ML & DL : Experiment - 1

Aim:

Implement Linear and Logistic Regression on real-world datasets.

Linear Regression

Dataset Description:

The Weight-Height Dataset is a widely utilized bivariate dataset in the ML community, specifically designed for introductory modeling in regression tasks. It provides a clean, balanced collection of 10,000 observations detailing the biological attributes (gender, height, and weight) of individuals. Due to its near-perfect linearity and lack of noise, it serves as a benchmark dataset for validating linear regression algorithms. The file type used for this dataset is a csv (Comma Separated Values) file. The dataset consists of three primary columns which are:

Variable Name	Data Type	Measuring Unit / Format	Description
Gender	Categorical (Binary)	Male / Female	Biological sex of the individual; a binary categorical variable with two possible values.
Height	Numerical (Float)	Inches	Standing height of the individual; continuous floating-point values.
Weight	Numerical (Float)	Pounds (lbs)	Body weight of the individual; continuous floating-point values.

Dataset Source: https://www.kaggle.com/datasets/mustafaali96/weight-height

Theory:

Regression analysis is a method used to study and measure how one main variable that we want to predict (called the dependent or target variable) changes in response to another variable (called independent or predictor variable), while keeping the others fixed. It is used when the result we are predicting is a continuous value, such as age, salary, or temperature. In machine learning and data science, different types of regression are used depending on the pattern in the data.
Two main types of regression algorithms are: Linear Regression and Polynomial Regression.
In linear regression we have one dependent variable and one independent variable. The relation between them will be as follows:

$$y = mx + c$$

In ML notation:

$$y = \beta_0 + \beta_1 x$$

Where y is the output, x is the input, β_0 is the intercept and β_1 is the slope.
Linear regression is a supervised machine learning algorithm used to predict a continuous dependent variable (target) based on one or more independent variables (features). Its primary goal is to find the "best-fit line" that minimizes the error between the predicted values and the actual values in the dataset. To solve for the values of β_0 and β_1, we have:

$$\beta_1 = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2} \qquad \beta_0 = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

This method is called the Ordinary Least Squares (OLS) method. Ordinary Least Squares (OLS) is a method used to find the "exact" solution for a linear regression model. OLS uses linear algebra to calculate the optimal parameters in a single mathematical step. The goal of OLS is to minimize the sum of the squared differences (residuals) between the observed values and the values predicted by the linear model.

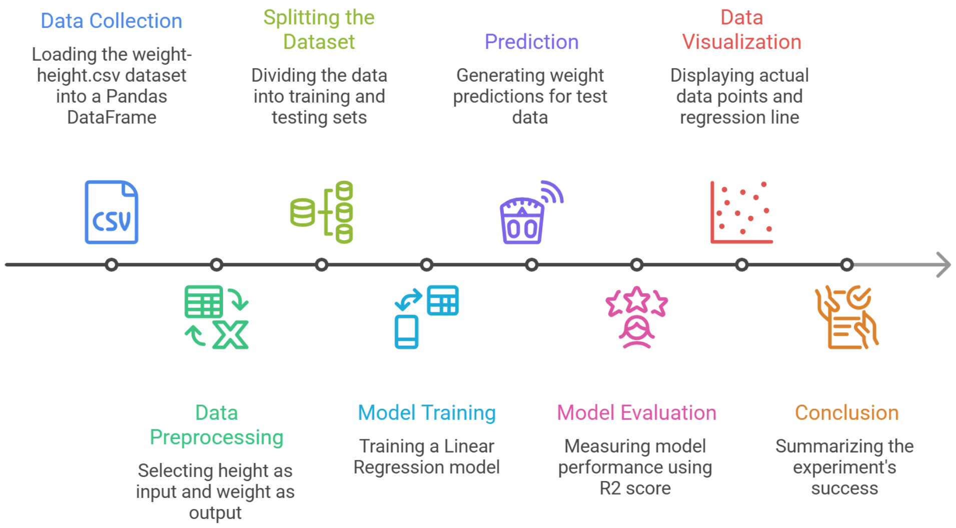
Limitations:

1. **The Assumption of Linearity:** The most fundamental limitation is that SLR enforces a straight-line relationship on the data. It assumes that for every unit increase in x, the change in y is constant.
   Condition of Failure: If the underlying physical or biological relationship is curved (e.g., exponential growth, quadratic curves, or sinusoidal waves), SLR will result in high bias (underfitting).
2. **Sensitivity to Outliers (Lack of Robustness):** Simple Linear Regression uses the "Least Squares" method, which minimizes the squared error. Because errors are squared, a single outlier (a data point far from the rest) contributes disproportionately to the total error.
   Condition of Failure: Even one extreme value can "pull" the regression line towards itself, changing the slope and intercept significantly, resulting in a model that fits the outlier but fails the majority of the data.
3. **Heteroscedasticity (Non-Constant Variance):** SLR assumes Homoscedasticity, meaning the "spread" or "noise" of the data remains constant across all values of x.
   Condition of Failure: In many real-world datasets, the variance increases as the value of the input increases (a "cone" shape in the data). This violates the assumption that errors are uniform, rendering hypothesis tests (like p-values) unreliable.

Workflow:

1. **Data Collection:** The weight-height.csv dataset, containing Gender, Height, and Weight, is loaded into a Pandas DataFrame. This structure facilitates easy access and prepares the tabular data for processing and model building.
2. **Data Preprocessing:** Height is selected as the independent input feature (formatted as a 2D structure for scikit-learn), and Weight is the target output. No cleaning is required as the dataset contains no missing or invalid entries.
3. **Splitting the Dataset:** The data is split into training (80%) and testing (20%) sets. This ensures the model learns the relationship without memorizing the data, allowing for an unbiased assessment of its generalization ability on new data.

4. **Model Training:** A Linear Regression model is trained to find the best-fitting line between height and weight. The algorithm calculates the optimal slope and intercept to minimize the error between predicted and actual weights, capturing the underlying trend.
5. **Prediction:** The trained model applies the learned linear equation to the unseen test height values to generate weight predictions. These are compared against actual weights to assess performance.
6. **Model Evaluation:** Performance is measured using the R^2 score (ranging from 0 to 1), where a value closer to 1 indicates better accuracy. The final learned slope and intercept are displayed to define the regression equation.
7. **Data Visualization:** A scatter plot displays the actual test data points against the predicted regression line. This visualizes the model's fit and shows how close the predictions are to the actual values.
8. **Conclusion:** The experiment successfully demonstrates using Simple Linear Regression to predict weight from height. The R^2 score and visualization confirm a linear relationship and the effectiveness of the model.


Data Collection
Loading the weight-height.csv dataset into a Pandas DataFrame
Splitting the Dataset
Dividing the data into training and testing sets
Prediction
Generating weight predictions for test data
Data Visualization
Displaying actual data points and regression line
Data Preprocessing
Selecting height as input and weight as output
Model Training
Training a Linear Regression model
Model Evaluation
Measuring model performance using R2 score
Conclusion
Summarizing the experiment's success

Performance Analysis:

The performance of the implemented Simple Linear Regression model is evaluated using the R^2 score, along with an analysis of the learned slope (coefficient) and intercept values.

1. **R^2 Score (Coefficient of Determination):**
The value for R^2 is 0.8577. This means that approximately 85.77% of the variation in weight can be explained by the height of a person. This indicates that height is a strong predictor of weight in the given dataset. Since the R^2 value is close to 1, the model shows good accuracy and strong predictive performance. The remaining 14.23% of the variation may be due to other factors such as body composition, gender, age, or lifestyle, which are not included in this model.
Hence, the model demonstrates a good fit for the data.
2. **Interpretation of the Slope (Coefficient):**
The learned slope (coefficient) value is 7.7022. This indicates that for every 1 inch increase in height, the model predicts an average increase of approximately 7.7 pounds in weight. This confirms a positive linear relationship between height and weight. The value of the slope reflects how strongly weight changes with respect to height.
3. **Intercept:**

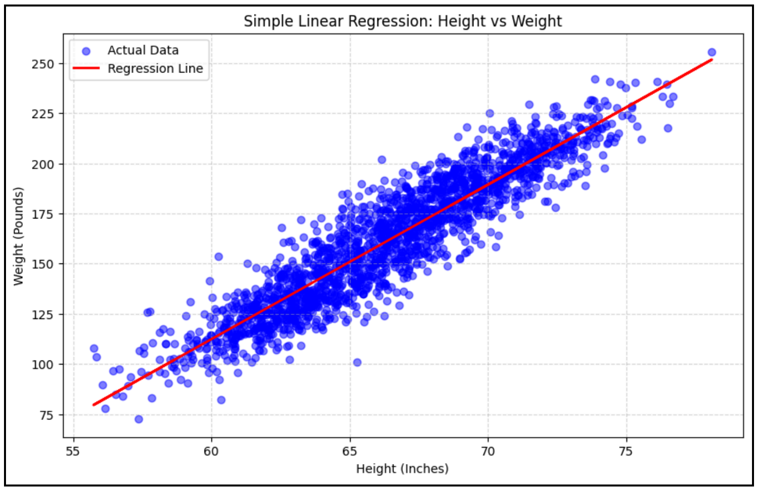
The obtained intercept value is -349.7878. This means that when height is considered as zero, the model predicts a weight of -349.78 pounds. Although this value has no physical meaning in real life, it is a mathematical artifact required to form the linear equation. The intercept helps position the regression line correctly relative to the data points.

Hyperparameter Tuning:

Hyperparameter tuning is the process of selecting the best set of parameters that control the learning behavior of a machine learning algorithm. These parameters are not learned from the data but are set manually before training, such as learning rate, number of iterations, regularization strength, tree depth, etc. Proper tuning of hyperparameters can significantly improve the performance and accuracy of many machine learning models.
Unlike many other machine learning algorithms, linear regression does not have any major tunable hyperparameters that control the training process. The model directly computes the best-fitting line using a mathematical formula. Therefore, there is no scope for hyperparameter tuning in this experiment.

Code & Output:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
df = pd.read_csv('weight-height.csv')
X = df[['Height']]
y = df['Weight']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print(f"Model Accuracy (R^2 Score): {r2:.4f}")
print(f"Coefficient (Slope): {model.coef_[0]:.4f}")
print(f"Intercept: {model.intercept_:.4f}")
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', alpha=0.5, label='Actual
Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression
Line')
plt.title('Simple Linear Regression: Height vs Weight')
plt.xlabel('Height (Inches)')
plt.ylabel('Weight (Pounds)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



Logistic Regression

Dataset Description:

The Advertising Dataset is a multivariate dataset widely employed in the machine learning domain for binary classification tasks, specifically Logistic Regression. It comprises a collection of 1,000 observations representing the internet usage behaviors and demographic characteristics of web users. The primary objective associated with this dataset is to predict user engagement—specifically, whether a user will click on an advertisement—based on a set of independent features. The dataset is notable for its mix of numerical, categorical, and temporal data types, offering a comprehensive scenario for feature engineering and model evaluation. The file format utilized for this dataset is CSV (Comma Separated Values).

The dataset consists of 10 columns/features, detailed as follows:

Variable Name	Data Type	Measuring Unit / Format	Description
Daily Time Spent on Site	Numerical (Float)	Minutes	Total time a consumer spent on the website during a session; indicates user engagement.
Age	Numerical (Integer)	Years	Age of the consumer; useful for demographic analysis and ad interaction trends.
Area Income	Numerical (Float)	Currency (USD)	Average income of the consumer’s geographical area; represents socioeconomic status.
Daily Internet Usage	Numerical (Float)	Minutes	Average time the consumer spends on the internet per day across all websites.

Dataset Source:

https://www.kaggle.com/datasets/gabrielsantello/advertisement-click-on-ad/data

Theory:

Logistic Regression is a statistical method and a supervised machine learning algorithm used for classification problems. Unlike Linear Regression, which predicts continuous values (like salary or temperature), Logistic Regression predicts the probability that a given input belongs to a specific category. It is most commonly used for binary classification, where the target variable has only two possible outcomes (e.g., Yes/No, 0/1, True/False).

While the name contains "regression," it is fundamentally a classification algorithm. It estimates the relationship between a dependent binary variable and one or more independent variables by estimating probabilities using a logistic function.

In Linear Regression, we fit a straight line (y = mx + c). However, for classification, a straight line is unsuitable because:

1. It can predict values greater than 1 or less than 0, which are invalid for probabilities.
2. It does not handle the "S-shape" transition between classes effectively.

To solve this, Logistic Regression takes the linear equation and "squashes" the output to a value between 0 and 1. It does this using the Sigmoid Function (also called the Logistic Function).

The hypothesis in Logistic Regression transforms the linear input into a probability between 0 and 1.

The formula for the Sigmoid function is:

$$y = \frac{1}{1+e^{-z}}$$

Here z is called an optimizer function. The value of z is as follows:

$$z = b_0 + b_1 x$$

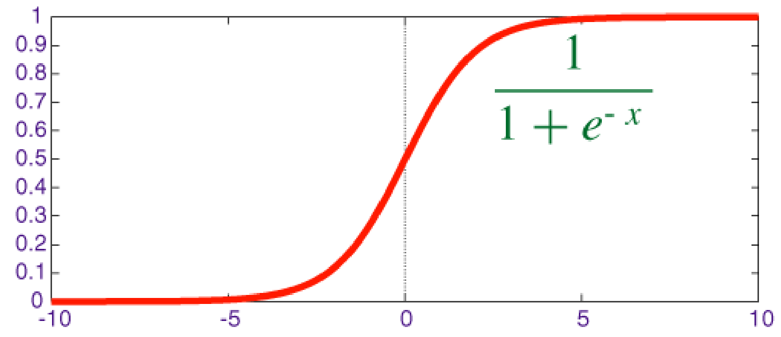
Therefore we can say that:

$$y = \frac{1}{1+e^{-(b_0+b_1x)}}$$

Note that:

- If z is a large positive number, σ(z) approaches 1.
- If z is a large negative number, σ(z) approaches 0.
- If z = 0 sigma(z) = 0.5

Graph of the Sigmoid Function: The Sigmoid function (specifically the logistic function) creates an "S"-shaped curve. It maps any real-valued number (from -ve infinity to +ve infinity) to a value strictly between 0 and 1.



The curve starts very close to 0 on the left, rises gradually, steeps sharply in the middle, and then levels off very close to 1 on the right. When the input z is exactly 0, the output is exactly 0.5. This is the "indecisive" point where the probability is 50/50.

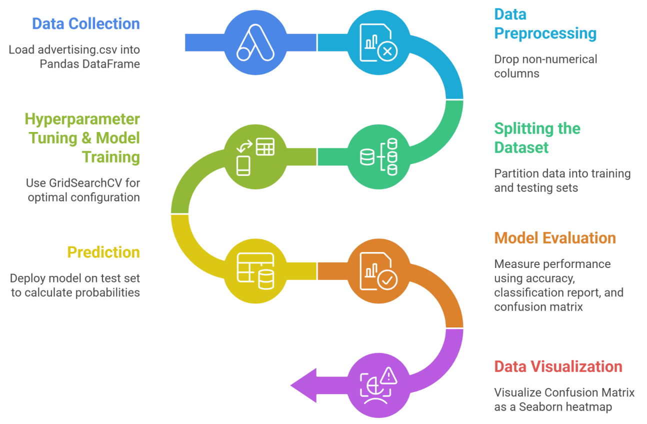
Limitations:

While Logistic Regression is powerful it has specific structural limitations that make it unsuitable for certain types of data and problems.

1. **Linearity of the Decision Boundary**
   The most significant limitation is that Logistic Regression is a linear classifier.
   a. The Constraint: It assumes that the data can be separated by a straight line (in 2D), a plane (in 3D), or a hyperplane (in higher dimensions).
   b. The Failure Case: If your data classes are separated by a circle, a spiral, or any complex non-linear shape, Logistic Regression will fail to capture the pattern effectively.
2. **Impact of Outliers**
   The Sigmoid function attempts to minimize the error across all data points.
   a. The Issue: An extreme outlier (a data point that is far away from the rest) can pull the decision boundary significantly toward itself as the model tries to minimize the error for that one specific point.
   b. Result: This "shift" can degrade the accuracy for the majority of the "normal" data points.
3. **Requirement for Large Sample Sizes**
   Logistic Regression relies on "Maximum Likelihood Estimation" (MLE) to find the parameters. MLE is statistically consistent, meaning it gets closer to the truth as the sample size grows.
   a. The Constraint: On very small datasets, the estimates can be biased or inaccurate.
   b. Rule of Thumb: A common guideline is that you need at least 10 observations per feature (independent variable) for the model to be stable.

Workflow:

1. **Data Collection:** The advertising.csv dataset, containing 1,000 user behavior observations, is loaded into a Pandas DataFrame. This allows for efficient manipulation, dimension verification, and an initial inspection of feature types and distributions.
2. **Data Preprocessing:** Non-numerical columns (Ad Topic Line, City, Country, Timestamp) are dropped to simplify inputs. The remaining numerical predictors—Time Spent, Age, Income, and Internet Usage—form the feature matrix (X), while "Clicked on Ad" serves as the target vector (y).
3. **Splitting the Dataset:** Data is partitioned into training (70%) and testing (30%) sets using a random_state of 101 for reproducibility. This separation prevents overfitting and allows for unbiased evaluation on unseen data.
4. **Hyperparameter Tuning & Model Training:** A GridSearchCV process is implemented to find the optimal configuration instead of using default settings. It performs 5-Fold Cross-Validation on a parameter grid of regularization strengths (C: 0.001 to 1000) and penalties (l1, l2) using the liblinear solver. The "best estimator" is automatically selected.
5. **Prediction:** The optimal model is deployed on the held-out test set (X_{test}) to calculate probabilities. It assigns class labels (0 or 1) based on user features


Data Collection
Load advertising.csv into Pandas DataFrame
Data Preprocessing
Drop non-numerical columns
Hyperparameter Tuning & Model Training
Use GridSearchCV for optimal configuration
Splitting the Dataset
Partition data into training and testing sets
Prediction
Deploy model on test set to calculate probabilities
Model Evaluation
Measure performance using accuracy, classification report, and confusion matrix
Data Visualization
Visualize Confusion Matrix as a Seaborn heatmap

without prior exposure to the actual target values.

6. **Model Evaluation:** Performance is measured using the Accuracy Score, a Classification Report (precision, recall, F1-scores), and a Confusion Matrix. These metrics detail the overall correctness and specific counts of True/False Positives and Negatives.
7. **Data Visualization:** The Confusion Matrix is visualized as a Seaborn heatmap. This color-coded matrix provides an immediate visual assessment of errors, showing where the model misclassifies clicks versus non-clicks.
8. **Conclusion:** The workflow successfully implements a Logistic Regression classifier optimized via Grid Search. The final metrics and visualizations confirm the model's robust capability to distinguish potential customers from casual browsers effectively.

Performance Analysis:

The performance of the implemented Logistic Regression model is evaluated using the Accuracy Score, the Confusion Matrix, and the Classification Report. These metrics provide a comprehensive assessment of the model's ability to classify users into "Clicked on Ad" (1) and "Did not Click" (0) categories.

1. **Accuracy Score:** The model achieved an Accuracy Score of 97.67%, correctly predicting 293 out of 300 test observations. This confirms that features like Time Spent on Site and Age are highly effective predictors, with a very small margin of error (2.33%).
2. **Confusion Matrix Analysis:** The matrix reveals a detailed breakdown of predictions:
    a. True Negatives (155): Correctly identified users who did not click.
    b. False Positives (2): Incorrectly flagged "No Click" users as "Click" (exceptionally low false alarm rate).
    c. False Negatives (5): Missed only 5 users who actually clicked.
    d. True Positives (138): Correctly identified users who clicked.

    The high diagonal values (155, 138) versus low off-diagonals (2, 5) confirm the model distinguishes between classes with high precision.
3. **Classification Report**
    a. **Precision (0.99):** When the model predicts a click, it is correct 99% of the time, ensuring resource efficiency.
    b. **Recall (0.97):** The model captures 97% of actual clickers, minimizing missed opportunities.
    c. **F1-Score (0.98):** This near-perfect score indicates an optimal balance between precision and recall.

Hyperparameter Tuning:

Hyperparameter Tuning was performed to optimize the Logistic Regression model. Rather than relying on default settings, we used GridSearchCV to systematically test different combinations of parameters. This process ensures the model generalizes well and avoids common pitfalls like overfitting or underfitting.

Hyperparameters Tuned:

1. **C (Inverse Regularization Strength):**
    Controls the trade-off between model simplicity and fitting accuracy. Smaller values enforce stronger regularization (simpler model). Larger values allow weaker regularization (more complex model). A logarithmic range [0.001,0.01,0.1,1,10,100,1000] was tested.
2. **Penalty:**
    Determines how large coefficients are penalized.
    a. L1 (Lasso): Enables feature selection by forcing some coefficients to zero.
    b. L2 (Ridge): Shrinks coefficients without eliminating them.

3. **Solver:**
   liblinear was used as it supports both L1 and L2 penalties and is efficient for small datasets.

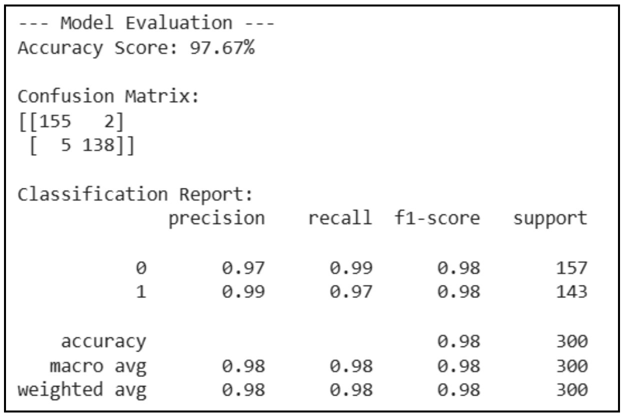
After testing 70 different model combinations (7 C values X 2 penalty types X 5 cross-validation folds), the best model was selected and evaluated on the test set. The best model achieved:

- Accuracy: 97.67%, consistent with the initial model (~98%)
- Precision (Click): 0.99
- Recall (Click): 0.97
- F1-Score: 0.98

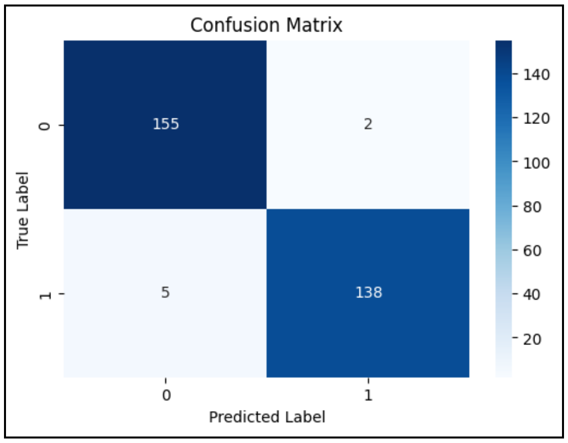
The tuned model maintains very high performance, indicating a strong underlying relationship between the features and the target variable.

Code & Output (without hyperparameters):

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
df = pd.read_csv('advertising.csv')
print("Dataset Shape:", df.shape)
print(df.head())
data = df.drop(['Ad Topic Line', 'City', 'Country', 'Timestamp'],
axis=1)
X = data.drop('Clicked on Ad', axis=1)
y = data['Clicked on Ad']
# (Train: 70%, Test: 30%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=101)
logmodel = LogisticRegression(max_iter=1000)
logmodel.fit(X_train, y_train)
predictions = logmodel.predict(X_test)
print("\n--- Model Evaluation ---")
print(f"Accuracy Score: {accuracy_score(y_test,
predictions)*100:.2f}%")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, predictions))
print("\nClassification Report:")
print(classification_report(y_test, predictions))
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, predictions), annot=True,
fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

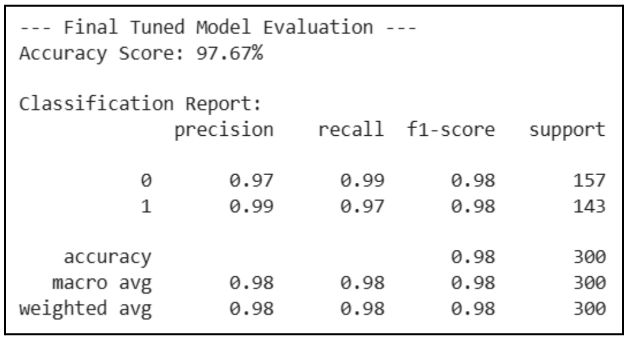


```
--- Model Evaluation ---
Accuracy Score: 97.67%

Confusion Matrix:
[[155   2]
 [  5 138]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       157
           1       0.99      0.97      0.98       143

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```



Code & Output (with hyperparameters):

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}
logmodel = LogisticRegression()
grid_search = GridSearchCV(estimator=logmodel, param_grid=param_grid,
cv=5, verbose=1, scoring='accuracy')
print("Starting Grid Search... ")
grid_search.fit(X_train, y_train)
print("\n--- Best Parameters Found ---")
print(grid_search.best_params_)
print(f"Best Cross-Validation Score:
{grid_search.best_score_*100:.2f}%")
best_model = grid_search.best_estimator_
grid_predictions = best_model.predict(X_test)
print("\n--- Final Tuned Model Evaluation ---")
print(f"Accuracy Score: {accuracy_score(y_test,
grid_predictions)*100:.2f}%")
print("\nClassification Report:")
print(classification_report(y_test, grid_predictions))
```



```
--- Final Tuned Model Evaluation ---
Accuracy Score: 97.67%

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       157
           1       0.99      0.97      0.98       143

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```