1. Importing Libraries

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

2. Data Collection and Loading

```
# Load your uploaded dataset
data_frame = pd.read_csv("Breast_Cancer_data.csv")

# Show first 5 rows
data_frame.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concav points_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

5 rows × 33 columns

```
print("\nMissing values BEFORE converting M/B:")
print(data_frame.isnull().sum())
```

```
Missing values BEFORE converting M/B:
id                           0
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
Unnamed: 32                569
dtype: int64
```

3. Drop useless column (569 NaN values)

```
data_frame = data_frame.drop(columns=['Unnamed: 32'])

print("\n---- After dropping Unnamed: 32 ----")
print(data_frame.isnull().sum())
```

```
---- After dropping Unnamed: 32 ----
id                         0
```

```
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
```

4. Inspect diagnosis values BEFORE mapping

```
print("\n---- Unique diagnosis values BEFORE cleaning ----")
print(data_frame["diagnosis"].unique())


#  Remove rows where diagnosis NOT 'M' or 'B'
```

```
---- Unique diagnosis values BEFORE cleaning ----
['M' 'B']
```

5. Remove rows where diagnosis NOT 'M' or 'B'

```
# ------------------------------------------------------------
valid_rows = data_frame["diagnosis"].isin(["M", "B"])
data_frame = data_frame[valid_rows]
```

6. Convert M/B → 1/0

```
#Convert Diagnosis Column (M/B → 1/0)
# --------------------------------------
# M = Malignant (1)
# B = Benign (0)
data_frame['diagnosis'] = data_frame['diagnosis'].map({'M': 1, 'B': 0})
```

```
data_frame.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concav points_mea
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

5 rows × 32 columns

```
print("\nDiagnosis value counts:")
print(data_frame['diagnosis'].value_counts())
```

```
Diagnosis value counts:
diagnosis
0    357
1    212
Name: count, dtype: int64
```

7. Prepare X and Y (remove id column)

```
print("\nStatistical Summary:")
print(data_frame.describe())
```

```
Statistical Summary:
                 id   diagnosis  radius_mean  texture_mean  perimeter_mean  \
count  5.690000e+02  569.000000   569.000000    569.000000      569.000000
mean   3.037183e+07    0.372583    14.127292     19.289649       91.969033
std    1.250206e+08    0.483918     3.524049      4.301036       24.298981
min    8.670000e+03    0.000000     6.981000      9.710000       43.790000
25%    8.692180e+05    0.000000    11.700000     16.170000       75.170000
50%    9.060240e+05    0.000000    13.370000     18.840000       86.240000
75%    8.813129e+06    1.000000    15.780000     21.800000      104.100000
max    9.113205e+08    1.000000    28.110000     39.280000      188.500000

         area_mean  smoothness_mean  compactness_mean  concavity_mean  \
count   569.000000       569.000000        569.000000      569.000000
mean    654.889104         0.096360          0.104341        0.088799
std     351.914129         0.014064          0.052813        0.079720
min     143.500000         0.052630          0.019380        0.000000
25%     420.300000         0.086370          0.064920        0.029560
50%     551.100000         0.095870          0.092630        0.061540
75%     782.700000         0.105300          0.130400        0.130700
max    2501.000000         0.163400          0.345400        0.426800

       concave points_mean  ...  radius_worst  texture_worst  perimeter_worst  \
count           569.000000  ...    569.000000     569.000000       569.000000
mean              0.048919  ...     16.269190      25.677223       107.261213
std               0.038803  ...      4.833242       6.146258        33.602542
min               0.000000  ...      7.930000      12.020000        50.410000
25%               0.020310  ...     13.010000      21.080000        84.110000
50%               0.033500  ...     14.970000      25.410000        97.660000
75%               0.074000  ...     18.790000      29.720000       125.400000
max               0.201200  ...     36.040000      49.540000       251.200000

        area_worst  smoothness_worst  compactness_worst  concavity_worst  \
count   569.000000        569.000000         569.000000       569.000000
mean    880.583128          0.132369           0.254265         0.272188
std     569.356993          0.022832           0.157336         0.208624
min     185.200000          0.071170           0.027290         0.000000
25%     515.300000          0.116600           0.147200         0.114500
50%     686.500000          0.131300           0.211900         0.226700
75%    1084.000000          0.146000           0.339100         0.382900
max    4254.000000          0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
75%                0.161400        0.317900                 0.092080
max                0.291000        0.663800                 0.207500

[8 rows x 32 columns]
```

```
X = data_frame.drop(columns=['id','diagnosis'])
Y = data_frame['diagnosis']
```

8. Train-test split

```
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=2
)
```

9. Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=5000)
model.fit(X_train, Y_train)
```

```
▾    LogisticRegression        i ?
LogisticRegression(max_iter=5000)
```

10. Evaluate the model

```
# ------------------------------------------------------------
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)
```

```
print("\nTraining Accuracy:", accuracy_score(Y_train, train_pred))
print("Testing Accuracy:", accuracy_score(Y_test, test_pred))
```

```
Training Accuracy: 0.9692307692307692
Testing Accuracy: 0.9298245614035088
```

11. Prediction system

```
sample_input = X.iloc[0].values.reshape(1, -1)
prediction = model.predict(sample_input)

print("\n---- Prediction Result ----")
if prediction[0] == 1:
    print("The Breast Cancer is **Malignant (M)**")
else:
    print("The Breast Cancer is **Benign (B)**")
```

```
---- Prediction Result ----
The Breast Cancer is **Malignant (M)**
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRe
  warnings.warn(
```

```
sample_input = X.iloc[[0]]   # KEEP feature names

prediction = model.predict(sample_input)

print("\n---- Prediction Result ----")
if prediction[0] == 1:
    print("The Breast Cancer is **Malignant (M)**")
else:
    print("The Breast Cancer is **Benign (B)**")
```

```
---- Prediction Result ----
The Breast Cancer is **Malignant (M)**
```