ASSIGNMENT 4 - CLASSIFICATION

The dataset "Breast Cancer" contains various features related to cell measurements, including characteristics such as radius, texture, smoothness, and compactness, along with a target variable indicating whether the tumor is malignant or benign. The primary goal of this project is to design and implement a comprehensive classification system that addresses key challenges in classifying tumors, such as feature scaling, model selection, and handling class imbalance. By applying effective classification algorithms, the objective is to analyze and predict whether a tumor is malignant or benign based on the provided features, ultimately enhancing the overall quality, reliability, and usability of the model for further analysis and machine learning applications. This task will focus on implementing and comparing multiple classification techniques to determine the best model for tumor classification.

SOURCE

The Breast Cancer dataset used for this project is available in the sklearn library. It can be loaded using the `load_breast_cancer()` function from `sklearn.datasets`.

IMPORTING MODULES

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

LOADING & PREPROCESSING

1. LOAD THE DATA AND CONVERT INTO DATA FRAME

In [6]:
```
# LOAD THE DATASET
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()

# Convert to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Display the first few rows
print(df.head())
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840   
1        20.57         17.77          132.90     1326.0          0.08474   
2        19.69         21.25          130.00     1203.0          0.10960   
3        11.42         20.38           77.58      386.1          0.14250   
4        20.29         14.34          135.10     1297.0          0.10030   

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419   
1           0.07864          0.0869              0.07017         0.1812   
2           0.15990          0.1974              0.12790         0.2069   
3           0.28390          0.2414              0.10520         0.2597   
4           0.13280          0.1980              0.10430         0.1809   

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0   
1                 0.05667  ...          23.41           158.80      1956.0   
2                 0.05999  ...          25.53           152.50      1709.0   
3                 0.09744  ...          26.50            98.87       567.7   
4                 0.05883  ...          16.67           152.20      1575.0   

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   worst symmetry  worst fractal dimension  target  
0          0.4601                  0.11890       0  
1          0.2750                  0.08902       0  
2          0.3613                  0.08758       0  
3          0.6638                  0.17300       0  
4          0.2364                  0.07678       0  

[5 rows x 31 columns]
```

2. DISPLAY FIRST & LAST ROWS

In [12]:
```
# DISPLAY FIRST FEW ROWS TO UNDERSTAND THE STRUCTURE OF THE DATA
print(df.head())
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840   
1        20.57         17.77          132.90     1326.0          0.08474   
2        19.69         21.25          130.00     1203.0          0.10960   
3        11.42         20.38           77.58      386.1          0.14250   
4        20.29         14.34          135.10     1297.0          0.10030   

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419   
1           0.07864          0.0869              0.07017         0.1812   
2           0.15990          0.1974              0.12790         0.2069   
3           0.28390          0.2414              0.10520         0.2597   
4           0.13280          0.1980              0.10430         0.1809   

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0   
1                 0.05667  ...          23.41           158.80      1956.0   
2                 0.05999  ...          25.53           152.50      1709.0   
3                 0.09744  ...          26.50            98.87       567.7   
4                 0.05883  ...          16.67           152.20      1575.0   

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   worst symmetry  worst fractal dimension  target  
0          0.4601                  0.11890       0  
1          0.2750                  0.08902       0  
2          0.3613                  0.08758       0  
3          0.6638                  0.17300       0  
4          0.2364                  0.07678       0  

[5 rows x 31 columns]
```

In [10]:
```
# DISPLAY LAST FEW ROWS TO UNDERSTAND THE STRUCTURE OF THE DATA
```

```
print(df.tail())
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100   
565        20.13         28.25          131.20     1261.0          0.09780   
566        16.60         28.08          108.30      858.1          0.08455   
567        20.60         29.33          140.10     1265.0          0.11780   
568         7.76         24.54           47.92      181.0          0.05263   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726   
565           0.10340         0.14400              0.09791         0.1752   
566           0.10230         0.09251              0.05302         0.1590   
567           0.27700         0.35140              0.15200         0.2397   
568           0.04362         0.00000              0.00000         0.1587   

     mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
564                 0.05623  ...          26.40           166.10      2027.0   
565                 0.05533  ...          38.25           155.00      1731.0   
566                 0.05648  ...          34.12           126.70      1124.0   
567                 0.07016  ...          39.42           184.60      1821.0   
568                 0.05884  ...          30.37            59.16       268.6   

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107   
565           0.11660            0.19220           0.3215   
566           0.11390            0.30940           0.3403   
567           0.16500            0.86810           0.9387   
568           0.08996            0.06444           0.0000   

     worst concave points  worst symmetry  worst fractal dimension  target  
564                0.2216          0.2060                  0.07115       0  
565                0.1628          0.2572                  0.06637       0  
566                0.1418          0.2218                  0.07820       0  
567                0.2650          0.4087                  0.12400       0  
568                0.0000          0.2871                  0.07039       1  

[5 rows x 31 columns]
```

3. DATATYPE OF EACH COLUMN

In [14]:
```
# DISPLAY DATA TYPE OF EACH COLUMN
print("Dataset Info:")
df.info()
```

```
Dataset 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  target                   569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

4. STATISTICAL SUMMARY OF DATA

In [16]:
```python
# DISPLAY STATISTICAL SUMMARY
print("Statistical Summary:")
df.describe()
```

```
Statistical Summary:
```

Out[16]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	me frac dimens
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097

8 rows × 31 columns

5. DISPLAY ALL COLUMN NAMES

In [18]:
```python
# DISPLAY PARTICULAR COLUMN
print("Columns of the dataset:")
df.columns
```

```
Columns of the dataset:
```

Out[18]:
```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

6. NULL / MISSING VALUES IN EACH COLUMN

In [20]:
```
# DISPLAY NULL VALUES IN EACH COLUMN
print("Null values in each column:")
print(df.isnull().sum())
```

```
Null values in each column:
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

7. DUPLICATE VALUES

In [22]:
```
# FINDING THE TOTAL NO OF DUPLICATES
df.duplicated().sum()
```

Out[22]:
```
0
```

8. FEATURE SCALING

In [26]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X = df.drop(columns=['target'])
y = df['target']

X_scaled = scaler.fit_transform(X)

print("\nScaled Feature Data (First 5 rows):")
print(X_scaled[:5])
```

```
Scaled Feature Data (First 5 rows):
[[ 1.09706398e+00 -2.07333501e+00  1.26993369e+00  9.84374905e-01
   1.56846633e+00  3.28351467e+00  2.65287398e+00  2.53247522e+00
   2.21751501e+00  2.25574689e+00  2.48973393e+00 -5.65265059e-01
   2.83303087e+00  2.48757756e+00 -2.14001647e-01  1.31686157e+00
   7.24026158e-01  6.60819941e-01  1.14875667e+00  9.07083081e-01
   1.88668963e+00 -1.35929347e+00  2.30360062e+00  2.00123749e+00
   1.30768627e+00  2.61666502e+00  2.10952635e+00  2.29607613e+00
   2.75062224e+00  1.93701461e+00]
 [ 1.82982061e+00 -3.53632408e-01  1.68595471e+00  1.90870825e+00
  -8.26962447e-01 -4.87071673e-01 -2.38458552e-02  5.48144156e-01
   1.39236330e-03 -8.68652457e-01  4.99254601e-01 -8.76243603e-01
   2.63326966e-01  7.42401948e-01 -6.05350847e-01 -6.92926270e-01
  -4.40780058e-01  2.60162067e-01 -8.05450380e-01 -9.94437403e-02
   1.80592744e+00 -3.69203222e-01  1.53512599e+00  1.89048899e+00
  -3.75611957e-01 -4.30444219e-01 -1.46748968e-01  1.08708430e+00
  -2.43889668e-01  2.81189987e-01]
 [ 1.57988811e+00  4.56186952e-01  1.56650313e+00  1.55888363e+00
   9.42210440e-01  1.05292554e+00  1.36347845e+00  2.03723076e+00
   9.39684817e-01 -3.98007910e-01  1.22867595e+00 -7.80083377e-01
   8.50928301e-01  1.18133606e+00 -2.97005012e-01  8.14973504e-01
   2.13076435e-01  1.42482747e+00  2.37035535e-01  2.93559404e-01
   1.51187025e+00 -2.39743838e-02  1.34747521e+00  1.45628455e+00
   5.27407405e-01  1.08293217e+00  8.54973944e-01  1.95500035e+00
   1.15225500e+00  2.01391209e-01]
 [-7.68909287e-01  2.53732112e-01 -5.92687167e-01 -7.64463792e-01
   3.28355348e+00  3.40290899e+00  1.91589718e+00  1.45170736e+00
   2.86738293e+00  4.91091929e+00  3.26373441e-01 -1.10409044e-01
   2.86593405e-01 -2.88378148e-01  6.89701660e-01  2.74428041e+00
   8.19518384e-01  1.11500701e+00  4.73268037e+00  2.04751088e+00
  -2.81464464e-01  1.33984094e-01 -2.49939304e-01 -5.50021228e-01
   3.39427470e+00  3.89339743e+00  1.98958826e+00  2.17578601e+00
   6.04604135e+00  4.93501034e+00]
 [ 1.75029663e+00 -1.15181643e+00  1.77657315e+00  1.82622928e+00
   2.80371830e-01  5.39340452e-01  1.37101143e+00  1.42849277e+00
  -9.56046689e-03 -5.62449981e-01  1.27054278e+00 -7.90243702e-01
   1.27318941e+00  1.19035676e+00  1.48306716e+00 -4.85198799e-02
   8.28470780e-01  1.14420474e+00 -3.61092272e-01  4.99328134e-01
   1.29857524e+00 -1.46677038e+00  1.33853946e+00  1.22072425e+00
   2.20556166e-01 -3.13394511e-01  6.13178758e-01  7.29259257e-01
  -8.68352984e-01 -3.97099619e-01]]
```

9. SPLITTING THE DATA INTO TRAINING AND TESTING SET

In [28]:
```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Display the shape of the split data
print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")
```

```
Training data shape: (398, 30)
Testing data shape: (171, 30)
```

Preprocessing steps with explanations:

1. *Load the Data*:

    - The dataset is loaded using load_breast_cancer() from sklearn.datasets. This gives us the feature data (X) and the target data (y), which indicates whether a tumor is malignant (1) or benign (0).

2. *Convert to DataFrame*:

    - Converted the data into pandas DataFrames for easier manipulation and analysis.

3. *Display First and Last Rows*:

    - Displayed the first few rows to understand the data structure and confirm it loaded correctly.

4. *Check Data Types*:

- Used `info()` to check the data types of the columns and ensure they are as expected (numerical values).

5. *Statistical Summary*:

   - Used `describe()` to view statistics (mean, min, max, etc.) of both features and target to understand their distribution.

6. *Display Column Names*:

   - Printed the column names of the features to know what variables we are working with.

7. *Check for Missing Values*:

   - Checked for missing values with `isnull().sum()` to ensure the dataset is complete.

8. *Find Duplicate Rows*:

   - Checked for duplicate rows using `duplicated().sum()` to ensure there are no repeated records.

9. *Feature Scaling*:

   - Scaled the features using `StandardScaler` to ensure that all features are on the same scale, which is important for some machine learning models.

10. *Train-Test Split*:

- Split the data into training and testing sets to evaluate the model's performance on unseen data.

These steps are necessary to clean and prepare the data for better model performance.

CLASSIFICATION ALGORITHMS IMPLEMENTATION

1. LINEAR REGRESSION ALGORITHM

In [37]:
```
from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()

lr_model.fit(X_train, y_train)

y_pred = lr_model.predict(X_test)
y_pred
```

Out[37]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1])
```

Logistic Regression works by estimating the probability of a binary outcome based on input features, using a logistic function. It assumes a linear relationship between the features (such as mean radius, texture, smoothness, etc.) and the target variable (malignant or benign tumor). This model is suitable for the Breast Cancer dataset because factors like cell characteristics likely have a linear influence on the likelihood of a tumor being malignant or benign, making it an appropriate choice for classifying tumor types.

2. DECISION TREE CLASSIFIER ALGORITHM

In [44]:
```
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()

dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)
y_pred_dt
```

Out[44]:
```
array([1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1])
```

The Decision Tree Classifier works by recursively splitting the data based on feature values to maximize information gain and minimize impurity within each subset. It does not assume a linear relationship between the target variable (malignant or benign tumor) and the input features (such as radius, texture, smoothness, etc.).

This model is suitable for the Breast Cancer dataset because it can capture non-linear relationships and complex interactions between features, such as how various cell characteristics might jointly influence the likelihood of a tumor being malignant or benign in ways that a linear model cannot.

3. RANDOM FOREST CLASSIFIER ALGORITHM

In [55]:
```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)
y_pred_rf
```

Out[55]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1])
```

The Random Forest Classifier is an ensemble learning method that constructs multiple decision trees and combines their predictions to enhance accuracy and reduce overfitting. It effectively handles complex, non-linear relationships between features and the target variable. This makes it suitable for the Breast Cancer dataset, as it can capture intricate interactions between factors like cell radius, texture, smoothness, and compactness, while providing robust predictions and insights into feature importance for classifying tumors as malignant or benign.

4. K NEAREST NEIGHBOUR CLASSIFIER ALGORITHM

In [62]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5)

knn_model.fit(X_train, y_train)

y_pred_knn = knn_model.predict(X_test)
y_pred_knn
```

Out[62]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1])
```

The k-Nearest Neighbors (k-NN) Classifier is a simple, instance-based learning algorithm that classifies data points based on the majority class of their nearest neighbors. It computes the distance between the input data and other points in the feature space to make predictions. This method is well-suited for the Breast Cancer dataset as it can effectively capture complex, non-linear relationships between features like cell radius, texture, and smoothness, which are important for classifying tumors as malignant or benign. The flexibility of k-NN to handle varied data patterns makes it a robust choice for classification tasks in this dataset.

5. SUPPORT VECTOR CLASSIFIER ALGORITHM

In [68]:
```
from sklearn.svm import SVC

svc_model = SVC(kernel='linear')

svc_model.fit(X_train, y_train)

y_pred_svc = svc_model.predict(X_test)
y_pred_svc
```

Out[68]:
```
array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1])
```

The Support Vector Classifier (SVC) is a classification model that finds a hyperplane that best separates the data into different classes, focusing on maximizing the margin between data points of different classes. It can handle both linear and non-linear decision boundaries by applying kernel functions, such as the Radial Basis Function (RBF). SVC is suitable for the Breast Cancer dataset because it can effectively capture complex, non-linear relationships between features like cell texture, radius, and smoothness, while also being robust to outliers and effective in high-dimensional spaces.

MODEL EVALUATION

1. LOGISTIC REGRESSION MODEL EVALUATION

In [73]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Accuracy: {accuracy * 100:.2f}%")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Classification Report
report = classification_report(y_test, y_pred)
print("\nClassification Report:")
print(report)
```

```
Logistic Regression Accuracy: 98.25%

Confusion Matrix:
[[ 62   1]
 [  2 106]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98        63
           1       0.99      0.98      0.99       108

    accuracy                           0.98       171
   macro avg       0.98      0.98      0.98       171
weighted avg       0.98      0.98      0.98       171
```

2. DECISION TREE CLASSIFIER MODEL EVALUATION

In [47]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate the accuracy of the model
accuracy_dtc = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Classifier Accuracy: {accuracy_dtc * 100:.2f}%")

#  Confusion Matrix
cm_dtc = confusion_matrix(y_test, y_pred_dt)
print("\nConfusion Matrix:")
print(cm_dtc)

# Classification Report
report_dtc = classification_report(y_test, y_pred_dt)
print("\nClassification Report:")
print(report_dtc)
```

```
Decision Tree Classifier Accuracy: 92.98%

Confusion Matrix:
[[60  3]
 [ 9 99]]

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.95      0.91        63
           1       0.97      0.92      0.94       108

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.93       171
weighted avg       0.93      0.93      0.93       171
```

3. RANDOM FOREST CLASSIFIER MODEL EVALUATION

In [75]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate the accuracy of the model
accuracy_rfc = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Classifier Accuracy: {accuracy_rfc * 100:.2f}%")

#  Confusion Matrix
cm_rfc = confusion_matrix(y_test, y_pred_rf)
print("\nConfusion Matrix:")
print(cm_rfc)

# Classification Report
report_rfc = classification_report(y_test, y_pred_rf)
print("\nClassification Report:")
print(report_rfc)
```

```
Random Forest Classifier Accuracy: 97.08%

Confusion Matrix:
[[ 59   4]
 [  1 107]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        63
           1       0.96      0.99      0.98       108

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

4. K NEAREST NEIGHBOUR CLASSIFIER MODEL EVALUATION

In [77]:

```python
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate the accuracy of the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"KNN Classifier Accuracy: {accuracy_knn * 100:.2f}%")

#  Confusion Matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
print("\nConfusion Matrix:")
print(cm_knn)

# Classification Report
report_knn = classification_report(y_test, y_pred_knn)
print("\nClassification Report:")
print(report_knn)
```

```
KNN Classifier Accuracy: 95.91%

Confusion Matrix:
[[ 59   4]
 [  3 105]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94        63
           1       0.96      0.97      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171
```

5. SUPPORT VECTOR CLASSIFIER MODEL EVALUATION

In [79]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate the accuracy of the model
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f"Support Vector Classifier Accuracy: {accuracy_svc * 100:.2f}%")

#  Confusion Matrix
cm_svc = confusion_matrix(y_test, y_pred_svc)
print("\nConfusion Matrix:")
print(cm_svc)

# Classification Report
report_svc = classification_report(y_test, y_pred_svc)
print("\nClassification Report:")
print(report_svc)
```

```
Support Vector Classifier Accuracy: 97.66%

Confusion Matrix:
[[ 61   2]
 [  2 106]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        63
           1       0.98      0.98      0.98       108

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

Summary of Best and Worst-Performing Models

Best-Performing Model:

The Random Forest Classifier is the best-performing model with the highest accuracy (97.08%) and the best balance in F1-scores across both classes. This indicates that it provides the most accurate predictions and captures complex, non-linear relationships within the data effectively.

Worst-Performing Model:

The Decision Tree Classifier performs the worst among the models tested, with the lowest accuracy (92.98%) and relatively lower F1-scores, particularly for class 0.

While it performs decently for class 1, its overall accuracy and precision/recall balance are weaker compared to the other models.

CONCLUSION

The Random Forest Classifier is the best model for the Breast Cancer dataset because it performs well and effectively handles the complex, non-linear relationships between the features. In contrast, Logistic Regression is the least effective model, likely due to its assumption of a linear decision boundary, which does not capture the intricate patterns within the data.

In [ ]:

```
```