Breast Cancer Detection

Using Machine Learning To Predict Diagnosis of a Breast Cancer

1. Identify the problem

Breast cancer is the most common malignancy among women, accounting for nearly 1 in 3 cancers diagnosed among women in the United States, and it is the second leading cause of cancer death among women. Breast Cancer occurs as a results of abnormal growth of cells in the breast tissue, commonly referred to as a Tumor. A tumor does not mean cancer - tumors can be benign (not cancerous), pre-malignant (pre-cancerous), or malignant (cancerous). Tests such as MRI, mammogram, ultrasound and biopsy are commonly used to diagnose breast cancer performed.

1.1 Expected outcome

Given breast cancer results from breast fine needle aspiration (FNA) test (is a quick and simple procedure to perform, which removes some fluid or cells from a breast lesion or cyst (a lump, sore or swelling) with a fine needle similar to a blood sample needle). Since this build a model that can classify a breast cancer tumor using two training classification:

- 1= Malignant (Cancerous) - Present
- 0= Benign (Not Cancerous) -Absent

1.2 Objective

Since the labels in the data are discrete, the predication falls into two categories, (i.e. Malignant or benign). In machine learning this is a classification problem.

> *Thus, the goal is to classify whether the breast cancer is benign or malignant and predict the recurrence and non-recurrence of malignant cases after a certain period. To achieve this we have used machine learning classification methods to fit a function that can predict the discrete class of new input.*

1.3 Identify data sources

The Breast Cancer datasets is available machine learning repository maintained by the University of California, Irvine. The dataset contains **569 samples of malignant and benign tumor cells**.

- The first two columns in the dataset store the unique ID numbers of the samples and the corresponding diagnosis (M=malignant, B=benign), respectively.
- The columns 3-32 contain 30 real-value features that have been computed from digitized images of the cell nuclei, which can be used to build a model to predict

whether a tumor is benign or malignant.

In [1]:
```
# importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# load dataset
df = pd.read_csv('data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 33 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

Data Preprocessing

In [5]:
```
df.isna().sum()
```

Out[5]:
```
id                         0
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
```

In [6]:
```
df = df.dropna(axis=1)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

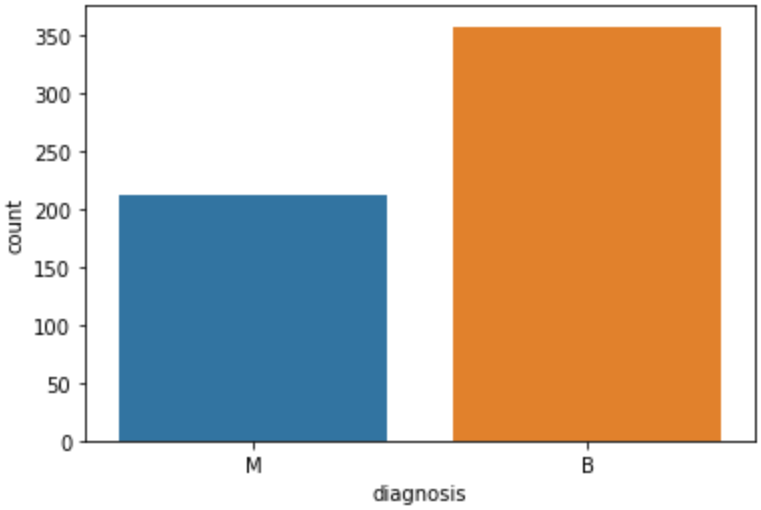
In [8]:
```
# count of malignant and benignate
df['diagnosis'].value_counts()
```

Out[8]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [9]:
```
sns.countplot(df['diagnosis'], label = 'count')
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[9]:
```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



In [10]:
```
df.dtypes
```

Out[10]:
```
id                         int64
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
```

In [11]:
```
# encoding Categorical data
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df.iloc[:,1] = le.fit_transform(df.iloc[:,1].values)
```

In [12]:
```
df.head()
```

Out[12]:

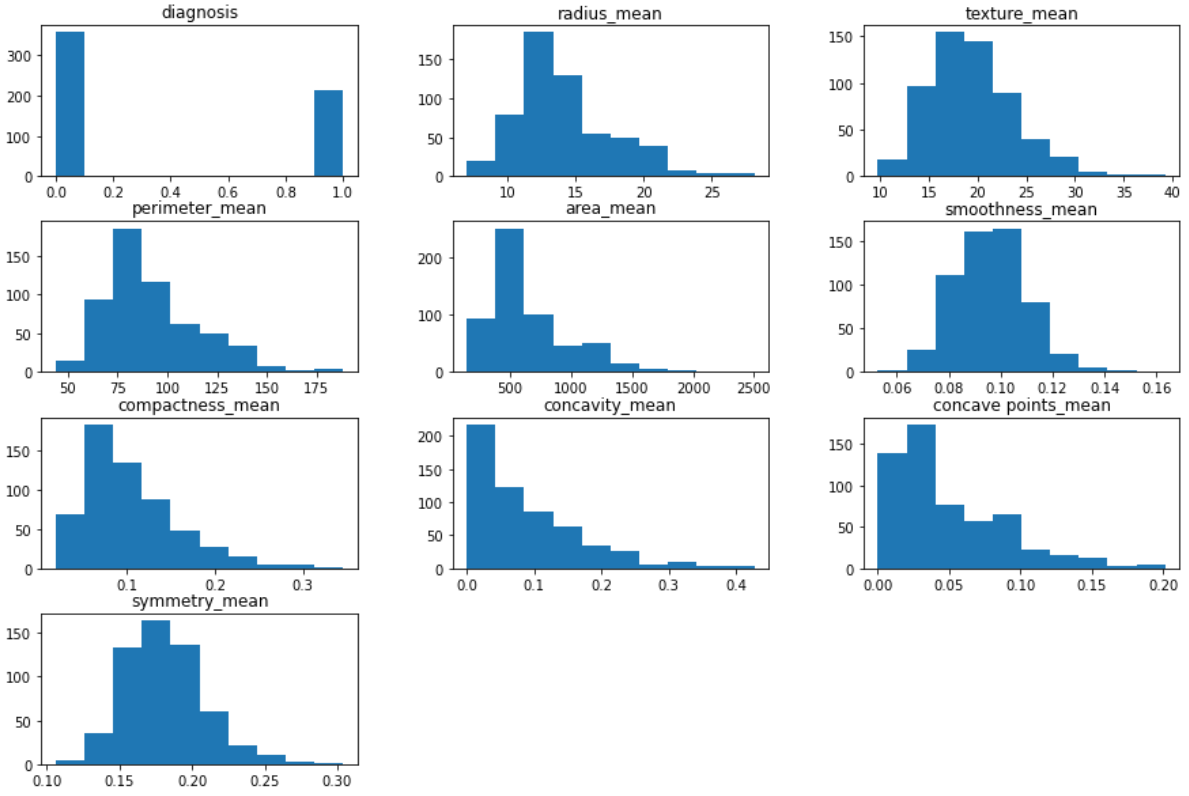
	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	1	17.99	10.38	122.80	1001.0	0.1184
1	842517	1	20.57	17.77	132.90	1326.0	0.0847
2	84300903	1	19.69	21.25	130.00	1203.0	0.1096
3	84348301	1	11.42	20.38	77.58	386.1	0.1425
4	84358402	1	20.29	14.34	135.10	1297.0	0.1003

5 rows × 32 columns

Separate columns into smaller dataframes to perform visualization

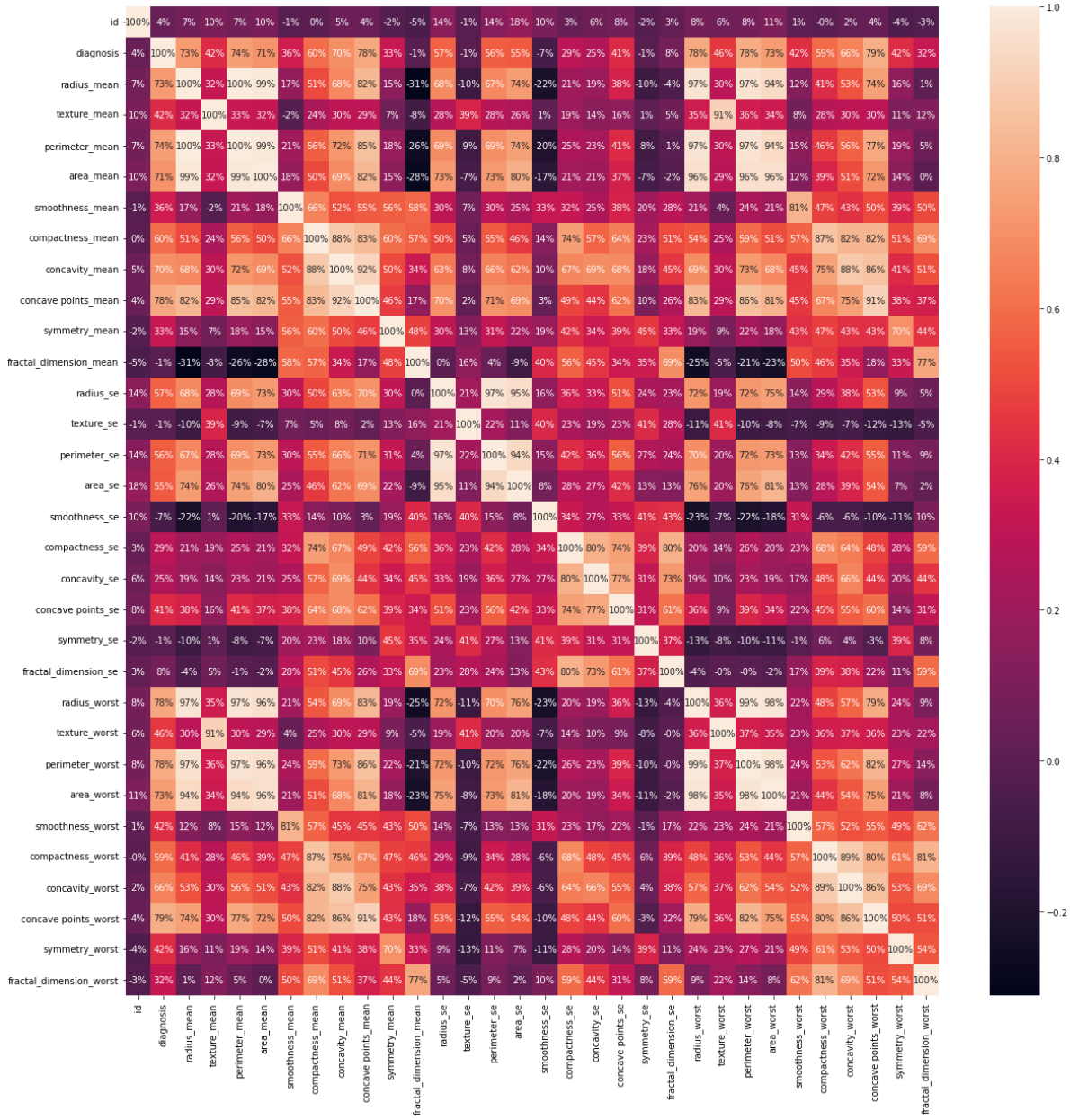
In [13]:
```
data_mean=df.iloc[:,1:11]
```

In [14]:
```
#Plot histograms of CUT1 variables
hist_mean=data_mean.hist(bins=10, figsize=(15, 10),grid=False,)
```

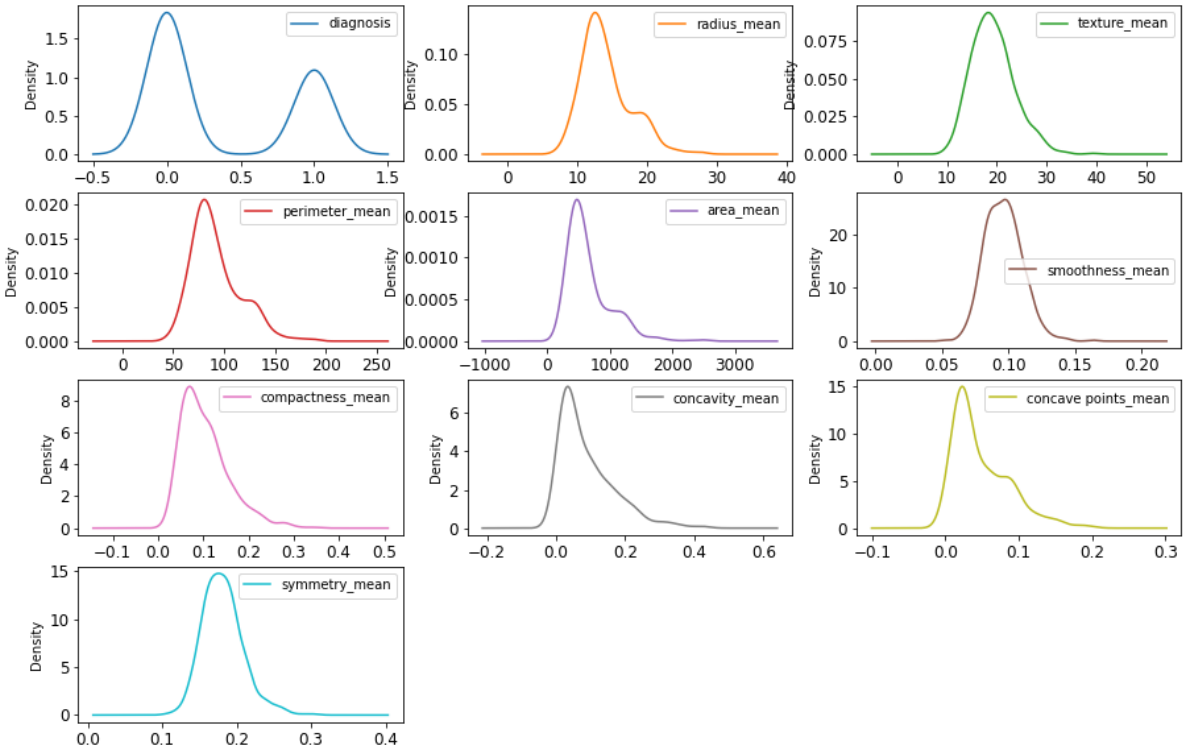


In [15]:
```
#Heatmap
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True, fmt = '.0%')
```

Out[15]:
```
<AxesSubplot:>
```



In [16]:
```
#Density Plots
plt = data_mean.plot(kind= 'density', subplots=True, layout=(4,3), sharex=False,
                     sharey=False,fontsize=12, figsize=(15,10))
```



Spliting the Data

In [17]:
```
# train test split
from sklearn.model_selection import train_test_split
```

In [18]:
```
x = df.drop(['diagnosis'], axis=1)
y = df['diagnosis'].values
```

In [19]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=0)
```

Model Building

In [20]:
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(x_train,y_train)
print("Logistic Regression accuracy : {:.2f}%".format(reg.score(x_test,y_test)*100))
```

```
Logistic Regression accuracy : 58.77%
```

In [21]:
```
# support vector classifier
from sklearn.svm import SVC
svm = SVC(random_state=1)
svm.fit(x_train,y_train)
print("SVC accuracy : {:.2f}%".format(svm.score(x_test,y_test)*100))
```

```
SVC accuracy : 58.77%
```

In [22]:
```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train,y_train)
print(" Naive Bayes accuracy : {:.2f}%".format(nb.score(x_test,y_test)*100))
```

```
Naive Bayes accuracy : 59.65%
```

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=1000,random_state=1)
rf.fit(x_train,y_train)
print("Random Forest Classifier accuracy : {:.2f}%".format(rf.score(x_test,y_test)*
```

```
Random Forest Classifier accuracy : 95.61%
```

In [161…
```
pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xgboost in c:\users\dell\appdata\roaming\python\pyt
hon39\site-packages (2.0.0)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages
(from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages
(from xgboost) (1.7.3)
Note: you may need to restart the kernel to use updated packages.
```

In [162…
```
import xgboost
xg = xgboost.XGBClassifier()
xg.fit(x_train,y_train)
print("XGboost accuracy : {:.2f}%".format(xg.score(x_test,y_test)*100))
```

```
XGboost accuracy : 98.25%
```

AdaBoostClassifier

In [26]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [82]:
```
abc=AdaBoostClassifier(n_estimators=10)
```

In [83]:
```
abc.fit(x_train,y_train)
```

Out[83]:
```
AdaBoostClassifier(n_estimators=10)
```

In [84]:
```
abc.score(x_train,y_train)
```

Out[84]:
```
0.9824175824175824
```

In [85]:
```
abc.score(x_test,y_test)
```

Out[85]:
```
0.9824561403508771
```

GradientBoostingClassifier

In [88]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [153…
```
gbc=GradientBoostingClassifier(n_estimators=10)
```

In [154…
```
gbc.fit(x_train,y_train)
```

Out[154]:
```
GradientBoostingClassifier(n_estimators=10)
```

In [155…
```
gbc.score(x_train,y_train)
```

Out[155]: 0.9846153846153847

In [156…
```
gbc.score(x_test,y_test)
```

Out[156]: 0.9736842105263158

Decision Tree Classifier

In [159…
```
from sklearn.tree import DecisionTreeClassifier
```

In [167…
```
DTC=DecisionTreeClassifier(splitter='random',)
```

In [168…
```
DTC.fit(x_train,y_train)
```

Out[168]:
```
DecisionTreeClassifier(splitter='random')
```

In [169…
```
DTC.score(x_train,y_train)
```

Out[169]: 1.0

In [170…
```
DTC.score(x_test,y_test)
```

Out[170]: 0.956140350877193

Random Forest Model

In [174…
```
from sklearn.ensemble import RandomForestClassifier
```

In [303…
```
RFC=RandomForestClassifier(max_features=8,)
```

In [304…
```
RFC.fit(x_train,y_train)
```

Out[304]:
```
RandomForestClassifier(max_features=8)
```

In [305…
```
RFC.score(x_train,y_train)
```

Out[305]: 1.0

In [306…
```
RFC.score(x_test,y_test)
```

Out[306]: 0.9824561403508771