Dataset:

Use the breast cancer dataset available in the sklearn library.

In [13]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd

# Load the breast cancer dataset
data = load_breast_cancer()

# Create a DataFrame with the feature data
df = pd.DataFrame(data.data, columns=data.feature_names)

# Add the target variable to the DataFrame
df['target'] = data.target

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

# Get a summary of the dataset
print(df.())
```

```
   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]
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038
```

```
803
min         0.052630        0.019380       0.000000          0.000
000
25%         0.086370        0.064920       0.029560          0.020
310
50%         0.095870        0.092630       0.061540          0.033
500
75%         0.105300        0.130400       0.130700          0.074
000
max         0.163400        0.345400       0.426800          0.201
200

       mean symmetry  mean fractal dimension  ...  worst texture  \
count     569.000000              569.000000  ...     569.000000
mean        0.181162                0.062798  ...      25.677223
std         0.027414                0.007060  ...       6.146258
min         0.106000                0.049960  ...      12.020000
25%         0.161900                0.057700  ...      21.080000
50%         0.179200                0.061540  ...      25.410000
75%         0.195700                0.066120  ...      29.720000
max         0.304000                0.097440  ...      49.540000

       worst perimeter  worst area  worst smoothness  worst compactness
\
count       569.000000  569.000000        569.000000         569.000000
mean        107.261213  880.583128          0.132369           0.254265
std          33.602542  569.356993          0.022832           0.157336
min          50.410000  185.200000          0.071170           0.027290
25%          84.110000  515.300000          0.116600           0.147200
50%          97.660000  686.500000          0.131300           0.211900
75%         125.400000 1084.000000          0.146000           0.339100
max         251.200000 4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

In [ ]:

```

```

Q1 : Loading and Preprocessing :

a.Load the breast cancer dataset from sklearn.

b.Preprocess the data to handle any missing values and perform necessary feature scaling.
c.Explain the preprocessing steps you performed and justify why they are necessary for this dataset.

In [4]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Load the breast cancer dataset
data = load_breast_cancer()

# Convert the data to a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)

# Add the target variable to the DataFrame
df['target'] = data.target

# Check for missing values
missing_values = df.isnull().sum()
print("Missing values in each feature:\n", missing_values)

# Perform feature scaling
scaler = StandardScaler()
df[data.feature_names] = scaler.fit_transform(df[data.feature_names])

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

```
Missing values in each feature:
 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
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  
\
0     1.097064     -2.073335        1.269934   0.984375         1.568466
1     1.829821     -0.353632        1.685955   1.908708        -0.826962
2     1.579888      0.456187        1.566503   1.558884         0.942210
3    -0.768909      0.253732       -0.592687  -0.764464         3.283553
4     1.750297     -1.151816        1.776573   1.826229         0.280372

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0          3.283515        2.652874             2.532475       2.217515
1         -0.487072       -0.023846             0.548144       0.001392
2          1.052926        1.363478             2.037231       0.939685
3          3.402909        1.915897             1.451707       2.867383
4          0.539340        1.371011             1.428493      -0.009560

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  
\
0                2.255747  ...      -1.359293         2.303601    2.001237
1               -0.868652  ...      -0.369203         1.535126    1.890489
2               -0.398008  ...      -0.023974         1.347475    1.456285
3                4.910919  ...       0.133984        -0.249939   -0.550021
4               -0.562450  ...      -1.466770         1.338539    1.220724

   worst smoothness  worst compactness  worst concavity  worst concave poi
nts  \
0          1.307686           2.616665         2.109526              2.296
076
1         -0.375612          -0.430444        -0.146749              1.087
```

```
084
2           0.527407           1.082932          0.854974              1.955
000
3           3.394275           3.893397          1.989588              2.175
786
4           0.220556          -0.313395          0.613179              0.729
259

   worst symmetry  worst fractal dimension  target
0        2.750622                 1.937015       0
1       -0.243890                 0.281190       0
2        1.152255                 0.201391       0
3        6.046041                 4.935010       0
4       -0.868353                -0.397100       0

[5 rows x 31 columns]
```

In [ ]:

2. Classification Algorithm Implementation

Implement the following five classification algorithms:

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier
4. Support Vector Machine (SVM)
5. k-Nearest Neighbors (k-NN)

For each algorithm, provide a brief description of how it works and why it might be suitable for this dataset.

Logistic Regression

In [5]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names],

# Initialize and train the Logistic Regression model
lr_model = LogisticRegression(max_iter=10000)
lr_model.fit(X_train, y_train)

# Make predictions and evaluate the model
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy:.4f}")
```

```
Logistic Regression Accuracy: 0.9737
```

Decision Tree Classifier

In [6]:
```
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions and evaluate the model
dt_predictions = dt_model.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
```

```
Decision Tree Accuracy: 0.9474
```

Random Forest Classifier

In [7]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions and evaluate the model
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy:.4f}")
```

```
Random Forest Accuracy: 0.9649
```

Support Vector Machine (SVM)

In [8]:
```
from sklearn.svm import SVC

# Initialize and train the SVM model
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

# Make predictions and evaluate the model
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f"SVM Accuracy: {svm_accuracy:.4f}")
```

```
SVM Accuracy: 0.9561
```

k-Nearest Neighbors (k-NN)

In [10]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
data = load_breast_cancer()

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names],

# Perform feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Ensure X_test is a NumPy array (just in case)
X_test = np.array(X_test)

# Initialize and train the k-NN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Make predictions and evaluate the model
knn_predictions = knn_model.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print(f"k-NN Accuracy: {knn_accuracy:.4f}")
```

```
k-NN Accuracy: 0.9474
```

3. Model Comparison :

Compare the performance of the five classification algorithms. Which algorithm performed the best and which one performed the worst?

In [11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the dataset
data = load_breast_cancer()

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

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names],

# Perform feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the models
models = {
    "Logistic Regression": LogisticRegression(max_iter=10000, random_state=
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(kernel='linear', random_state=42),
    "k-NN": KNeighborsClassifier(n_neighbors=5)
}

# Train, predict, and evaluate each model
accuracies = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    accuracies[model_name] = accuracy
    print(f"{model_name} Accuracy: {accuracy:.4f}")

# Find the best and worst performing models
best_model = max(accuracies, key=accuracies.get)
worst_model = min(accuracies, key=accuracies.get)

print("\nBest Performing Model:")
print(f"{best_model} with accuracy {accuracies[best_model]:.4f}")

print("\nWorst Performing Model:")
print(f"{worst_model} with accuracy {accuracies[worst_model]:.4f}")
```

```
Logistic Regression Accuracy: 0.9737
Decision Tree Accuracy: 0.9474
Random Forest Accuracy: 0.9649
SVM Accuracy: 0.9561
k-NN Accuracy: 0.9474

Best Performing Model:
Logistic Regression with accuracy 0.9737

Worst Performing Model:
Decision Tree with accuracy 0.9474
```

In [ ]:

In [ ]:

In [ ]: