```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyt
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fil

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preser
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
```

```
/kaggle/input/cancer/wdbc.names
/kaggle/input/cancer/wdbc.data
```

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, cc
```

```
columns = [
    "id", "diagnosis",
    "radius_mean", "texture_mean", "perimeter_mean", "area_mean", "smoothness_mean",
    "compactness_mean", "concavity_mean", "concave_points_mean", "symmetry_mean", "frac
    "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_se",
    "compactness_se", "concavity_se", "concave_points_se", "symmetry_se", "fractal_dime
    "radius_worst", "texture_worst", "perimeter_worst", "area_worst", "smoothness_worst
    "compactness_worst", "concavity_worst", "concave_points_worst", "symmetry_worst", "
]
```

```
df = pd.read_csv(
    "/kaggle/input/cancer/wdbc.data",
    header=None,
    names=columns
)

print("Dataset shape:", df.shape)
df.head()
```

```
Dataset shape: (569, 32)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
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 × 32 columns

```
# Convert diagnosis to string and remove spaces
df["diagnosis"] = df["diagnosis"].astype(str).str.strip()

# Keep ONLY valid labels
df = df[df["diagnosis"].isin(["M", "B"])]

# Encode target
df["diagnosis"] = df["diagnosis"].map({"M": 1, "B": 0}).astype(int)


# Verify labels
print("Label distribution:")
print(df["diagnosis"].value_counts())
print("Total rows after cleaning:", df.shape[0])
```

```
Label distribution:
diagnosis
0    357
1    212
Name: count, dtype: int64
Total rows after cleaning: 569
```

```
X = df.drop(["id", "diagnosis"], axis=1)
y = df["diagnosis"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train set:", X_train.shape)
print("Test set:", X_test.shape)
```

```
Train set: (455, 30)
Test set: (114, 30)
```

```
# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
log_model = LogisticRegression(max_iter=500)
log_model.fit(X_train_scaled, y_train)

# Predictions
y_train_pred_log = log_model.predict(X_train_scaled)
y_test_pred_log = log_model.predict(X_test_scaled)

# Metrics
print("LOGISTIC REGRESSION RESULTS")
print("Train Error:", 1 - accuracy_score(y_train, y_train_pred_log))
print("Test Error:", 1 - accuracy_score(y_test, y_test_pred_log))
print("Accuracy:", accuracy_score(y_test, y_test_pred_log))
print("Precision:", precision_score(y_test, y_test_pred_log))
print("Recall:", recall_score(y_test, y_test_pred_log))
print("F1 Score:", f1_score(y_test, y_test_pred_log))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_log))
```

```
LOGISTIC REGRESSION RESULTS
Train Error: 0.01318681318681314
Test Error: 0.03508771929824561
Accuracy: 0.9649122807017544
Precision: 0.975
Recall: 0.9285714285714286
F1 Score: 0.9512195121951219
Confusion Matrix:
 [[71  1]
 [ 3 39]]
```

```
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(X_train, y_train)

y_train_pred_tree = tree_model.predict(X_train)
y_test_pred_tree = tree_model.predict(X_test)

print("DECISION TREE RESULTS")
print("Train Error:", 1 - accuracy_score(y_train, y_train_pred_tree))
print("Test Error:", 1 - accuracy_score(y_test, y_test_pred_tree))
print("Accuracy:", accuracy_score(y_test, y_test_pred_tree))
print("Precision:", precision_score(y_test, y_test_pred_tree))
print("Recall:", recall_score(y_test, y_test_pred_tree))
print("F1 Score:", f1_score(y_test, y_test_pred_tree))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred_tree))
```

```
DECISION TREE RESULTS
Train Error: 0.0
Test Error: 0.07017543859649122
Accuracy: 0.9298245614035088
Precision: 0.9047619047619048
Recall: 0.9047619047619048
F1 Score: 0.9047619047619048
Confusion Matrix:
 [[68  4]
 [ 4 38]]
```

```
comparison = pd.DataFrame({
    "Model": ["Logistic Regression", "Decision Tree"],
    "Train Error": [
        1 - accuracy_score(y_train, y_train_pred_log),
        1 - accuracy_score(y_train, y_train_pred_tree)
    ],
    "Test Error": [
        1 - accuracy_score(y_test, y_test_pred_log),
        1 - accuracy_score(y_test, y_test_pred_tree)
    ]
})

comparison
```

	Model	Train Error	Test Error
0	Logistic Regression	0.013187	0.035088
1	Decision Tree	0.000000	0.070175