Machine Learning Lab 4 - Predicting Breast Cancer

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.linear_model.LogisticRegression.html
6. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
7. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
8. 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	compactness_worst	concavity_wo
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	0.6656	0.71
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	0.1866	0.24
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	0.4245	0.45
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	0.8663	0.68
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	0.2050	0.40

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	compactness_worst	concavity_wo
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	0.21130	0.41
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	0.19220	0.32
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	0.30940	0.34
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	0.86810	0.93
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	0.06444	0.00

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_worst	smoothness_worst	compactnes
543	B	13.210	28.06	84.88	538.4	0.08671	0.06877	0.02987	0.03275	0.1628	...	14.37	37.17	92.48	629.6	0.10720	
68	B	9.029	17.33	58.79	250.5	0.10660	0.14130	0.31300	0.04375	0.2111	...	10.31	22.65	65.50	324.7	0.14820	
362	B	12.760	18.84	81.87	496.6	0.09676	0.07952	0.02688	0.01781	0.1759	...	13.75	25.99	87.82	579.7	0.12980	
24	M	16.650	21.38	110.00	904.6	0.11210	0.14570	0.15250	0.09170	0.1995	...	26.46	31.56	177.00	2215.0	0.18050	
397	B	12.800	17.46	83.05	508.3	0.08044	0.08895	0.07390	0.04083	0.1574	...	13.74	21.06	90.72	591.0	0.09534	

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_worst	area_worst	smoothness_
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	569.000000	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	107.261213	880.583128	0.1
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	33.602542	569.356993	0.0
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	50.410000	185.200000	0.0
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	84.110000	515.300000	0.1
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	97.660000	686.500000	0.1
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	125.400000	1084.000000	0.1
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	251.200000	4254.000000	0.2

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_worst	area_worst	smoothness_wors
131	15.46	19.48	101.70	748.9	0.10920	0.12230	0.14660	0.080870	0.1931	0.05796	...	19.26	26.00	124.90	1156.0	0.154
200	12.23	19.56	78.54	461.0	0.09586	0.08087	0.04187	0.041070	0.1979	0.06013	...	14.44	28.36	92.15	638.4	0.142
139	11.28	13.39	73.00	384.8	0.11640	0.11360	0.04635	0.047960	0.1771	0.06072	...	11.92	15.77	76.53	434.0	0.136
135	12.77	22.47	81.72	506.3	0.09055	0.05761	0.04711	0.027040	0.1585	0.06065	...	14.49	33.37	92.04	653.6	0.141
144	10.75	14.97	68.26	355.3	0.07793	0.05139	0.02251	0.007875	0.1399	0.05688	...	11.95	20.72	77.79	441.2	0.107

5 rows × 30 columns

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

Out[14]:
```
488    B
487    M
208    B
445    B
207    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_worst	area_worst	smoothness_wors
171	13.43	19.63	85.84	565.4	0.09048	0.06288	0.05858	0.03438	0.1598	0.05671	...	17.98	29.87	116.60	993.6	0.140
518	12.88	18.22	84.45	493.1	0.12180	0.16610	0.04825	0.05303	0.1709	0.07253	...	15.05	24.37	99.31	674.7	0.145
352	25.73	17.46	174.20	2010.0	0.11490	0.23630	0.33680	0.19130	0.1956	0.06121	...	33.13	23.58	229.30	3234.0	0.153
158	12.06	12.74	76.84	448.6	0.09311	0.05241	0.01972	0.01963	0.1590	0.05907	...	13.14	18.41	84.08	532.8	0.127
166	10.80	9.71	68.77	357.6	0.09594	0.05736	0.02531	0.01698	0.1381	0.06400	...	11.60	12.02	73.66	414.0	0.143

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_worst	area_worst	smoothness_wors
537	11.690	24.44	76.37	406.4	0.12360	0.15520	0.045150	0.045310	0.2131	0.07405	...	12.980	32.19	86.12	487.7	0.176
327	12.030	17.93	76.09	446.0	0.07683	0.03892	0.001546	0.005592	0.1382	0.06070	...	13.070	22.25	82.74	523.4	0.101
538	7.729	25.49	47.98	178.8	0.08098	0.04878	0.000000	0.000000	0.1870	0.07285	...	9.077	30.92	57.17	248.0	0.125
175	8.671	14.45	54.42	227.2	0.09138	0.04276	0.000000	0.000000	0.1722	0.06724	...	9.262	17.04	58.36	259.2	0.116
338	10.050	17.53	64.41	310.8	0.10070	0.07326	0.025110	0.017750	0.1890	0.06331	...	11.160	26.84	71.98	384.0	0.140

5 rows × 30 columns

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

Out[18]:
```
337    M
242    B
164    M
322    B
512    M
Name: diagnosis, dtype: object
```

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

Out[19]:
```
122    M
328    M
207    M
442    B
436    B
Name: diagnosis, dtype: object
```

Logistic Regression

Fitting and Predicting

In [20]:
```
from sklearn.linear_model import LogisticRegression
logistic_regressor = LogisticRegression()
```

In [21]:
```
logistic_regressor.fit(X_train, Y_train)
```

Out[21]:
```
LogisticRegression()
```

In [22]:
```
predA = logistic_regressor.predict(X_test)
y_prob = logistic_regressor.predict_proba(X_test)
```

In [23]:
```
predA[0:5]
```

Out[23]:
```
array(['B', 'B', 'B', 'B', 'B'], dtype=object)
```

In [24]:
```
from sklearn.metrics import accuracy_score
accuracy_score(predA, Y_test)
```

Out[24]:
```
0.9532163742690059
```

In [25]:
```
y_prob[0:5]
```

Out[25]:
```
array([[0.99752283, 0.00247717],
       [0.98704683, 0.01295317],
       [0.99537735, 0.00462265],
       [0.99631079, 0.00368921],
       [0.99835446, 0.00164554]])
```

K - Nearest Neighbours

Fitting and Predicting

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors = 2)
KNN.fit(X_train, Y_train)
predB = KNN.predict(X_test)
accuracy_score(predB, Y_test)
```

Out[26]:
```
0.935672514619883
```

Decision Tree

Fitting and Predicting

In [27]:
```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier()
DT.fit(X_train,Y_train)
predC = DT.predict(X_test)
accuracy_score(predC,Y_test)
```

Out[27]:
```
0.9298245614035088
```

Classification Report

In [28]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, predA))
```

```
              precision    recall  f1-score   support

           B       0.96      0.96      0.96       105
           M       0.94      0.94      0.94        66

    accuracy                           0.95       171
   macro avg       0.95      0.95      0.95       171
weighted avg       0.95      0.95      0.95       171
```

Confusion Matrix

In [29]:
```
from sklearn.metrics import confusion_matrix
```

In [30]:
```
CM = confusion_matrix(Y_test, predA)
CM
```

Out[30]:
```
array([[101,   4],
       [  4,  62]], dtype=int64)
```

In [31]:
```
Accuracy = (CM[0][0] + CM[1][1]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
Accuracy
```

Out[31]:
```
0.9532163742690059
```

In [32]:
```
ErrorRate = (CM[0][1] + CM[1][0]) / (CM[0][0] + CM[1][1] + CM[0][1] + CM[1][0])
ErrorRate
```

Out[32]:
```
0.04678362573099415
```

In [33]:
```
Sensitivity = CM[0][0]/(CM[0][0] + CM[1][0])
Sensitivity
```

Out[33]:
```
0.9619047619047619
```

In [34]:
```
Specificity = CM[1][1]/(CM[1][1] + CM[0][1])
Specificity
```

Out[34]:
```
0.9393939393939394
```

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

Out[35]:
```
0.9619047619047619
```

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

Out[36]:
```
0.9619047619047619
```

In [37]:
```
F1Score = (2*(Precision*Recall))/(Precision + Recall)
F1Score
```

Out[37]:
```
0.9619047619047619
```

Evaluating the Effect of Parameters For Logistic Regression

In [38]:
```
from sklearn.metrics import accuracy_score
```

In [39]:
```
def doLogisticRegression(X, Y, test_size = 0.30, random_state = 42, penalty='l2', solver='lbfgs'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, random_state = random_state)

    logistic_regressor = LogisticRegression(penalty = penalty, solver = solver)
    logistic_regressor.fit(X_train, Y_train)
    y_pred = logistic_regressor.predict(X_test)

    acc_score = accuracy_score(Y_test, y_pred)

    return acc_score
```

In [40]:
```
df2 = pd.DataFrame(columns = ['Test Size', 'Random States', 'Penalty', 'Solvers', 'Accuracy'])
df2
```

Out[40]:

Test Size	Random States	Penalty	Solvers	Accuracy

In [41]:
```
penalties = ['none', 'l2']
test_size = [0.30, 0.25, 0.20]
random_states = [21, 42, 84]
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

for t_size in test_size:
    for r_state in random_states:
        for penalty in penalties:
            for solver in solvers:
                accuracy = doLogisticRegression(X, Y, t_size, r_state, penalty)
                BreastCancerResults = {}
                BreastCancerResults['Test Size'] = t_size
                BreastCancerResults['Random States'] = r_state
                BreastCancerResults['Penalty'] = penalty
                BreastCancerResults['Solvers'] = solver
                BreastCancerResults['Accuracy'] = accuracy

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

In [42]:
```
df2.sample(10)
```

Out[42]:

	Test Size	Random States	Penalty	Solvers	Accuracy
23	0.30	84	none	sag	0.959064
54	0.25	84	none	saga	0.965035
84	0.20	84	none	saga	0.973684
27	0.30	84	l2	liblinear	0.959064
77	0.20	42	l2	liblinear	0.964912
9	0.30	21	l2	saga	0.947368
25	0.30	84	l2	newton-cg	0.959064
43	0.25	42	none	sag	0.965035
68	0.20	21	l2	sag	0.921053
58	0.25	84	l2	sag	0.965035

Evaluating the Effect of Parameters For K Nearest Neighbours

In [43]:
```
def doKNearestNeighbour(X, Y, test_size = 0.20, randomstate = 8,nn = 5 ):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = test_size, random_state = randomstate)
    cls1 = KNeighborsClassifier(n_neighbors = nn)
    cls1.fit(X_train, Y_train)
    pred1 = cls1.predict(X_test)
    acc_score1 = accuracy_score(pred1,Y_test)
    return acc_score1
```

In [44]:
```
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']
penalties = [ 'l1', 'elasticnet','none', 'l2']
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
```

In [45]:
```
df3 = pd.DataFrame(columns = ['Test Size', 'Random States','Number of neighbours','K-nearest neighbour Accuracy'])
```

In [46]:
```
for t_size in test_size:
    for r_state in random_states:
        for neigh in n_neighbours:
            a1 = doKNearestNeighbour(X, Y, t_size, r_state,neigh)
            KNearestNeighbours = {}
            KNearestNeighbours['Test Size'] = t_size
            KNearestNeighbours['Random States'] = r_state
            KNearestNeighbours['Number of neighbours'] = neigh
            KNearestNeighbours['K-nearest neighbour Accuracy'] = a1

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

In [47]:
```
df3.sample(10)
```

Out[47]:

	Test Size	Random States	Number of neighbours	K-nearest neighbour Accuracy
22	0.25	42.0	4.0	0.951049
5	0.30	27.0	3.0	0.923977
43	0.10	27.0	5.0	0.947368
9	0.30	42.0	3.0	0.941520
2	0.30	8.0	4.0	0.947368
31	0.20	27.0	5.0	0.929825
0	0.30	8.0	2.0	0.935673
4	0.30	27.0	2.0	0.906433
28	0.20	27.0	2.0	0.885965
26	0.20	8.0	4.0	0.964912

Evaluating the Effect of Parameters For Decision Tree

In [48]:
```
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)
    pred2 = cls2.predict(X_test)
    acc_score2 = accuracy_score(pred2,Y_test)
    return acc_score2
```

In [49]:
```
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']
penalties = [ 'l1', 'elasticnet','none', 'l2']
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
```

In [50]:
```
df4 = pd.DataFrame(columns = ['Test Size', 'Random States','Decision Tree Accuracy','Criterions','Max features'])
```

In [51]:
```
for t_size in test_size:
    for r_state in random_states:
        for crs in criterions:
            for mfs in maxfeatures:

                a2 = doDecisionTree(X, Y, t_size, r_state, crs, mfs)
                DecisionTree = {}
                DecisionTree['Test Size'] = t_size
                DecisionTree['Random States'] = r_state
                DecisionTree['Decision Tree Accuracy'] = a2
                DecisionTree['Criterions'] = crs
                DecisionTree['Max features'] = mfs

                df4 = df4.append(DecisionTree, ignore_index = True)
```

In [52]:
```
df4.sample(10)
```

Out[52]:

	Test Size	Random States	Decision Tree Accuracy	Criterions	Max features
23	0.25	8	0.937063	entropy	log2
66	0.10	42	0.912281	gini	auto
14	0.30	42	0.941520	gini	log2
51	0.20	42	0.938596	entropy	auto
29	0.25	27	0.909091	entropy	log2
68	0.10	42	0.947368	gini	log2
65	0.10	27	0.929825	entropy	log2
17	0.30	42	0.929825	entropy	log2
59	0.10	8	0.912281	entropy	log2
27	0.25	27	0.916084	entropy	auto

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.