize outliers, When max and min value. MinMaxScaler() 2) Standarization => variance=1 and std= 0, outliers remain the same, mostly used in machine learning model. StandardScaler() : Standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as: z = (x - u) / s

In [11]:
```python
from sklearn.preprocessing import StandardScaler
```

In [12]:
```python
scaler = StandardScaler()
scaled_feature = scaler.fit_transform(feature)
```

In [13]:
```python
from sklearn.model_selection import train_test_split

#splitting the dataset into training and test
X_train,X_test,y_train,y_test = train_test_split(scaled_feature,target,test_size=0.33)
```

In [14]:
```python
# getting size of following
X_train.shape ,X_test.shape ,y_train.shape ,y_test.shape
```

Out[14]:
```
((919, 4), (453, 4), (919,), (453,))
```

Step3) Train your random forests to perform the classification; experiment with the number of trees in your forest

In [15]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
```

mostly in practice 100 trees are used in random forest

In [16]:
```python
no_tree = [2,25,50,100,250,500]
accuracy_list=[]
for i in no_tree:
    clf = RandomForestClassifier(n_estimators=i,random_state=100)
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    accuracy_list.append(accuracy)
```

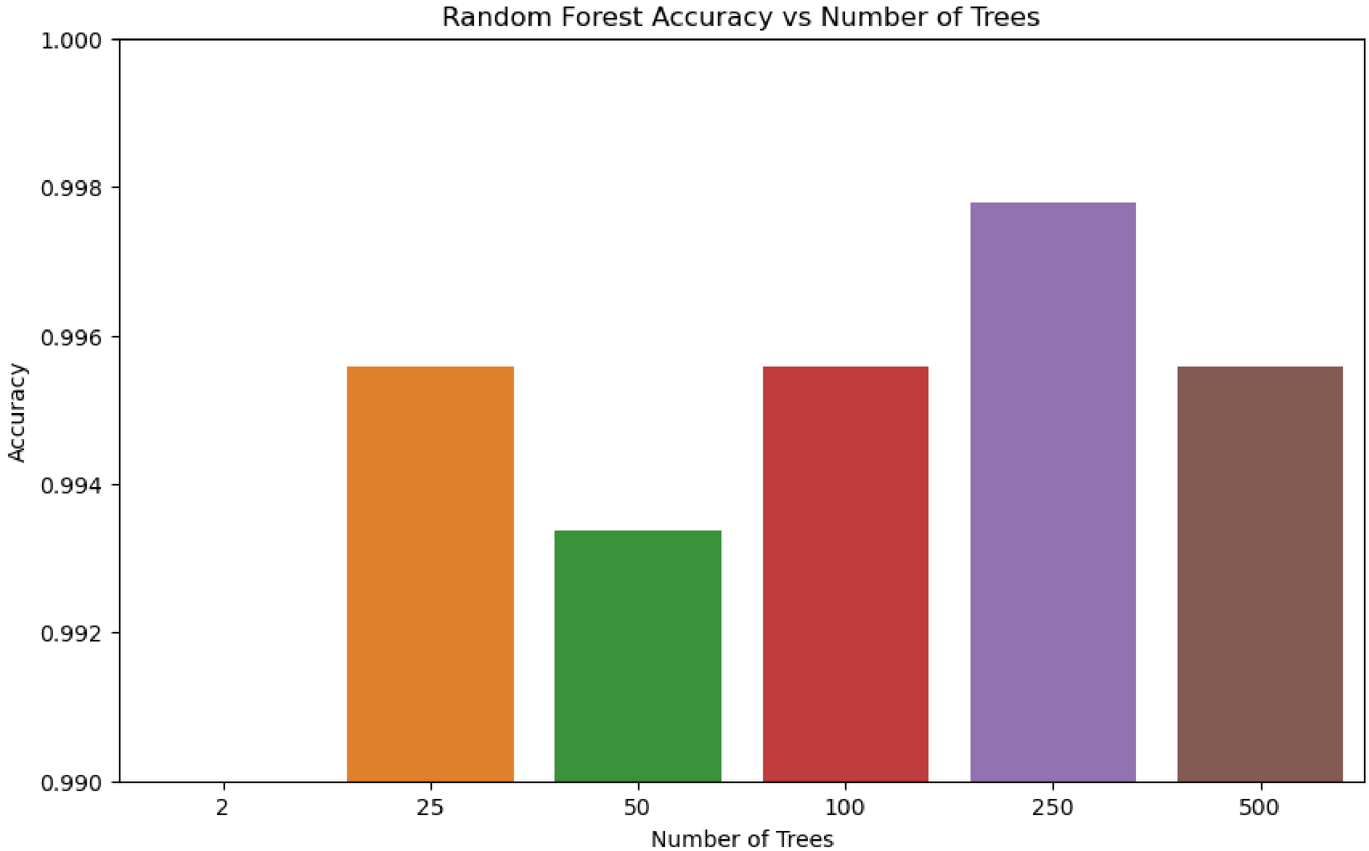
In [17]:
```python
#Evaluating the accuracy of algorithm with respect to number of decision trees
j=0
for i in no_tree:
    print('The accuracy score for',i, 'tree is', accuracy_list[j])
    j +=1
```

```
The accuracy score for 2 tree is 0.9624724061810155
The accuracy score for 25 tree is 0.9955849889624724
The accuracy score for 50 tree is 0.9933774834437086
The accuracy score for 100 tree is 0.9955849889624724
The accuracy score for 250 tree is 0.9977924944812362
The accuracy score for 500 tree is 0.9955849889624724
```

In [18]:
```python
accuracy_list
```

Out[18]:
```
[0.9624724061810155,
 0.9955849889624724,
 0.9933774834437086,
 0.9955849889624724,
 0.9977924944812362,
 0.9955849889624724]
```

In [19]:
```python
plt.figure(figsize=(10, 6))
sns.barplot(x=no_tree,y=accuracy_list)
plt.title('Random Forest Accuracy vs Number of Trees')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.ylim(0.99, 1.0)  # Set y-axis limit for better visualization of differences plt.grid(axis='y
plt.show()
```



Step4) Evaluate the (overall) accuracy of your algorithm

Performance Metrics

In [20]:
```
from sklearn.metrics import classification_report,roc_curve, auc
```

In [21]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       253
           1       0.99      0.99      0.99       200

    accuracy                           1.00       453
   macro avg       1.00      1.00      1.00       453
weighted avg       1.00      1.00      1.00       453
```

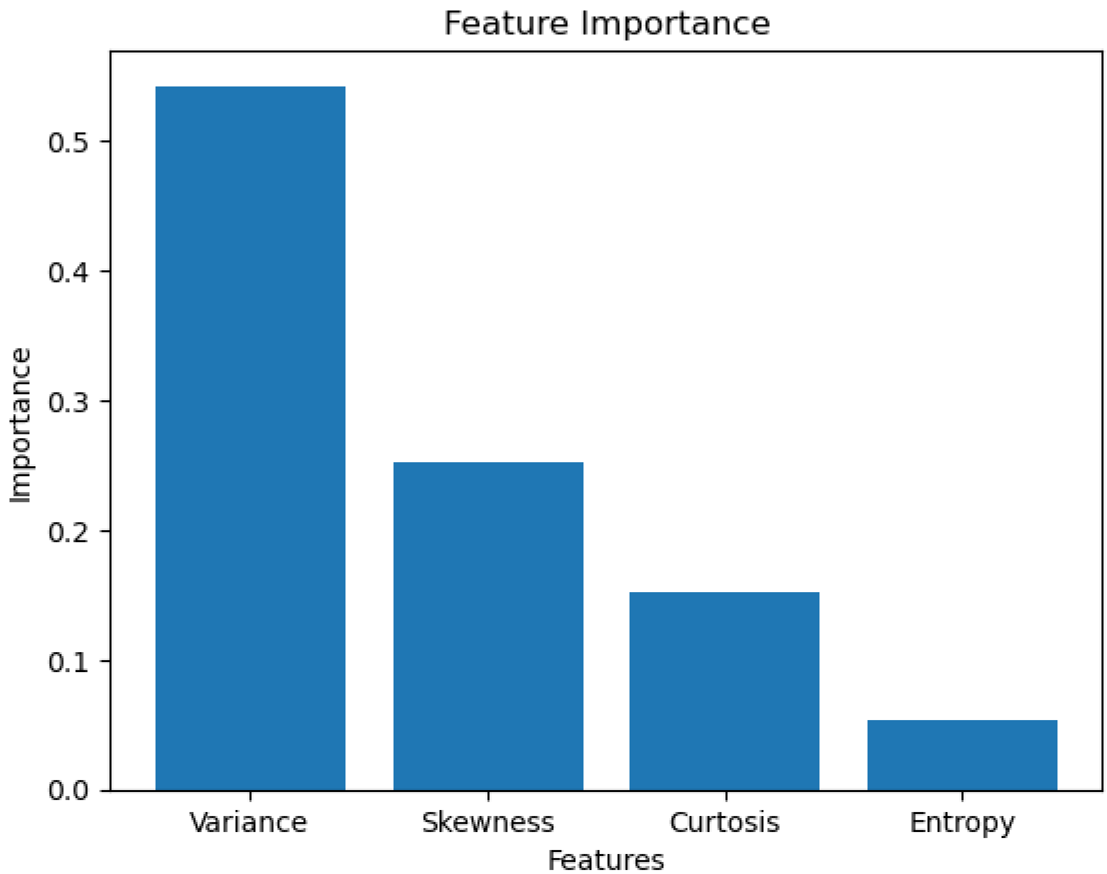
In [22]:
```
# feature importance
clf.feature_importances_
```

Out[22]:
```
array([0.54265136, 0.25175902, 0.15250391, 0.0530857 ])
```

In [23]:
```
feature.columns
```

Out[23]:
```
Index(['Variance', 'Skewness', 'Curtosis', 'Entropy'], dtype='object')
```

In [24]:
```
plt.bar(feature.columns,clf.feature_importances_)
plt.title('Feature Importance')
plt.ylabel('Importance')
plt.xlabel("Features")
plt.show()
```



Confusion Matrix

In [25]:
```
y_pred = clf.predict(X_test)
overall_accuracy = accuracy_score(y_test,y_pred)
overall_accuracy
```

Out[25]:
```
0.9955849889624724
```

In [26]:
```
# evaluating the accuracy of algorithm by 10 Iterations
from sklearn.model_selection import cross_val_score
accuracy_iteration = cross_val_score(clf,X_train,y_train,cv=10)
```

In [27]:
```
j=0
for accuracy in accuracy_iteration:
    print('Accuracy of algorithm by',j+1,'iteration is', accuracy_iteration[j])
    j+=1
```

```
Accuracy of algorithm by 1 iteration is 1.0
Accuracy of algorithm by 2 iteration is 0.9782608695652174
Accuracy of algorithm by 3 iteration is 1.0
Accuracy of algorithm by 4 iteration is 0.9891304347826086
Accuracy of algorithm by 5 iteration is 0.9782608695652174
Accuracy of algorithm by 6 iteration is 0.9782608695652174
Accuracy of algorithm by 7 iteration is 1.0
Accuracy of algorithm by 8 iteration is 1.0
Accuracy of algorithm by 9 iteration is 0.9891304347826086
Accuracy of algorithm by 10 iteration is 1.0
```

In [28]:
```
print('Mode of accuray is' , stat.mode(accuracy_iteration))
```

```
Mode of accuray is 1.0
```

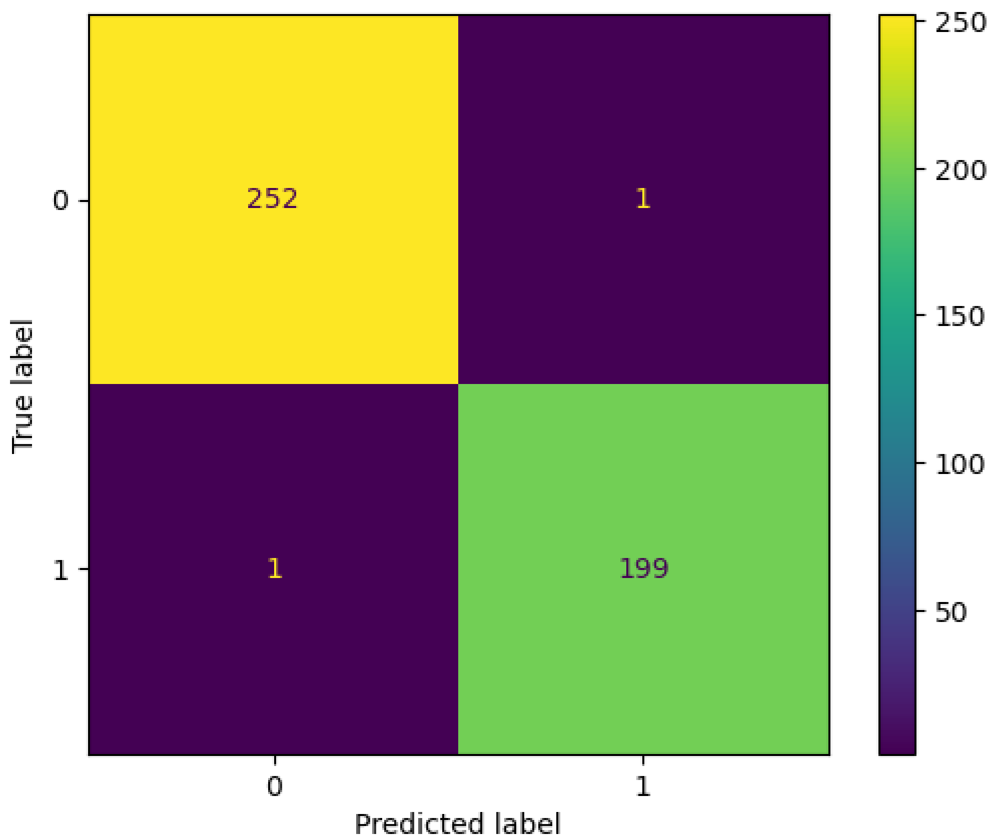
In [29]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[29]:
```
array([[252,   1],
       [  1, 199]], dtype=int64)
```

In [30]:
```
# finding True Negative, False Positive, False Negative and True Positive
tn=cm[0,0]
fp=cm[0,1]
fn=cm[1,0]
tp=cm[1,1]
```

In [31]:
```
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
```

Out[31]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x26ef6e384f0>
```



Approximately less than 1% of the dataset is predicting false.The algorithm is pretty good with the Approximate accuracy of 99%.

Area Under The Curve

In [33]:
```
# finding FPR => False Positive Rate and TPR => True Positive Rate
fpr_dt, tpr_dt, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr_dt, tpr_dt)
roc_auc
```

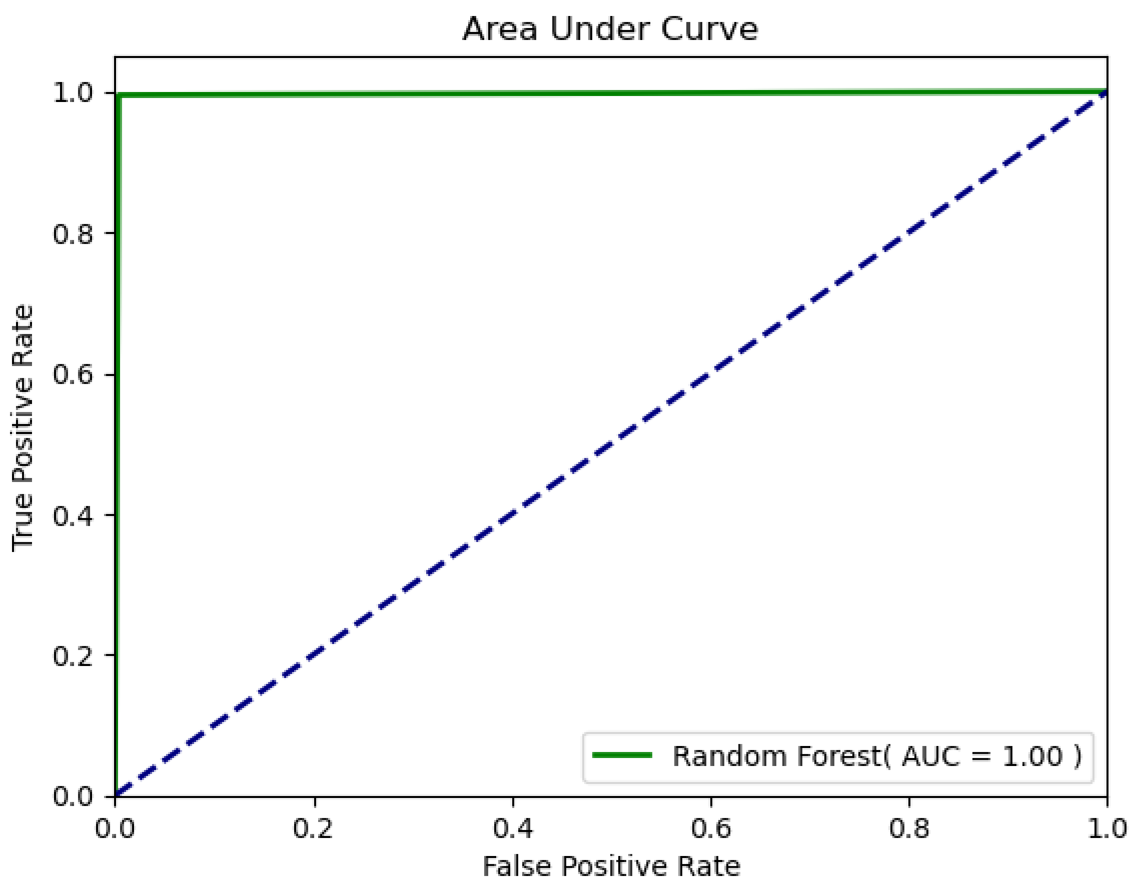
Out[33]:
```
0.9955237154150198
```

In [34]:
```
plt.figure(1)

plt.plot(fpr_dt, tpr_dt, color ='green',lw = 2, label='Random Forest( AUC = %0.2f )'% roc_auc)
plt.plot([0,1],[0,1], color='navy',lw = 2, linestyle='--' )

plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Area Under Curve')
plt.legend(loc='lower right')
plt.show
```

Out[34]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Area under is curve approximately equal to 1 means the model bestest fits the dataset.

In [ ]: