BREAST CANCER DETECTION AND ANALYSIS

➢ INTRODUCTION

One of the most prevalent and leading causes of cancer in women is breast cancer. It has now become a frequent health problem, and its prevalence has recently increased. The easiest approach to dealing with breast cancer findings is to recognize them early on. Early detection of breast cancer is facilitated by computer-aided detection and diagnosis (CAD) technologies, which can help people live longer lives.

AI's use in clinical areas is growing quickly because of its success in predicting and grouping, especially in the clinical analysis of breast cancer. It is also used a lot in biomedical research.

In this Machine learning project we are going to analyze and classify Breast Cancer (that the breast cancer belongs to which category), as basically there are two categories of breast cancer that is:

- Malignant type breast cancer
- Benign type breast cancer

➢ ALGORITHM

- Data Collection
- Data Formatting
- Model Selection
- Training
- Testing

Data Collection: We have collected data and made our own dataset. We have created the .csv file in which information was present.

Data Formatting: The collected data is formatted into suitable data sets. We check the collinearity with Standard Division Blue.

Model Selection: We have tried different models to have which will give us best result with minimum the error of the predicted value. But Linear Regression Model gave us the best result with 95%accuracy.

Training: The data sets was divided such that x_train is used to trainthe model with corresponding x_test value.

Testing: The model was tested with y_train and stored in y_predict . Both y_train and y_predict was compared.

In this work, we will perform classification using the Breast cancer dataset.
This notebook is a simple classification comparison model that want to highlight the main steps to follow. In the first part, we import data and perform some analysis: histogram representation to show the data distribution, correlation matrix representation, outliers detection and elimination. Then we will use GridSerch to compare the performance of three classification algorithms and to performhyperparameters tuning.
In the third part, we perform the classification using the model selected with the best parameters anddiscuss the results. In the fourth section, we perform the same task using a neural network. In the last part, we compare the results obtained and some comments on this project.

Import the necessary libraries and dataset and reframing dataset into data frame

Data set used : Breast Cancer Wisconsin (Diagnostic) Data Set

```
In [1]: #Importing the libraries
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import sklearn.datasets

In [2]: # Reading data from data file and storing as a dataframe
        breast_cancer_dataset=sklearn.datasets.load_breast_cancer()
        df=pd.DataFrame(breast_cancer_dataset.data, columns=breast_cancer_dataset.feature_names)

In [3]: breast_cancer_dataset
```

Printing the data frame

```
In [4]: df.head(6)
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	v smooth
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.1578	0.08089	0.2087	0.07613	...	15.47	23.75	103.40	741.6	0.

6 rows × 30 columns

Adding diagnosis column to the dataframe

```
In [5]: # Adding the target column to the data frame
        df['Diagnosis'] = breast_cancer_dataset.target

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

Out[6]:

mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	Diagnosis
0.31740	0.14740	0.2149	0.06879	...	29.41	179.10	1819.0	0.14070	0.41860	0.6599	0.2542	0.2929	0.09873	0
0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0
0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	0
0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	0
0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	0
0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	1

Checking and correcting the data frame for any null values

```
In [7]: # Returns if any null values or not
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  Diagnosis                569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

Describing the dataframe

```
In [8]: # returns size of the dataset
        df.shape

Out[8]: (569, 31)

In [9]: #describing dataset
        df.describe()
```

Out[9]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	peri
count	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	...	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	...	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	...	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	...	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	...	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	...	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	...	49.540000	251.2

8 rows × 31 columns

```
In [10]: df['Diagnosis'].value_counts()

Out[10]: 1    357
         0    212
         Name: Diagnosis, dtype: int64
```

Splitting the data set into training and testing data

```
In [11]: #Separating features and diagnosis column, Dependant (X), Independant(Y)
         X = df.drop(columns='Diagnosis',axis=1)
         Y = df['Diagnosis']

In [12]: #Separating the data into test and train data set
         from sklearn.model_selection import train_test_split
         X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

Scaling and fitting the data

```
In [13]: #feature scaling
         from sklearn.preprocessing import StandardScaler
         X_train=StandardScaler().fit_transform(X_train)
         X_test=StandardScaler().fit_transform(X_test)
```

Training algorithms against training data

In this step we train Logistic Regression, Decision tree and random forest algorithm against our training data for further statistical analysis.
Then we assign the output of the function into a variable named model.

```
In [14]: def models(X_train,Y_train):

             # Logistic Regression
             from sklearn.linear_model import LogisticRegression
             log=LogisticRegression(random_state=0)
             log.fit(X_train,Y_train)

             # Decision tree
             from sklearn.tree import DecisionTreeClassifier
             tree=DecisionTreeClassifier(random_state=0,criterion="entropy")
             tree.fit(X_train,Y_train)

             # Random forest
             from sklearn.ensemble import RandomForestClassifier
             forest=RandomForestClassifier(random_state=0,criterion="entropy",n_estimators=10)
             forest.fit(X_train,Y_train)

             return log,tree,forest

In [15]: model=models(X_train,Y_train)
```

Testing the model

In this step, we analyze our model for the 3 algorithms against our testing data set.

```
In [16]: # Testing the models

         from sklearn.metrics import accuracy_score

         for i in range(len(model)):
             print("Model",i)
             print("Accuracy : ",100*accuracy_score(Y_test,model[i].predict(X_test)), "%")

Model 0
Accuracy :  95.6140350877193 %
Model 1
Accuracy :  93.85964912280701 %
Model 2
Accuracy :  97.36842105263158 %
```

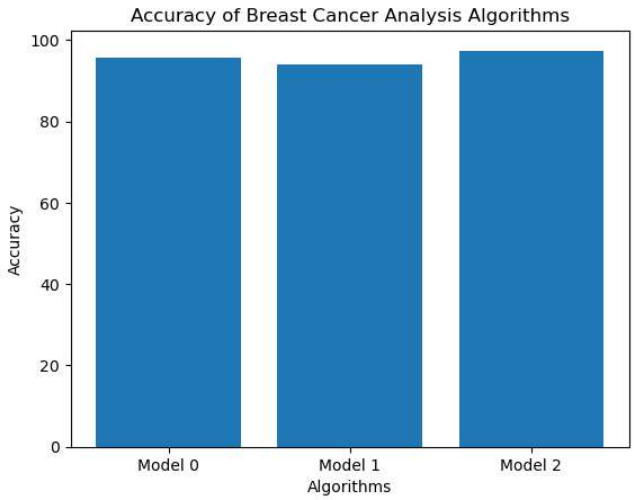
CONCLUSION

In [21]:
```
algorithm_names = ['Model 0', 'Model 1', 'Model 2']
accuracy_values = [0,0,0]
for i in range(len(model)):
    accuracy_values[i] = 100*accuracy_score(Y_test,model[i].predict(X_test))

# Plotting the bar chart
plt.bar(algorithm_names, accuracy_values)

# Adding labels and title
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.title('Accuracy of Breast Cancer Analysis Algorithms')

# Displaying the chart
plt.show()
```



As evident from the bar chart, our model 2 performed best against the testing data set.