Machine Learning Lab 5 - Predicting Breast Cancer II

Submitted By
Name: **Rathod Nishit Shailesh**
Register Number: **19112014**
Class: **5 BSc Data Science**

Lab Overview

About the Dataset

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

- ftp.cs.wisc.edu - cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository:

1. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Attribute Information:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Ten real-valued features are computed for each cell nucleus:
   A. radius (mean of distances from center to points on the perimeter)
   B. texture (standard deviation of gray-scale values)
   C. perimeter
   D. area
   E. smoothness (local variation in radius lengths)
   F. compactness (perimeter^2 / area - 1.0)
   G. concavity (severity of concave portions of the contour)
   H. concave points (number of concave portions of the contour)
   I. symmetry
   J. fractal dimension ("coastline approximation" - 1)

- The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.
- All feature values are recoded with four significant digits.
- Missing attribute values: none
- Class distribution: 357 benign, 212 malignant

Objective

- Get familiar with the problem statement, Know the dataset thoroghly, Analyse the given dataset by exploring the hidden insights with beautiful visuals and Train & Test the model for acurate classification prediction of Breast Cancer.

Problem Definition

- Understand the Dataset & Features.
- Perform Data Preprocessing Technique to Get Balanced Structured Data.
- Perform Statistical Data Analysis and Derive Valuable Inferences.
- Perform Exploratory Data Analysis and Derive Valuable Insights.
- Train and Test through Different Classification Models for Better Pricdiction.

Approach

This is an extension to the Problem Defnintion. Mention the process/appraoch that you have followed in order to reach out the above problem defintion.

- Step 1: Know the dataset thoroughly.
- Step 2: Perform preprocessing on data.
- Step 3: Import needfull libraries as an when you try to plot different graphs and evaluate the model.
- Step 4: Perform Statistical Data Analysis and Derive Valuable Inferences.
- Step 5: Perform Exploratory Data Analysis and Derive Valuable Insights.
- Step 6: Train and Test through Different Classification Models for Better Breast Cancer Prediction.
- Step 7: Help the doctors with insights for predicting if a patient is diaganose with breast cancer or not.

Sections

Here, mentioned sections are defined in the below code. For this lab, the sections are -

1. Lab Overview
2. Dataset Overview
3. Data Analyst Process
4. About Different Classification Models
5. Implementation and Evaluation of Different Classification Models
6. Conclusion

References

1. https://pandas.pydata.org/
2. https://matplotlib.org/
3. https://seaborn.pydata.org/
4. https://plotly.com/
5. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
7. https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

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

%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv("BreastCancer.csv")
```

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

Out[3]:
```
(569, 33)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['id', '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')
```

In [5]:
```
df.head()
```

Out[5]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374

5 rows × 33 columns

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

Out[6]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.40	166.10	2027.0	0.14100
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.25	155.00	1731.0	0.11660
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.12	126.70	1124.0	0.11390
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.42	184.60	1821.0	0.16500
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37	59.16	268.6	0.08996

5 rows × 33 columns

In [7]:
```
df.drop('Unnamed: 32', axis = 1, inplace = True)
df.drop('id', axis = 1, inplace = True)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 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
dtypes: float64(30), object(1)
memory usage: 137.9+ KB
```

In [9]:
```
df.sample(5)
```

Out[9]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_wors
528	B	13.94	13.17	90.31	594.2	0.12480	0.09755	0.10100	0.06615	0.1976	...	14.62	15.38	94.52	653.
380	B	11.27	12.96	73.16	386.3	0.12370	0.11110	0.07900	0.05550	0.2018	...	12.84	20.53	84.93	476.
219	M	19.53	32.47	128.00	1223.0	0.08420	0.11300	0.11450	0.06637	0.1428	...	27.90	45.41	180.20	2477.
336	B	12.99	14.23	84.08	514.3	0.09462	0.09965	0.03738	0.02098	0.1652	...	13.72	16.91	87.38	576.
128	B	15.10	16.39	99.58	674.5	0.11500	0.18070	0.11380	0.08534	0.2001	...	16.11	18.33	105.90	762.

5 rows × 31 columns

In [10]:
```
df.describe()
```

Out[10]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223	107.2
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258	33.6
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000	50.4
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000	84.1
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000	97.6
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000	125.4
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000	251.2

8 rows × 30 columns

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
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
dtype: int64
```

Exploratory Data Analysis

$! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

$ from pandas_profiling import ProfileReport

$ profile = ProfileReport(df)

$ profile.to_notebook_iframe()

$ profile.to_file(output_file = "19112014_NishitRathod_EDA_BreastCancer.html")

Implementing Classification Models

Biffurcating Classification Parameter from the Dataset.

In [12]:
```
Y = df['diagnosis']
X = df.drop(['diagnosis'], axis=1)
```

In [13]:
```
X.sample(5)
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_wo
32	17.02	23.98	112.80	899.3	0.11970	0.14960	0.241700	0.12030	0.2248	0.06382	...	20.88	32.09	136.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.10430	0.1809	0.05883	...	22.54	16.67	152.
226	10.44	15.46	66.62	329.6	0.10530	0.07722	0.006643	0.01216	0.1788	0.06450	...	11.52	19.80	73.
112	14.26	19.65	97.83	629.9	0.07837	0.22330	0.300300	0.07798	0.1704	0.07769	...	15.30	23.73	107.
409	12.27	17.92	78.41	466.1	0.08685	0.06526	0.032110	0.02653	0.1966	0.05597	...	14.10	28.88	89.

5 rows × 30 columns

In [14]:
```
Y.sample(5)
```

Out[14]:
```
112    B
254    M
539    B
464    B
132    M
Name: diagnosis, dtype: object
```

Train Test Split

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.30, random_state = 8)
```

In [16]:
```
X_train.sample(5)
```

Out[16]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_wo
128	15.100	16.39	99.58	674.5	0.11500	0.18070	0.11380	0.085340	0.2001	0.06467	...	16.110	18.33	105.
505	9.676	13.14	64.12	272.5	0.12550	0.22040	0.11880	0.070380	0.2057	0.09575	...	10.600	18.04	69.
231	11.320	27.08	71.76	395.7	0.06883	0.03813	0.01633	0.003125	0.1869	0.05628	...	12.080	33.75	79.
305	11.600	24.49	74.23	417.2	0.07474	0.05688	0.01974	0.013130	0.1935	0.05878	...	12.440	31.62	81.
412	9.397	21.68	59.75	268.8	0.07969	0.06053	0.03735	0.005128	0.1274	0.06724	...	9.965	27.99	66.

5 rows × 30 columns

In [17]:
```
X_test.sample(5)
```

Out[17]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_wo
374	13.690	16.07	87.84	579.1	0.08302	0.06374	0.02556	0.02031	0.1872	0.05669	...	14.840	20.21	99.
510	11.740	14.69	76.31	426.0	0.08099	0.09661	0.06726	0.02639	0.1499	0.06758	...	12.450	17.60	81.
126	13.610	24.69	87.76	572.6	0.09258	0.07862	0.05285	0.03085	0.1761	0.06130	...	16.890	35.64	113.
175	8.671	14.45	54.42	227.2	0.09138	0.04276	0.00000	0.00000	0.1722	0.06724	...	9.262	17.04	58.
195	12.910	16.33	82.53	516.4	0.07941	0.05366	0.03873	0.02377	0.1829	0.05667	...	13.880	22.00	90.

5 rows × 30 columns

In [18]:
```
Y_train.sample(5)
```

Out[18]:
```
189    B
238    B
106    B
11     M
506    B
Name: diagnosis, dtype: object
```

In [19]:
```
Y_test.sample(5)
```

Out[19]:
```
100    M
523    B
457    B
431    B
553    B
Name: diagnosis, dtype: object
```

In [20]:
```
from sklearn.metrics import accuracy_score
```

Decision Tree

Fitting and Predicting

In [21]:
```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier()
DT.fit(X_train,Y_train)
predA = DT.predict(X_test)
accuracy_score(predA,Y_test)
```

Out[21]:
```
0.9415204678362573
```

Classification Report

In [22]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predA))
```

```
              precision    recall  f1-score   support

           B       0.95      0.95      0.95       105
           M       0.92      0.92      0.92        66

    accuracy                           0.94       171
   macro avg       0.94      0.94      0.94       171
weighted avg       0.94      0.94      0.94       171
```

Confusion Matrix

In [23]:
```
from sklearn.metrics import confusion_matrix
```

In [24]:
```
CM = confusion_matrix(Y_test, predA)
CM
```

Out[24]:
```
array([[100,   5],
       [  5,  61]], dtype=int64)
```

In [25]:
```
Accuracy = (CM[0][0] + CM[1][1]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
Accuracy
```

Out[25]:
```
0.9415204678362573
```

In [26]:
```
ErrorRate = (CM[0][1] + CM[1][0]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
ErrorRate
```

Out[26]:
```
0.05847953216374269
```

In [27]:
```
Sensitivity = CM[0][0]/(CM[0][0] + CM[1][0])
Sensitivity
```

Out[27]:
```
0.9523809523809523
```

In [28]:
```
Specificity = CM[1][1]/(CM[1][1] + CM[0][1])
Specificity
```

Out[28]:
```
0.9242424242424242
```

In [29]:
```
Recall = CM[0][0]/(CM[0][0] + CM[1][0])
Recall
```

Out[29]:
```
0.9523809523809523
```

In [30]:
```
Precision = CM[0][0]/(CM[0][0] + CM[0][1])
Precision
```

Out[30]:
```
0.9523809523809523
```

In [31]:
```
F1Score = (2*(Precision*Recall))/(Precision + Recall)
F1Score
```

Out[31]:
```
0.9523809523809523
```

Random Forest Classifier

In [32]:
```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators=300)
RF.fit(X_train, Y_train)
predB = RF.predict(X_test)
accuracy_score(predB,Y_test)
```

Out[32]:
```
0.9707602339181286
```

Classification Report

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

```
              precision    recall  f1-score   support

           B       0.95      1.00      0.98       105
           M       1.00      0.92      0.96        66

    accuracy                           0.97       171
   macro avg       0.98      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

Confusion Matrix

In [34]:
```
CM = confusion_matrix(Y_test, predB)
CM
```

Out[34]:
```
array([[105,   0],
       [  5,  61]], dtype=int64)
```

In [35]:
```
Accuracy = (CM[0][0] + CM[1][1]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
Accuracy
```

Out[35]:
```
0.9707602339181286
```

In [36]:
```
ErrorRate = (CM[0][1] + CM[1][0]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
ErrorRate
```

Out[36]:
```
0.029239766081871343
```

In [37]:
```
Sensitivity = CM[0][0]/(CM[0][0] + CM[1][0])
Sensitivity
```

Out[37]:
```
0.9545454545454546
```

In [38]:
```
Specificity = CM[1][1]/(CM[1][1] + CM[0][1])
Specificity
```

Out[38]:
```
1.0
```

In [39]:
```
Recall = CM[0][0]/(CM[0][0] + CM[1][0])
Recall
```

Out[39]:
```
0.9545454545454546
```

In [40]:
```
Precision = CM[0][0]/(CM[0][0] + CM[0][1])
Precision
```

Out[40]:
```
1.0
```

In [41]:
```
F1Score = (2*(Precision*Recall))/(Precision + Recall)
F1Score
```

Out[41]:
```
0.9767441860465117
```

Evaluating the Effect of Parameters For Decision Tree

In [42]:
```
def doDecisionTree(X, Y, test_size = 0.20, randomstate = 8,c='gini',mf = 'auto'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, random_state = randomstate)
    cls2 = DecisionTreeClassifier(criterion = c, max_features = mf)
    cls2.fit(X_train,Y_train)
    pred = cls2.predict(X_test)
    acc_score2 = accuracy_score(pred,Y_test)
    return acc_score2
```

In [43]:
```
test_size = [0.30, 0.25, 0.20,0.10]
random_states = [8, 27, 42]
n_neighbours = [2,3,4,5]

criterions=['gini', 'entropy']
maxfeatures=['auto', 'sqrt', 'log2']
```

In [44]:
```
df1 = pd.DataFrame(columns = ['Test Size', 'Random States','Decision Tree Accuracy','Criterions','Max features'])
```

In [45]:
```
for t_size in test_size:
    for r_state in random_states:
        for crs in criterions:
            for mfs in maxfeatures:

                Algo1 = doDecisionTree(X, Y, t_size, r_state, crs, mfs)
                DecisionTree = {}
                DecisionTree['Test Size'] = t_size
                DecisionTree['Random States'] = r_state
                DecisionTree['Decision Tree Accuracy'] = Algo1
                DecisionTree['Criterions'] = crs
                DecisionTree['Max features'] = mfs

                df1 = df1.append(DecisionTree, ignore_index = True)
```

In [46]:
```
df1.sample(10)
```

Out[46]:

	Test Size	Random States	Decision Tree Accuracy	Criterions	Max features
6	0.30	27	0.894737	gini	auto
35	0.25	42	0.944056	entropy	log2
25	0.25	27	0.958042	gini	sqrt
15	0.30	42	0.947368	entropy	auto
30	0.25	42	0.951049	gini	auto
17	0.30	42	0.923977	entropy	log2
50	0.20	42	0.956140	gini	log2
54	0.10	8	0.947368	gini	auto
19	0.25	8	0.944056	gini	sqrt
53	0.20	42	0.956140	entropy	log2

Evaluating the Effect of Parameters For Random Forest Classification

In [47]:
```
nesti = 100
def doRandomForestClassifier(X, Y, test_size = 0.20, randomstate = 8,c = 'gini',mf = 'auto'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, random_state = randomstate)
    cls2 = RandomForestClassifier(criterion = c, max_features = mf, n_estimators = nesti)
    cls2.fit(X_train,Y_train)
    pred = cls2.predict(X_test)
    acc_score2 = accuracy_score(pred,Y_test)
    return acc_score2
```

In [48]:
```
test_size = [0.30, 0.25, 0.20,0.10]
random_states = [8, 27, 42]
n_neighbours = [2,3,4,5]
n_estimator = [100,200,300,400]

criterions=['gini', 'entropy']
maxfeatures=['auto', 'sqrt', 'log2']
```

In [49]:
```
df2 = pd.DataFrame(columns = ['Test Size', 'Random States','N_Estimators','Randome State Accuracy','Criterions','Max features'])
```

In [50]:
```
for t_size in test_size:
    for r_state in random_states:
        for n_esti in n_estimator:
            for crs in criterions:
                for mfs in maxfeatures:

                    Algo2 = doRandomForestClassifier(X, Y, t_size, r_state, crs, mfs)
                    RandomForest = {}
                    RandomForest['Test Size'] = t_size
                    RandomForest['Random States'] = r_state
                    RandomForest['N_Estimators'] = n_esti
                    RandomForest['Randome State Accuracy'] = Algo2
                    RandomForest['Criterions'] = crs
                    RandomForest['Max features'] = mfs

                    df2 = df2.append(RandomForest, ignore_index = True)
```

In [51]:
```
df2.sample(10)
```

Out[51]:

	Test Size	Random States	N_Estimators	Randome State Accuracy	Criterions	Max features
46	0.30	27	400	0.964912	entropy	sqrt
225	0.10	8	200	0.947368	entropy	auto
269	0.10	42	100	0.964912	entropy	log2
229	0.10	8	300	0.929825	gini	sqrt
214	0.20	42	400	0.964912	entropy	sqrt
75	0.25	8	100	0.951049	entropy	auto
90	0.25	8	400	0.951049	gini	auto
4	0.30	8	100	0.970760	entropy	sqrt
22	0.30	8	400	0.970760	entropy	sqrt
278	0.10	42	300	0.964912	gini	log2

Decision Tree V/S Random Forest

In [52]:
```
df3 = pd.DataFrame(columns = ['Test Size', 'Random States','Decision Tree Accuracy','Random Forest Accuracy'])
```

In [53]:
```
for t_size in test_size:
    for r_state in random_states:
        Algo1 = doDecisionTree(X, Y, t_size, r_state)
        Algo2 = doRandomForestClassifier(X, Y, t_size, r_state)
        DTvsRF = {}
        DTvsRF['Test Size'] = t_size
        DTvsRF['Random States'] = r_state
        DTvsRF['Decision Tree Accuracy'] = Algo1
        DTvsRF['Random Forest Accuracy'] = Algo2

        df3 = df3.append(DTvsRF, ignore_index = True)
```

In [54]:
```
df3.sample(10)
```

Out[54]:

	Test Size	Random States	Decision Tree Accuracy	Random Forest Accuracy
4	0.25	27.0	0.902098	0.965035
1	0.30	27.0	0.894737	0.970760
9	0.10	8.0	0.912281	0.964912
5	0.25	42.0	0.930070	0.965035
7	0.20	27.0	0.938596	0.964912
3	0.25	8.0	0.951049	0.951049
11	0.10	42.0	0.929825	0.964912
6	0.20	8.0	0.912281	0.956140
8	0.20	42.0	0.929825	0.964912
0	0.30	8.0	0.959064	0.964912

In [55]:
```
df3.to_csv('DecisionTree_VS_RandomForest.csv')
```

Exploratory Data Analysis On The Results Derived From - Decision Tree and Random Forest Algorithm.

$! pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

$ from pandas_profiling import ProfileReport

$ profile = ProfileReport(df3)

$ profile.to_notebook_iframe()

$ profile.to_file(output_file = "19112014_NishitRathod_EDA_BreastCancer_ResultsDerivedFromAlgorithms.html")

Conclusion

In this lab, we have tried to gain the knowledge about data and its variables, further we did some preprocessing to the data in order to bring it into more analyst friendly mode, laterly we implemented various graphs using various libraries in order to get valuable insights, furthermore, we implemented and evaluated various classification models to get high accuracy in terms of predicting breast cancer which can help the doctors to predict breast cancer for the patients.