# Breast Cancer Prediction

**Breast Cancer Prediction** is a classification task aimed at predicting the diagnosis of a breast mass as either malignant or benign. The dataset used for this prediction consists of features computed from a digitized image of a fine needle aspirate (FNA) of the breast mass. These features describe various characteristics of the cell nuclei present in the image.

The dataset contains the following information for each instance:

1. ID number: A unique identifier for each sample.
2. Diagnosis: The target variable indicating the diagnosis, where 'M' represents malignant and 'B' represents benign.

For each cell nucleus, ten real-valued features are computed, which are:

1. Radius: The mean distance from the center to points on the perimeter of the nucleus.
2. Texture: The standard deviation of gray-scale values in the nucleus.
3. Perimeter: The perimeter of the nucleus.
4. Area: The area of the nucleus.
5. Smoothness: A measure of local variation in radius lengths.
6. Compactness: Computed as the square of the perimeter divided by the area minus 1.0.
7. Concavity: Describes the severity of concave portions of the nucleus contour.
8. Concave points: Represents the number of concave portions of the nucleus contour.
9. Symmetry: Measures the symmetry of the nucleus.
10. Fractal dimension: This feature approximates the "coastline" of the nucleus, using the concept of fractal geometry.

These features provide quantitative measurements that can be used to assess the characteristics of cell nuclei and aid in distinguishing between malignant and benign breast masses. By training a machine learning model on this dataset, it is possible to develop a predictive model that can assist in the early detection and diagnosis of breast cancer.

In [ ]:

```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
#importing the dataset
df = pd.read_csv('data.csv')
df.head()
```

Out[ ]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoo |
|---|---|---|---|---|---|---|---|
| **0** | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | |
| **1** | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | |
| **2** | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | |
| **3** | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | |
| **4** | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | |

5 rows × 33 columns

# Data Preprocessing Part 1

In [ ]:

```
# dropping unnecessary columns
df.drop(['Unnamed: 32','id'],axis=1,inplace=True)
```

In [ ]:

```
#checking for the missing values
df.isnull().sum()
```

Out[ ]:

```
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
```

In [ ]:

```
#checking the data types of the columns
df.dtypes
```

Out[ ]:

```
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 [ ]:

```
# checking the data description
df.describe()
```

Out[ ]:

| | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | cc |
|---|---|---|---|---|---|---|
| **count** | 569.000000 | 569.000000 | 569.000000 | 569.000000 | 569.000000 | |
| **mean** | 14.127292 | 19.289649 | 91.969033 | 654.889104 | 0.096360 | |
| **std** | 3.524049 | 4.301036 | 24.298981 | 351.914129 | 0.014064 | |
| **min** | 6.981000 | 9.710000 | 43.790000 | 143.500000 | 0.052630 | |
| **25%** | 11.700000 | 16.170000 | 75.170000 | 420.300000 | 0.086370 | |
| **50%** | 13.370000 | 18.840000 | 86.240000 | 551.100000 | 0.095870 | |
| **75%** | 15.780000 | 21.800000 | 104.100000 | 782.700000 | 0.105300 | |
| **max** | 28.110000 | 39.280000 | 188.500000 | 2501.000000 | 0.163400 | |

8 rows × 30 columns

# Exploratory Data Analysis

In [ ]:

```
# coorelation between the columns diagnosis and the other columns
df.corr()['diagnosis'].sort_values()
```

Out[ ]:

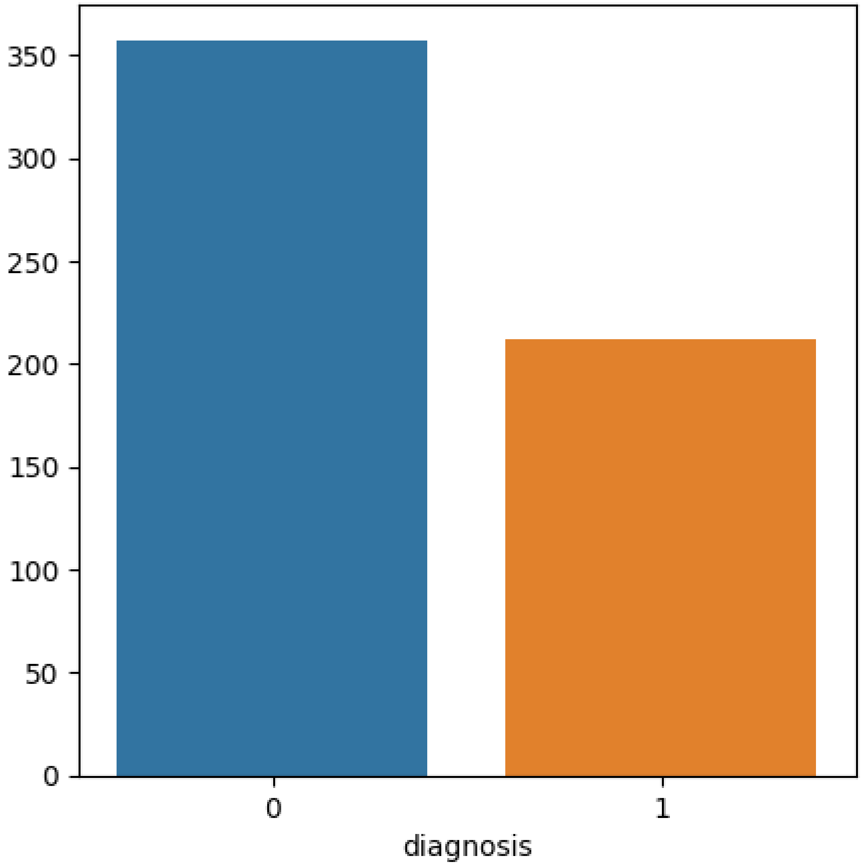
```
smoothness_se             -0.067016
fractal_dimension_mean    -0.012838
texture_se                -0.008303
symmetry_se               -0.006522
fractal_dimension_se       0.077972
concavity_se               0.253730
compactness_se             0.292999
fractal_dimension_worst    0.323872
symmetry_mean              0.330499
smoothness_mean            0.358560
concave points_se          0.408042
texture_mean               0.415185
symmetry_worst             0.416294
smoothness_worst           0.421465
texture_worst              0.456903
area_se                    0.548236
perimeter_se               0.556141
radius_se                  0.567134
compactness_worst          0.590998
compactness_mean           0.596534
concavity_worst            0.659610
concavity_mean             0.696360
area_mean                  0.708984
radius_mean                0.730029
area_worst                 0.733825
perimeter_mean             0.742636
radius_worst               0.776454
concave points_mean        0.776614
perimeter_worst            0.782914
concave points_worst       0.793566
diagnosis                  1.000000
Name: diagnosis, dtype: float64
```

In [ ]:

```
# bar plot for the number of diagnosis
plt.figure(figsize=(5,5))
sns.barplot(x=df['diagnosis'].value_counts().index,y=df['diagnosis'].value_count
```

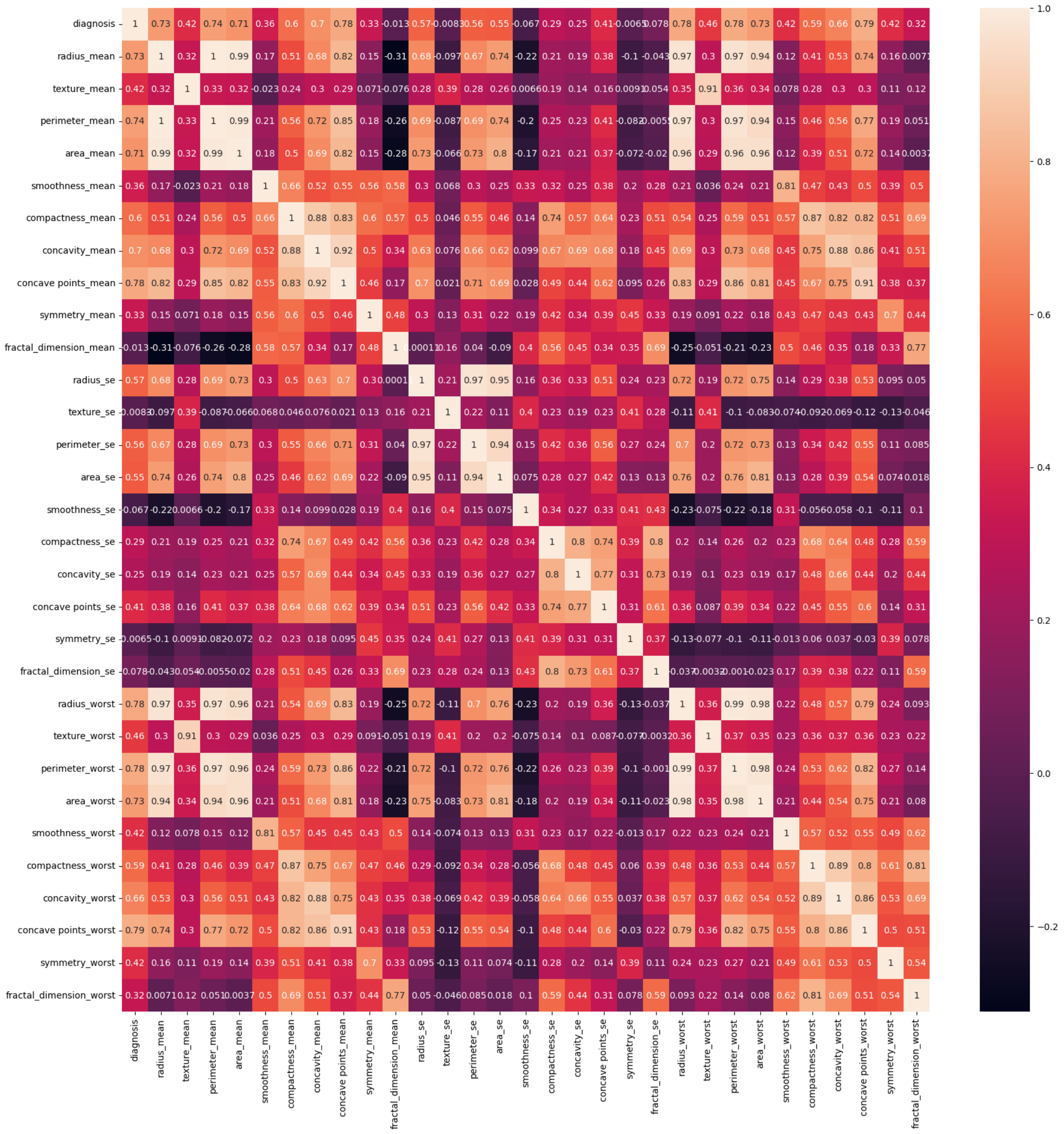
Out[ ]:

```
<Axes: xlabel='diagnosis'>
```



```
In [ ]: # create a heatmap to check the correlation
        plt.figure(figsize=(20,20))
        sns.heatmap(df.corr(),annot=True)

Out[ ]: <Axes: >
```



# Train Test Split

```
In [ ]: from sklearn.model_selection import train_test_split
        X_train,X_test,y_train,y_test = train_test_split(df.drop(['diagnosis'],axis=1),c
```

# Using Decision Tree Classifier

```
In [ ]: from sklearn.tree import DecisionTreeClassifier
        dtree = DecisionTreeClassifier()
        dtree.fit(X_train,y_train)
```

```
Out[ ]: ▾ DecisionTreeClassifier

        DecisionTreeClassifier()
```

```
In [ ]: #predicting the diagnosis
        y_pred = dtree.predict(X_test)
```

# Model Evaluation

In [ ]:

```
# printing samples from predicted and actual values
print('Predicted values: ',y_pred[:10])
print('Actual values: ',y_test[:10])
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [ ]:

```
# model evaluation
print(dtree.score(X_test,y_test))
```

```
0.935672514619883
```

# Using logistic regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2k
fra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_lo
gistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]:

```
▾ LogisticRegression

LogisticRegression()
```

In [ ]:

```
yhat = logmodel.predict(X_test)
```

# Model Evaluation

In [ ]:

```
# printing samples from predicted and actual values
print('Predicted values: ',yhat[:10])
print('Actual values: ',y_test[:10])
```

```
Predicted values:  ['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B']
Actual values:  204    B
70     M
131    M
431    B
540    B
567    M
369    M
29     M
81     B
477    B
Name: diagnosis, dtype: object
```

In [ ]:

```
# model evaluation
print(logmodel.score(X_test,y_test))
```

```
0.9707602339181286
```

# Conclusion

From both the models we can see that the accuracy is 93.5% and 97% respectively. But we can see that the recall value for the logistic regression is 97% which is better than the decision tree classifier. So we can say that the logistic regression is better than the decision tree classifier.