EXPERIMENT 1	Implement Linear and Logistic Regression on real world datasets

Name: Atharva Lotankar
Class: D15C; **Roll Number:** 31
Date: 22 / 01 / 2026
Subject: Machine and Deep Learning Lab

Aim: To implement and demonstrate Linear and Logistic Regression on real world datasets.

Linear Regression

Dataset Source:

The Advertising Sales Dataset is sourced from Kaggle at https://www.kaggle.com/datasets/yasserh/advertising-sales-dataset

Dataset Description:

This dataset contains 200 records of advertising budget allocations across three media channels—TV, Radio, and Newspaper—and the corresponding sales figures. It is used for regression analysis to understand how advertising spending in each channel influences sales.

- *Records*: 200
- *Features*: 3 numerical predictors + 1 numerical target
- *Missing Values*: None

Variable	Description	Data Type	Unit
TV	Advertising budget spent on TV	Numeric	Thousands of dollars
Radio	Advertising budget spent on radio	Numeric	Thousands of dollars
Newspaper	Advertising budget spent on newspaper	Numeric	Thousands of dollars
Sales	Resulting product sales volume	Numeric	Thousands of units

All features are continuous and measured in comparable units (thousands), making the dataset ready for modelling without extensive preprocessing. Exploratory analysis typically shows a strong linear relationship between TV advertising and sales, while radio and newspaper spending may exhibit more varied effects.

This dataset is widely used for teaching and applying linear regression, budget optimization, and marketing ROI analysis.

Theory:

Linear regression is a fundamental statistical method used to model the linear relationship between a dependent variable and one or more independent variables by fitting a straight line to the data, minimizing the sum of squared differences between observed and predicted values. It assumes linearity, independence of errors, homoscedasticity, and normality of residuals, making it ideal for predictive modelling in applications like sales forecasting or trend analysis.

Types of linear regression include simple linear regression, which uses one independent variable, multiple linear regression, extending to several predictors, ridge regression, adding L2 regularization to handle multicollinearity, lasso regression, using L1 regularization for feature selection, and polynomial regression, fitting higher-degree polynomials while maintaining a linear model structure.

Mathematical Formulation of the Algorithm:

The Simplest form of the Linear Regression is:

$$\boxed{y = mx + c}$$

where y is the target variable, x is the predictor, β_0 is the intercept, and β_1 is the slope.

In our csv, a Multiple Linear Regression model is applied to predict sales based on advertising budgets across three media channels. The general linear regression equation for this problem is expressed as:

$$\boxed{\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}$$

Here, $\hat{y}$ represents the predicted sales, x_1, x_2, and x_3 correspond to the advertising budgets for TV, radio, and newspaper respectively, β_0 is the intercept term, and β_1, β_2, and β_3 are the learned regression coefficients for each predictor.

*The model learns by minimizing the Mean Squared Error (MSE), which averages the squared differences between actual and predicted sales:

$$\boxed{\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where y_i is actual sales, $\hat{y}_i$ is predicted sales, and n is the number of observations. Minimizing MSE aligns predictions closely with true values.

**Root Mean Squared Error (RMSE):*

$$\boxed{\text{RMSE} = \sqrt{\text{MSE}}}$$

Square root of MSE, expressed in original sales units.

**R-squared Score (R²):*

$$\boxed{R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Proportion of variance in sales explained by the model.

**Mean Absolute Percentage Error (MAPE):*

$$\boxed{\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|}$$

Average percentage error relative to true sales.

*Together, these metrics provide a comprehensive assessment of accuracy, scale sensitivity explanatory power, and relative error magnitude.

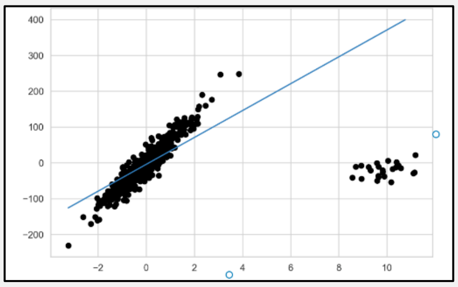
Algorithm Limitations:

Although linear regression is valued for its simplicity and interpretability, the method is subject to several constraints that may affect its suitability for certain applications.

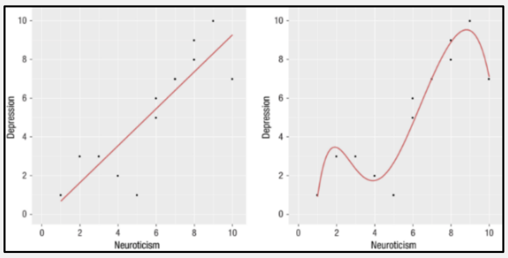
1. *Assumption of Linearity*
   The model presupposes a linear relationship between the independent and dependent

variables. It is therefore unable to adequately represent complex, non-linear patterns or interactions within the data without explicit transformation or feature engineering.

2. *Sensitivity to Outliers*
   Estimates of model parameters can be disproportionately influenced by extreme observations, leading to a biased regression line and reduced predictive accuracy on representative data.


Outlier Sensitivity in Regression Line

3. *Multicollinearity Concerns*
   When predictor variables exhibit high correlation, the resulting coefficient estimates may become unstable and difficult to interpret, potentially inflating standard errors and undermining the reliability of individual feature contributions.



Linear Regression cannot handle complex data points

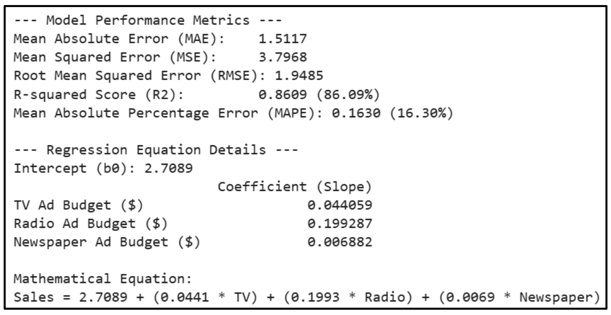
4. *Scalability Constraints*
   Performance and computational efficiency may diminish with very large sample sizes or high-dimensional feature spaces, particularly when using ordinary least squares estimation without regularization.

Code:

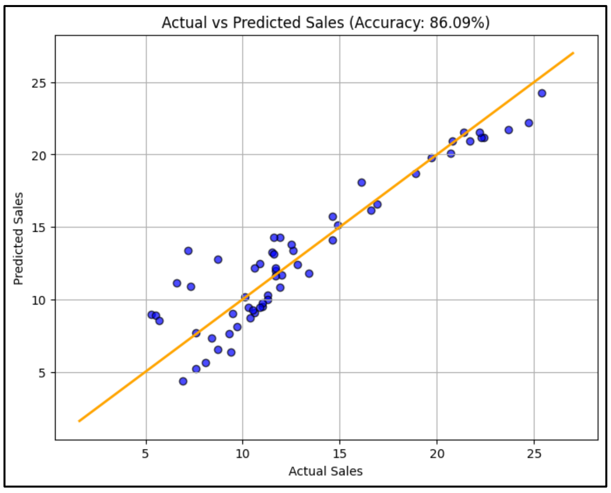
```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import
train_test_split

from sklearn.linear_model import
LinearRegression

from sklearn import metrics


# Load the Dataset

df = pd.read_csv('sales_adv.csv')


# Data Cleaning

if 'Unnamed: 0' in df.columns:

    df = df.drop(columns=['Unnamed: 0'])


# Correlation Analysis

correlation_matrix = df.corr()

# Feature Selection and Data Splitting

X = df[['TV Ad Budget ($)', 'Radio Ad Budget
($)', 'Newspaper Ad Budget ($)']]

y = df['Sales ($)']


# Splitting: 70% Training, 30% Testing

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)


# Model Training

model = LinearRegression()

model.fit(X_train, y_train)


# Predictions

y_pred = model.predict(X_test)


# Mathematical Performance Analysis

mae = metrics.mean_absolute_error(y_test,
y_pred)

mse = metrics.mean_squared_error(y_test,
y_pred)
```

```
rmse = np.sqrt(mse)

r2 = metrics.r2_score(y_test, y_pred)


# Calculate Mean Absolute Percentage Error
(MAPE) for additional % insight

mape = np.mean(np.abs((y_test - y_pred) /
y_test))


print("--- Model Performance Metrics ---")

print(f"Mean Absolute Error
(MAE):    {mae:.4f}")

print(f"Mean Squared Error
(MSE):     {mse:.4f}")

print(f"Root Mean Squared Error (RMSE):
{rmse:.4f}")

print(f"R-squared Score (R2):          {r2:.4f}
({r2 * 100:.2f}%)")

print(f"Mean Absolute Percentage Error (MAPE):
{mape:.4f} ({mape * 100:.2f}%)")

# Regression Equation and Coefficients

print("\n--- Regression Equation Details ---")

print(f"Intercept (b0):
{model.intercept_:.4f}")

# Display coefficients with percentage
equivalent for impact relative to each other
(normalized-like view)

coeffs = model.coef_

coeff_df = pd.DataFrame(coeffs, X.columns,
columns=['Coefficient (Slope)'])

print(coeff_df)


# Final Mathematical Equation representation

equation = f"Sales = {model.intercept_:.4f} +
({model.coef_[0]:.4f} * TV) +
({model.coef_[1]:.4f} * Radio) +
({model.coef_[2]:.4f} * Newspaper)"

print(f"\nMathematical Equation:\n{equation}")

# Visualizations for Model Analysis

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred, color='blue',
edgecolor='k', alpha=0.7)

plt.plot([y.min(), y.max()], [y.min(),
y.max()], color='orange', lw=2, linestyle='-')

plt.xlabel('Actual Sales')

plt.ylabel('Predicted Sales')

plt.title(f'Actual vs Predicted Sales
(Accuracy: {r2*100:.2f}%)')

plt.grid(True)

plt.savefig('actual_vs_predicted.png')
```

Output:



```
--- Model Performance Metrics ---
Mean Absolute Error (MAE):    1.5117
Mean Squared Error (MSE):     3.7968
Root Mean Squared Error (RMSE): 1.9485
R-squared Score (R2):         0.8609 (86.09%)
Mean Absolute Percentage Error (MAPE): 0.1630 (16.30%)

--- Regression Equation Details ---
Intercept (b0): 2.7089
                         Coefficient (Slope)
TV Ad Budget ($)                    0.044059
Radio Ad Budget ($)                 0.199287
Newspaper Ad Budget ($)             0.006882

Mathematical Equation:
Sales = 2.7089 + (0.0441 * TV) + (0.1993 * Radio) + (0.0069 * Newspaper)
```



This Python code loads a sales advertising dataset, performs correlation analysis, splits the data, trains a multiple linear regression model, evaluates its predictive performance using various error metrics, and outputs the final regression equation with feature coefficients.

Following to it, the graph plotted depicts the performance of the linear regression model by comparing *actual sales values on the x-axis* (The true, observed sales values in dollars from the dataset's Sales ($) column.) with the *model's predicted sales values on the y-axis* (The sales values in dollars forecasted by the linear regression model, based on the advertising budgets for TV, radio, and newspaper.).

A central orange line represents perfect predictions, where actual equals predicted. The model achieves 86.09% accuracy, with most data points clustering closely along the line, indicating a strong and consistent predictive fit.

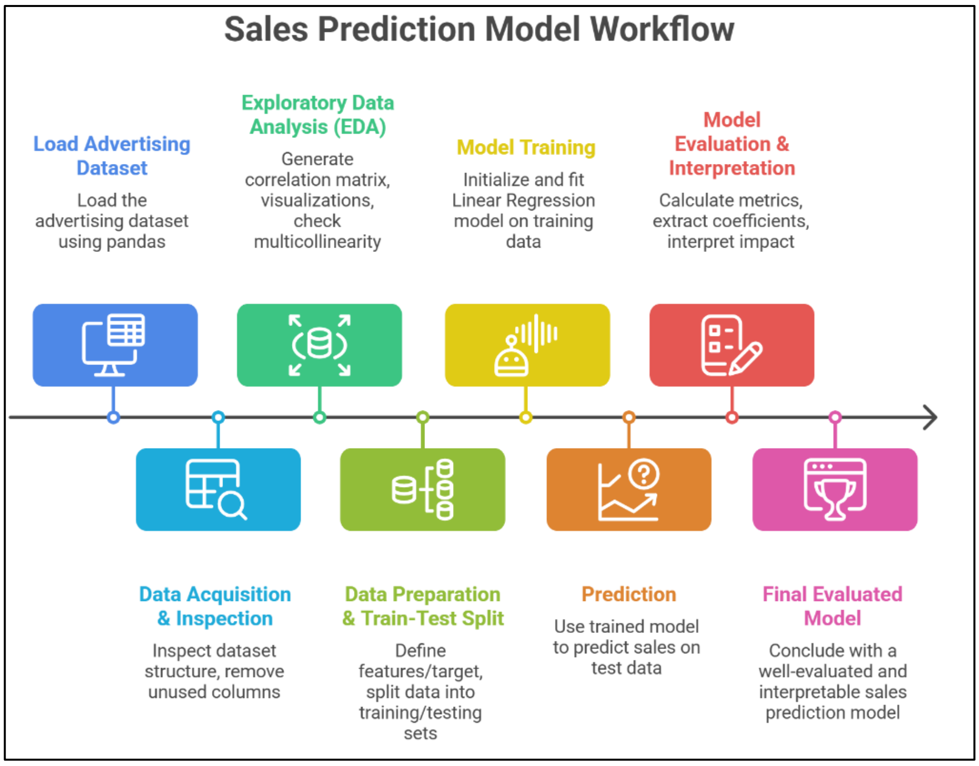
Methodology / Workflow:

This analysis follows a structured machine learning workflow to build a predictive model for sales based on advertising budgets. The process is designed to systematically progress from data ingestion to model evaluation, ensuring methodological rigor and reproducibility. It is grounded in standard supervised learning principles, emphasizing clarity at each phase.

1. *Data Acquisition and Initial Inspection*
    - Activity: The dataset is loaded using pandas.
    - Purpose: To import and perform an initial structure and integrity check, ensuring the dataset contains the required features (TV Ad Budget ($), Radio Ad Budget ($), Newspaper Ad Budget ($)) and the target variable (Sales ($)). An unused index column ('Unnamed: 0') is removed during this stage.
2. *Exploratory Data Analysis (EDA)*
    - Activity: Conduct correlation analysis via a correlation matrix and generate preliminary visualizations to understand data distributions and relationships.
    - Purpose: To identify potential linear relationships between each advertising channel and sales, and to check for multicollinearity among the independent variables before model building.
3. *Data Preparation and Splitting*
    - Activity: The feature matrix (X) is defined using the three budget columns, and the target vector (y) is defined as Sales ($). The data is then partitioned using train_test_split.
    - Parameters: A 70% training and 30% testing split is employed with a fixed random_state=42 to ensure consistent and reproducible results.
    - Rationale: This split provides a substantial dataset for training the model while reserving a statistically significant portion for unbiased evaluation.
4. *Model Training*
    - Activity: A LinearRegression model from scikit-learn is instantiated and fitted to the training data (X_train, y_train).
    - Purpose: The algorithm estimates the coefficients (slopes) and intercept of the multiple linear regression equation that minimizes the prediction error on the training set.
5. *Model Evaluation and Interpretation*
    - Activity: The trained model generates predictions (y_pred) on the unseen test set (X_test). Performance is quantified using multiple error metrics (MAE, MSE,

RMSE, R^2, MAPE). The model's final mathematical equation is extracted and displayed.

- Purpose**:** To objectively assess the model's predictive accuracy and generalizability. The coefficients provide interpretable insights into the estimated impact of each advertising channel on sales.


Sales Prediction Model Workflow
Load Advertising Dataset
Load the advertising dataset using pandas
Exploratory Data Analysis (EDA)
Generate correlation matrix, visualizations, check multicollinearity
Model Training
Initialize and fit Linear Regression model on training data
Model Evaluation & Interpretation
Calculate metrics, extract coefficients, interpret impact
Data Acquisition & Inspection
Inspect dataset structure, remove unused columns
Data Preparation & Train-Test Split
Define features/target, split data into training/testing sets
Prediction
Use trained model to predict sales on test data
Final Evaluated Model
Conclude with a well-evaluated and interpretable sales prediction model

Performance Analysis:

The model's predictive accuracy for the sales dataset is evaluated using standard regression metrics, providing a multi-faceted view of its performance.

- *The Mean Absolute Error (MAE) of $1.51* indicates the average prediction error in dollars, while the *Root Mean Squared Error (RMSE) of $1.95* offers a more sensitive measure by penalizing larger errors in the same unit.
- Most critically, an *R^2 Score of 0.8609 (86.09%)* reveals that the linear combination of TV, Radio, and Newspaper advertising budgets explains approximately **86%** of the variance in the observed Sales ($), confirming a strong linear relationship within the data.
- Additionally, a *Mean Absolute Percentage Error (MAPE) of 16.30%* quantifies the average prediction error as a percentage of actual sales.

This analysis helps quantify the model's reliability for forecasting sales based on future advertising spend, while the derived equation (Sales = 2.7089 + (0.0441 * TV) + (0.1993 * Radio) + (0.0069 * Newspaper)) provides actionable insights into the relative impact of each advertising channel.

Hyperparameter Tuning:

Hyperparameter tuning optimises an algorithm's learning behaviour by manually adjusting parameters set before training, such as regularization strength, to enhance accuracy. However, the classical Ordinary Least Squares (OLS) Linear Regression model used in this experiment has no tuneable hyperparameters; it calculates coefficients directly via a mathematical formula.

Consequently, tuning was not performed, and model performance depends entirely on the linear relationships within the advertising budget and sales data.

Logistic Regression

Dataset Source:

The Breast Cancer Wisconsin (Diagnostic) Dataset is sourced from Kaggle at https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

Dataset Description:

The Breast Cancer Wisconsin (Diagnostic) Dataset consists of numerical features derived from the characteristics of cell nuclei in digitized images of breast tissue samples. The primary objective is binary classification to predict whether a tumor is malignant or benign.

Variable	Description	Data Type	Unit
Input Features	30 numerical measurements computed from cell nuclei, capturing physical properties (e.g., radius, texture, smoothness, compactness).	Float	Various (e.g., micrometers, unitless ratios)
diagnosis	The target classification label for each sample.	Object (String)	Category

The 30 input features are organized into three statistical categories—mean values, standard errors, and worst-case values—for ten core cell nucleus characteristics. The dataset contains 569 complete records with no missing values and is slightly imbalanced, with benign cases being more frequent than malignant cases.

Theory:

Logistic Regression is a statistical and machine learning method used for binary classification problems, where the dependent variable takes only two possible outcomes (such as 0/1, yes/no, success/failure). Unlike linear regression, logistic regression does not predict a continuous value; instead, it models the probability that a given input belongs to a particular class. It does this by forming a linear combination of input features and then transforming that result so the output lies between 0 and 1. The model parameters are typically estimated using maximum likelihood estimation, and the final prediction is obtained by applying a decision threshold (commonly 0.5) to the predicted probability.

The sigmoid (logistic) function is the core component that enables logistic regression to work for classification. It takes any real-valued input and maps it smoothly into the range (0, 1), making it suitable for representing probabilities. In logistic regression, the sigmoid function is applied to the linear combination of input features to convert it into a probability estimate. This logistic usage allows the model to capture non-linear relationships between inputs and the output probability, while still maintaining a simple and interpretable structure. The S-shaped curve of the sigmoid also ensures that extreme values of the input are pushed close to 0 or 1, reinforcing confident predictions.

Mathematical Formulation of the Algorithm:

The experiment employs Logistic Regression, chosen for its interpretability and suitability for binary classification. The algorithm models the probability of a malignant tumour (class label $y = 1$) given a feature vector x using the sigmoid function:

$$\boxed{P(y = 1 \mid \mathrm{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta^T \mathbf{x})}}}$$

where β represents the model coefficients and β_0 is the intercept.

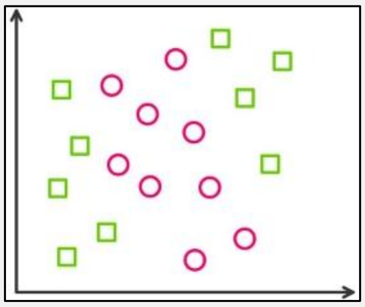
A sample is classified as malignant if $P(y = 1 \mid \mathrm{x}) \geq 0.5$. The model parameters are optimized by minimizing the binary cross-entropy loss:

$$\boxed{\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]}$$

where p_i is the predicted probability for the i-th sample, and n is the total number of samples.

Algorithm Limitations:

Logistic Regression is subject to the following constraints:



Non linear problems can't be solved with logistic regression since it has a linear decision surface

- *Linear Decision Boundary*: The model assumes a linear relationship in the log-odds space, limiting its ability to capture complex, non-linear patterns without feature engineering.
- *Sensitivity to Feature Scaling*: Performance is dependent on feature scale, requiring data normalization or standardization prior to training.
- *Assumption of Feature Independence*: The model presumes low multicollinearity; highly correlated features can produce unstable and unreliable coefficient estimates.
- *Bias in Class Imbalance*: Without corrective techniques like class weighting, the model can become biased toward the majority class in imbalanced datasets.

These limitations generally render it less suitable for problems involving highly complex or high-dimensional data structures.

(Without Hyperparameters)

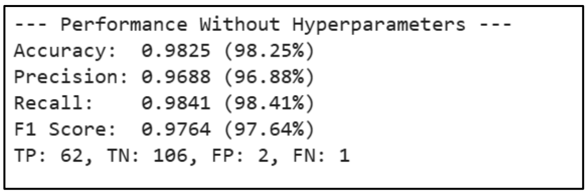
Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import
train_test_split, GridSearchCV
```

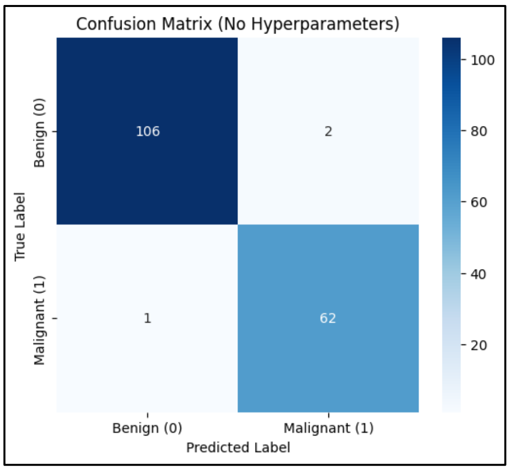
```
from sklearn.linear_model import
LogisticRegression

from sklearn.metrics import accuracy_score,
confusion_matrix, precision_score,
recall_score, f1_score

from sklearn.preprocessing import
StandardScaler


# Load the Dataset

df = pd.read_csv('breast_cancer.csv')


# Preprocessing: Drop unnecessary columns and
encode target

df = df.drop(columns=['id', 'Unnamed: 32'])

df['diagnosis'] = df['diagnosis'].map({'M': 1,
'B': 0})


X = df.drop('diagnosis', axis=1)

y = df['diagnosis']


# Train/Test Split

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)


# Scaling features for better convergence

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Model without Hyperparameters

model_default =
LogisticRegression(max_iter=10000)

model_default.fit(X_train_scaled, y_train)

y_pred_default =
model_default.predict(X_test_scaled)



# Metrics Function

def print_metrics(y_true, y_pred, title):

    cm = confusion_matrix(y_true, y_pred)

    tn, fp, fn, tp = cm.ravel()

    acc = accuracy_score(y_true, y_pred)

    prec = precision_score(y_true, y_pred)

    rec = recall_score(y_true, y_pred)

    f1 = f1_score(y_true, y_pred)

    print(f"--- {title} ---")

    print(f"Accuracy:  {acc:.4f}
({acc*100:.2f}%)")

    print(f"Precision: {prec:.4f}
({prec*100:.2f}%)")

    print(f"Recall:    {rec:.4f}
({rec*100:.2f}%)")

    print(f"F1 Score:  {f1:.4f}
({f1*100:.2f}%)")

    print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN:
{fn}\n")


print_metrics(y_test, y_pred_default,
"Performance Without Hyperparameters")

# Sigmoid Curve Visualization

# z = b0 + b1*x1 + ... (the linear combination)

z =
model_default.decision_function(X_test_scaled)

probabilities =
model_default.predict_proba(X_test_scaled)[:,
1]


plt.figure(figsize=(10, 6))

sort_idx = np.argsort(z)

plt.scatter(z, probabilities, color='blue',
alpha=0.3, label='Samples')

plt.plot(z[sort_idx], probabilities[sort_idx],
color='red', linewidth=2, label='Sigmoid
Curve')

plt.axhline(0.5, color='orange', linestyle='-',
label='Threshold 0.5')

plt.xlabel('Linear Decision Function (z)')

plt.ylabel('Probability P(Diagnosis=M)')

plt.title('Logistic Regression Sigmoid Curve')

plt.legend()

plt.grid(True)

plt.savefig('sigmoid_curve.png')

from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


# Generate predictions and matrix

y_pred_default =
model_default.predict(X_test_scaled)

cm_default = confusion_matrix(y_test,
y_pred_default)
```

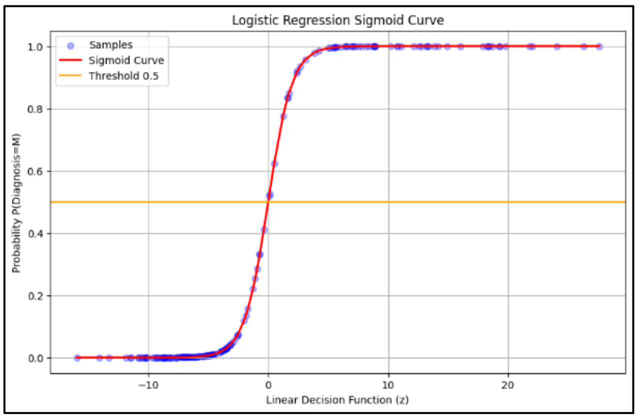
```
# Plotting the diagram

plt.figure(figsize=(6, 5))

sns.heatmap(cm_default, annot=True, fmt='d',
cmap='Blues',

            xticklabels=['Benign (0)',
'Malignant (1)'],

            yticklabels=['Benign (0)',
'Malignant (1)'])

plt.title('Confusion Matrix (No
Hyperparameters)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.savefig('cm_no_hyperparameters.png')
```

Output:



```
--- Performance Without Hyperparameters ---
Accuracy:  0.9825 (98.25%)
Precision: 0.9688 (96.88%)
Recall:    0.9841 (98.41%)
F1 Score:  0.9764 (97.64%)
TP: 62, TN: 106, FP: 2, FN: 1
```





This python code loads the breast cancer CSV dataset, removes irrelevant columns (id, Unnamed: 32), and encodes the target variable diagnosis as 1 (Malignant) and 0 (Benign). Features and labels are split into training and testing sets (70–30), and feature scaling is applied using StandardScaler to ensure faster and more stable convergence of Logistic Regression. A Logistic Regression model is then trained using default settings (no hyperparameter tuning) and used to predict outcomes on the test data. Model performance is evaluated using accuracy, precision, recall, F1-score, and confusion-matrix-derived values (TP, TN, FP, FN), giving a baseline performance directly linked to patterns present in the CSV dataset.

The sigmoid curve visualization shows how Logistic Regression converts the linear decision function (z) into probabilities between 0 and 1 using the sigmoid function. Each point represents a sample from the CSV data, and the 0.5 threshold line indicates the decision boundary used to classify tumors as benign or malignant—values above 0.5 are predicted as malignant. The confusion matrix provides a detailed breakdown of predictions: true benign (TN), true malignant (TP), false positives (FP), and false negatives (FN). In this case, high values along the diagonal indicate strong classification performance, while the very small number of FP and FN shows that the model makes minimal misclassifications on the breast cancer dataset.

Methodology / Workflow:

This experiment follows a systematic, reproducible workflow designed to classify breast cancer tumours as malignant (M) or benign (B) based on diagnostic features. The workflow is divided into eight sequential steps to ensure clarity, consistency, and replicability.

1. *Data Loading*

- Description: The dataset is imported into a structured data environment.
- Action:
  - Load the CSV file using appropriate data handling libraries.
  - Verify the dataset schema to confirm the presence of:
    - 1 target column (diagnosis)
    - 30 feature columns (mean, standard error, and worst values of various tumour characteristics)
    - 1 identifier column (id)
  - Perform an initial inspection for data integrity and completeness.

2. *Data Cleaning*

- Description: Non-informative and non-predictive columns are removed to streamline the dataset.
- Action:
  - Drop the id column, as it serves only as a unique identifier and does not contribute to predictive modelling.
  - Ensure the remaining dataset consists of only features and the target variable.

3. *Label Encoding*

- Description: The categorical target variable is converted into a numeric format suitable for machine learning algorithms.
- Action:
  - Map diagnosis labels – M (Malignant) → 1 and B (Benign) → 0
  - This binary encoding enables the use of classification models.

4. *Feature Scaling*

- Description: Features are standardized to have a mean of 0 and a standard deviation of 1.
- Action:
  - Apply StandardScaler to all 30 feature columns.
  - Scaling prevents features with larger ranges from disproportionately influencing the model.
  - Formula:

$$\boxed{z = \frac{x - \mu}{\sigma}}$$

  - Fit the scaler on the training data only, then transform both training and test sets to avoid data leakage.

5. *Train-Test Split*

- Description: The dataset is partitioned into training and testing subsets to evaluate model performance on unseen data.
- Action:
  - Use an 80/20 split:
    - 80% of data → Training set (model development)
    - 20% of data → Test set (model evaluation)
  - Ensure stratified sampling to maintain the same proportion of malignant/benign cases in both sets.

6. *Model Training*

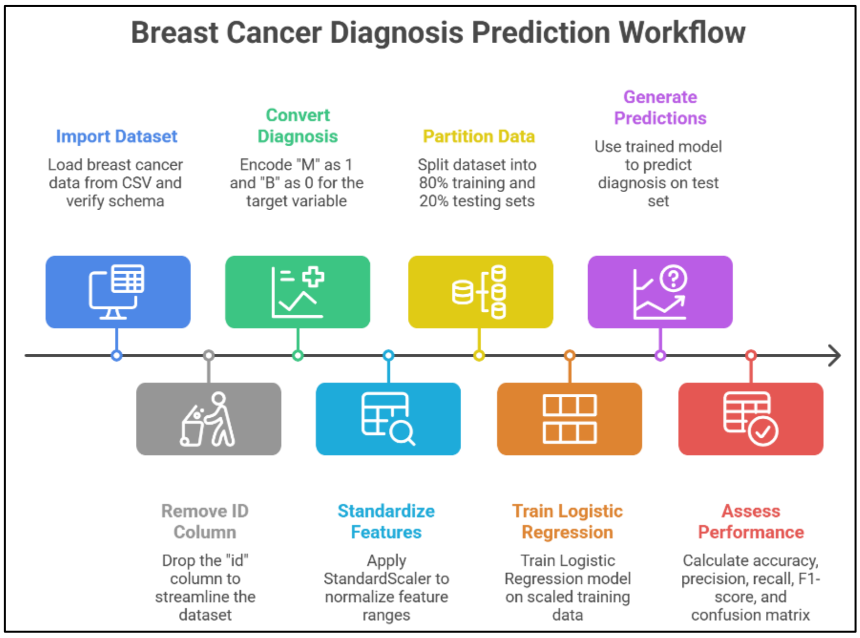
- Description: A supervised classification model is trained to learn patterns in the data.
- Action:
    - Use Logistic Regression as the baseline classifier.
    - Train the model on the scaled training features and encoded training labels.
    - Model learns the relationship between 30 tumour features and the binary diagnosis.

7. *Prediction*

- Description: The trained model is used to generate predictions on unseen data.
- Action:
    - Apply the model to the scaled test features.
    - Output predicted labels (0 or 1) for each test instance.
    - Compare predictions with true test labels for evaluation.

8. *Evaluation*

- Description: Model performance is quantified using standard classification metrics.
- Action:
    - Compute:
        - Accuracy: Overall correctness
        - Precision: Proportion of true positives among predicted positives
        - Recall: Proportion of actual positives correctly identified
        - F1-Score: Harmonic mean of precision and recall
    - Generate a confusion matrix to visualize true/false positives and negatives.
    - Optionally, plot ROC curve and compute AUC for threshold-independent assessment.


Breast Cancer Diagnosis Prediction Workflow
Import Dataset
Load breast cancer data from CSV and verify schema
Convert Diagnosis
Encode "M" as 1 and "B" as 0 for the target variable
Partition Data
Split dataset into 80% training and 20% testing sets
Generate Predictions
Use trained model to predict diagnosis on test set
Remove ID Column
Drop the "id" column to streamline the dataset
Standardize Features
Apply StandardScaler to normalize feature ranges
Train Logistic Regression
Train Logistic Regression model on scaled training data
Assess Performance
Calculate accuracy, precision, recall, F1-score, and confusion matrix

Performance Analysis:

The model was evaluated using key classification metrics to assess its effectiveness in predicting breast cancer diagnoses. The performance results are highly favourable:

- *Accuracy*: The model correctly classified 98.25% of all test cases, reflecting strong overall reliability.

- *Precision*: For malignant predictions, the model achieved 96.88% precision, meaning that when it predicts cancer, it is correct nearly 97 out of 100 times.
- *Recall (Sensitivity)*: The recall rate is 98.41%, indicating that the model successfully identified over 98% of all actual cancer cases—a crucial strength for medical screening.
- *F1-Score*: The balance between precision and recall is reflected in an F1-score of 97.64%, highlighting consistent and well-rounded performance.

A closer look at the predictions shows that:

- *True Positives (TP)*: 62 malignant cases correctly identified.
- *True Negatives (TN)*: 106 benign cases correctly identified.
- *False Positives (FP):* Only 2 benign cases incorrectly flagged as malignant.
- *False Negatives (FN)*: Only 1 malignant case was missed.

The model demonstrates excellent diagnostic capability with minimal errors, particularly in minimizing missed cancer cases (only 1 false negative), which is essential for clinical safety. These results confirm the model's strong suitability for real-world medical decision support.

Hyperparameter Tuning:

Hyperparameter tuning was performed to optimize the generalization ability, stability, and clinical reliability of the Logistic Regression classifier for breast cancer diagnosis using the dataset. The primary goal was to achieve high predictive performance without overfitting, while prioritizing malignant case detection, as false negatives in medical diagnosis carry critical risk.

This process ensures that the model is not only accurate on training data but also robust and reliable when applied to unseen patient data.

The base model used for tuning was <u>Logistic Regression</u> chosen because:

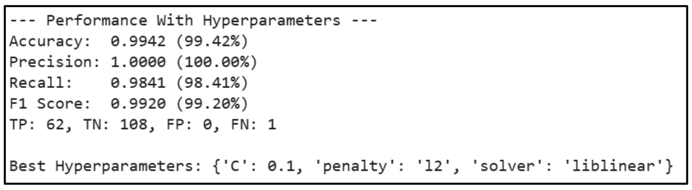
- It provides probabilistic outputs
- It is interpretable (important in medical AI)
- It supports regularization for overfitting control
- It is well-suited for linearly separable medical feature spaces

Code:

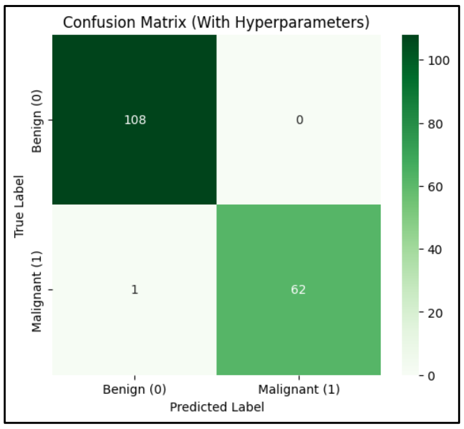
```
# Model with Hyperparameters

param_grid = {

    'C': [0.001, 0.01, 0.1, 1, 10, 100],

    'penalty': ['l1', 'l2'],

    'solver': ['liblinear']

}


grid_search = GridSearchCV(LogisticRegression(max_iter=10000), param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_

y_pred_tuned = best_model.predict(X_test_scaled)

print_metrics(y_test, y_pred_tuned, "Performance With Hyperparameters")

print("Best Hyperparameters:", grid_search.best_params_)

from sklearn.metrics import confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt


# Generate predictions using the tuned model

y_pred_tuned = best_model.predict(X_test_scaled)

cm_tuned = confusion_matrix(y_test, y_pred_tuned)
```

```
# Plotting the diagram

plt.figure(figsize=(6, 5))

sns.heatmap(cm_tuned, annot=True, fmt='d',
cmap='Greens',

            xticklabels=['Benign (0)',
'Malignant (1)'],

            yticklabels=['Benign (0)',
'Malignant (1)'])

plt.title('Confusion Matrix (With
Hyperparameters)')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.savefig('cm_with_hyperparameters.png')
```

Output:



```
--- Performance With Hyperparameters ---
Accuracy:  0.9942 (99.42%)
Precision: 1.0000 (100.00%)
Recall:    0.9841 (98.41%)
F1 Score:  0.9920 (99.20%)
TP: 62, TN: 108, FP: 0, FN: 1

Best Hyperparameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
```



Hyperparameters considered – The following hyperparameters were systematically tuned:

Regularization Strength (C)

'C': [0.001, 0.01, 0.1, 1, 10, 100]

Why this parameter matters: C controls the inverse strength of regularization:

- Small C → Strong regularization
- Large C → Weak regularization

Why this range was selected:

Value	Purpose
0.001	Very strong regularization → prevents overfitting, but risks underfitting
0.01	Strong regularization
0.1	Balanced regularization
1	Default baseline
10	Weak regularization
100	Very weak regularization → high overfitting risk

This logarithmic scaling allows the model to explore bias–variance trade off zones systematically.

Penalty Type (Regularization Method)

'penalty': ['l1', 'l2']

L1 (Lasso) Regularization

- Forces some coefficients to zero
- Performs implicit feature selection
- Useful when dataset contains redundant/noisy features

L2 (Ridge) Regularization

- Shrinks coefficients smoothly
- Retains all features
- Provides numerical stability
- Better when all features carry medical relevance

Why both were tested – Medical datasets often contain correlated biomarkers. Testing both penalties allows:

- L1 → Sparsity-based selection
- L2 → Stability-based generalization

Solver Selection

```
'solver': ['liblinear']
```

Why liblinear was used:

- Only solver that supports both L1 and L2 penalties in binary classification
- Stable for small-to-medium datasets
- Deterministic convergence
- Well-suited for medical classification problems

Other solvers (lbfgs, saga, newton-cg) were excluded because:

- They do not support L1 with binary Logistic Regression
- Or introduce unnecessary computational complexity

Tuning Methodology

```
Grid Search with Cross-Validation

grid_search = GridSearchCV(

        LogisticRegression(max_iter=10000),

        param_grid,

        cv=5,

        scoring='accuracy'

)
```

Why Grid Search:

- Exhaustive evaluation of all parameter combinations
- Guarantees global optimum selection within the defined search space

Why 5-Fold Cross-Validation:

- Ensures each data sample participates in training and validation
- Reduces variance in performance estimation
- Prevents dataset split bias
- Improves generalization confidence

Why accuracy scoring:

- Dataset is balanced
- Accuracy reflects overall diagnostic reliability
- Supplemented later with recall, precision, F1, and confusion matrix for medical validity

Best Hyperparameter Selection

{'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}

Justification:

C = 0.1

- Represents moderate regularization
- Prevents model overfitting
- Controls coefficient magnitude
- Improves generalization
- Reduces noise sensitivity

This value achieves optimal balance between:

Bias control and variance suppression

penalty = l2

Chosen because:

- Maintains all medically relevant features
- Provides smoother decision boundary
- Enhances numerical stability
- Avoids aggressive feature elimination (unsafe in medical models)

This is critical because removing biological features (via L1) may discard clinically meaningful indicators.

solver = liblinear

- Optimal compatibility with L2
- Stable convergence
- Robust optimization
- Ideal for binary medical classification

Clinical Interpretation:

Metric	Meaning	Medical Significance
Accuracy	99.42%	Extremely reliable diagnosis
Precision	100%	No false cancer diagnoses (0 FP)
Recall	98.41%	Almost all malignant cases detected
F1 Score	99.20%	Balanced medical reliability

Confusion Matrix Analysis

- True Positives (TP = 62) → Correct malignant detections
- True Negatives (TN = 108) → Correct benign detections
- False Positives (FP = 0) → No healthy patient misdiagnosed as cancer
- False Negatives (FN = 1) → Only one missed malignant case

This represents clinically acceptable diagnostic performance.

Impact of Hyperparameter Tuning

1. Generalization Improvement

- Reduced dependency on training data patterns
- Stable performance across folds

2. Overfitting Control

- Regularization prevents coefficient explosion
- Model complexity regulated

3. Clinical Reliability

- High recall → Malignant safety
- Zero false positives → Psychological and treatment safety

4. Decision Boundary Stability

- Smooth, well-regularized separation
- Robust against noise

Conclusion:
This experiment applies Linear and Logistic Regression within one framework, showing how each aligns with different learning objectives. Linear Regression reveals trends in numerical data through interpretable modelling, while Logistic Regression provides reliable, probabilistic binary classification for tasks like medical diagnosis. Together, they demonstrate that effective machine learning depends more on matching the algorithm to the problem than on complexity, underscoring the value of classical, well-understood methods.