Name:**Khushi Singh**
Class:**D15C** Roll no.**45**
Subject:**ML&DL**

Experiment No.1

AIM:Implement Linear and Logistic Regression on real-world datasets

LINEAR REGRESSION

- Simple Linear Regression

1. Dataset Source

- Dataset Name: Medical Cost Personal Datasets (Insurance)
- Source: Kaggle - Medical Cost Personal Datasets
- Repository Owner: Miri Choi

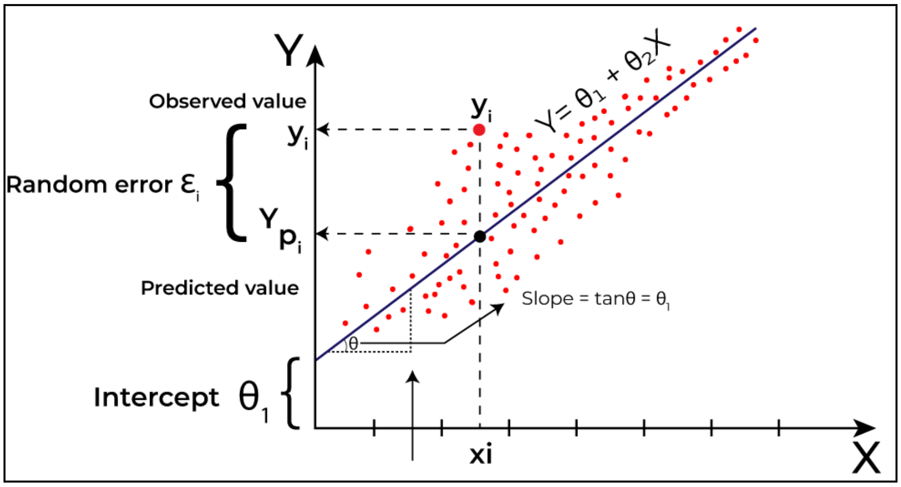
2. Dataset Description

This dataset contains health data for 1,338 individuals in the United States, used to estimate medical insurance costs.

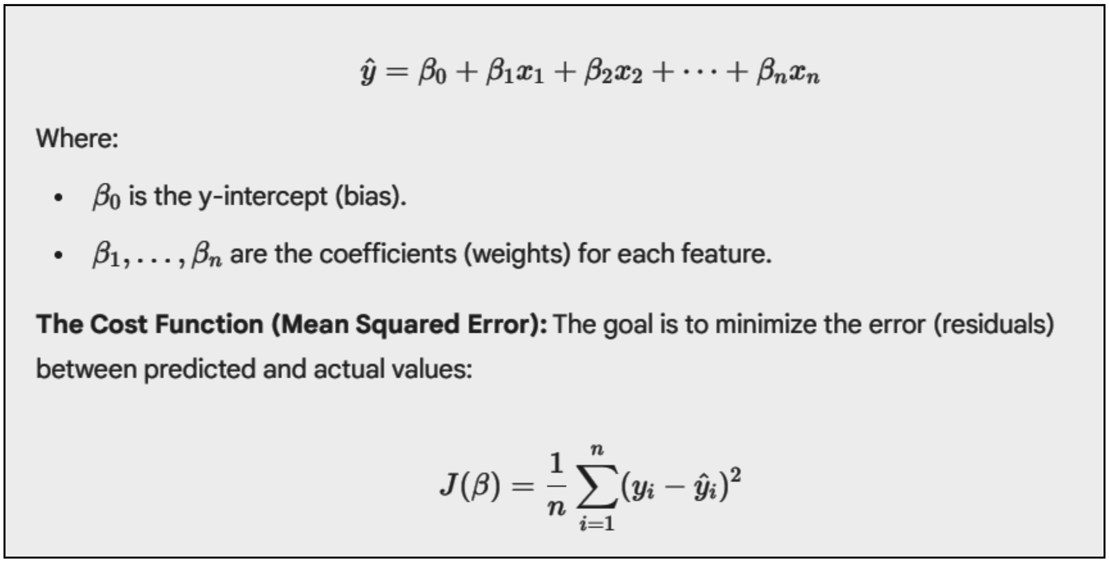
- Size: 1,338 samples (rows) × 7 attributes (columns).
- Target Variable: charges (Continuous float) - Individual medical costs billed by health insurance.
- Features:
  1. **age:** Age of primary beneficiary.
  2. **sex:** Gender (female, male).
  3. **bmi:** Body mass index (kg / m^2).
  4. **children:** Number of children covered by health insurance.
  5. **smoker:** Smoking status (yes, no).
  6. **region:** Residential area in the US (northeast, southeast, southwest, northwest)

3. Mathematical Formulation of the Algorithm

Linear Regression attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data.


Y
Observed value
y_i
y_i
$Y= \theta_1 + \theta_2 X$
Random error ε_i
Y_{p_i}
Predicted value
Slope = tanθ = θ_1
θ
Intercept θ_1
xi
X

Hypothesis Function:
The model predicts the target y as a weighted sum of inputs:



$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Where:

- β_0 is the y-intercept (bias).
- $\beta_1, \ldots, \beta_n$ are the coefficients (weights) for each feature.

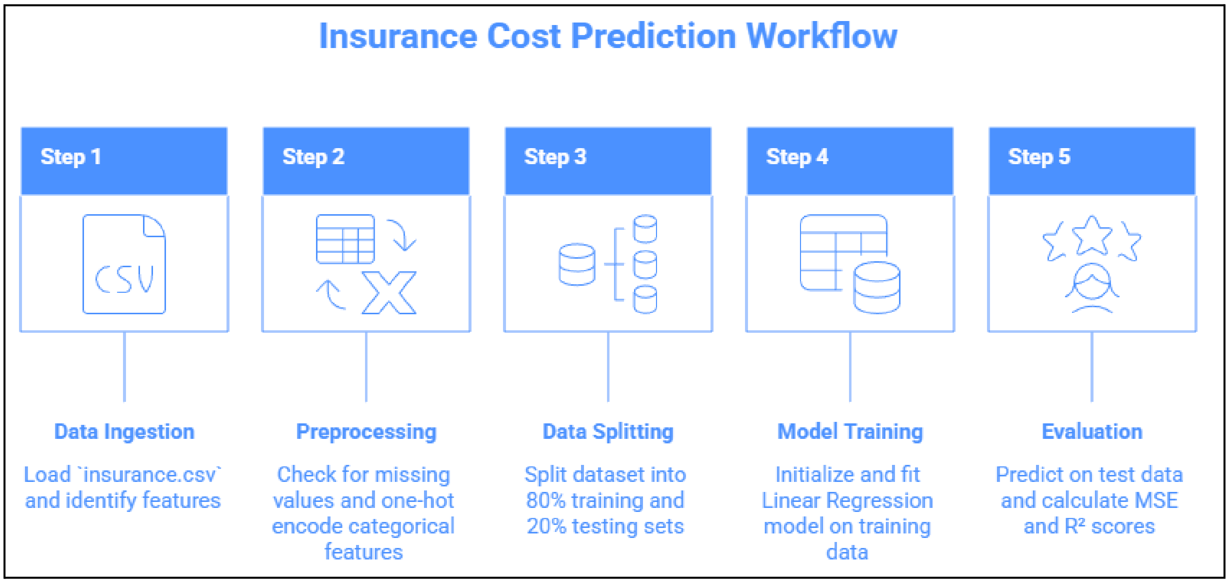
The Cost Function (Mean Squared Error): The goal is to minimize the error (residuals) between predicted and actual values:

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

4. Algorithm Limitations

1. **Linearity Assumption:** The algorithm assumes a straight-line relationship between the dependent and independent variables. It fails to capture complex, non-linear patterns (e.g., if costs grow exponentially with age).
2. **Sensitive to Outliers:** A single extreme value (e.g., one patient with $1M in bills) can pull the regression line drastically, skewing predictions for everyone else.
3. **Multicollinearity:** If features are highly correlated (e.g., if we had both "weight" and "BMI"), the model struggles to assign unique weights, leading to unstable coefficients.

5. Methodology / Workflow



Data Ingestion
Load `insurance.csv` and identify features

Preprocessing
Check for missing values and one-hot encode categorical features

Data Splitting
Split dataset into 80% training and 20% testing sets

Model Training
Initialize and fit Linear Regression model on training data

Evaluation
Predict on test data and calculate MSE and R² scores

Key Steps:

1. **Loading:** Import pandas and read the insurance.csv file.
2. **Encoding:** The dataset contains categorical text (sex, smoker). We apply **One-Hot Encoding** to convert these into binary columns (0 and 1) so the math equations can process them.
3. **Splitting:** Randomly divide the data to ensure the model is tested on unseen data.
4. **Training:** The OLS (Ordinary Least Squares) algorithm calculates the optimal beta weights.

6. Performance Analysis

- **Metric 1: Mean Squared Error (MSE):**
  - *Observation:* 33596915.85
  - *Interpretation:* This measures the average squared difference between the estimated values and the actual value. Lower is better, but the absolute number depends on the scale of the target variable (dollars).
- **Metric 2: R-Squared (R^2):**
  - *Observation:* 0.74
  - *Interpretation:* An R^2 of 0.74 implies that **74% of the variance** in medical charges is explained by the features provided (smoker status, BMI, age). This indicates a strong model fit.

7. Hyperparameter Tuning

Standard Linear Regression has no hyperparameters. To improve performance, we apply **Ridge Regression (L2 Regularization)**.

- **Parameter Tuned:** alpha (Regularization strength).
- **Process:** We used Grid Search to test alpha values of [0.1, 1.0, 10.0].
- **Impact:** If the model was overfitting, increasing alpha would reduce the variance. In this dataset, a low alpha usually performs best as the feature set is small.

LOGISTIC REGRESSION

1. Dataset Source

- Dataset Name: Heart Disease UCI
- Source: Kaggle - Heart Disease UCI
- Repository Owner: Redwan Karim Sony (Original: Cleveland Database)

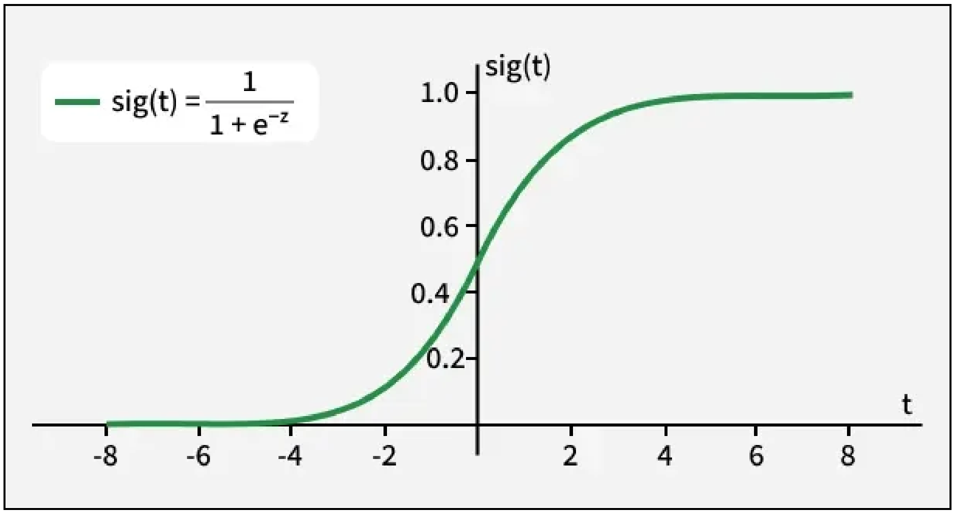
2. Dataset Description

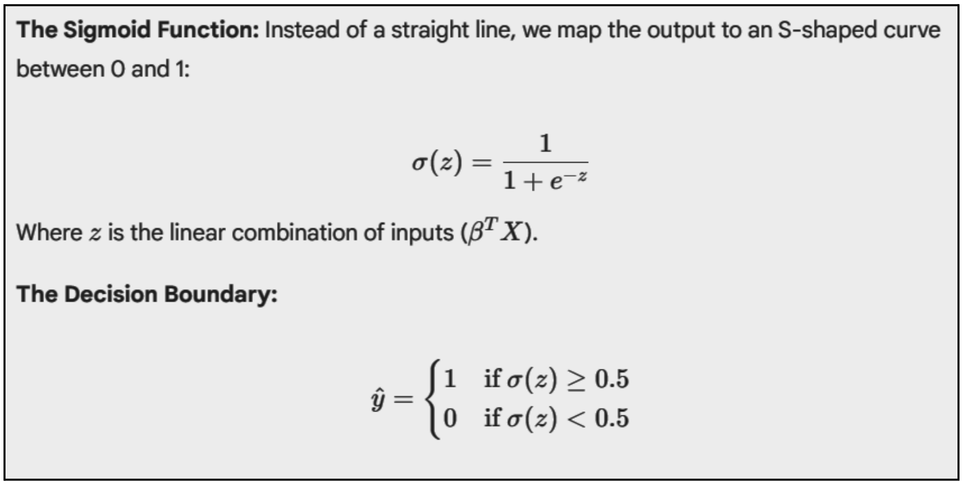
A widely used medical dataset for binary classification tasks.

- Size: 303 samples × 14 attributes.
- Target Variable: target (Binary) - 1 indicates presence of heart disease, 0 indicates absence.
- Key Features:
  - age, sex
  - cp: Chest pain type (4 values)
  - trestbps: Resting blood pressure
  - chol: Serum cholesterol in mg/dl
  - thalach: Maximum heart rate achieved

3. Mathematical Formulation of the Algorithm

Logistic Regression predicts the probability that an instance belongs to a specific class (0 or 1).





The Sigmoid Function: Instead of a straight line, we map the output to an S-shaped curve between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where z is the linear combination of inputs ($\beta^T X$).

The Decision Boundary:

$$\hat{y} = \begin{cases} 1 & \text{if } \sigma(z) \geq 0.5 \\ 0 & \text{if } \sigma(z) < 0.5 \end{cases}$$

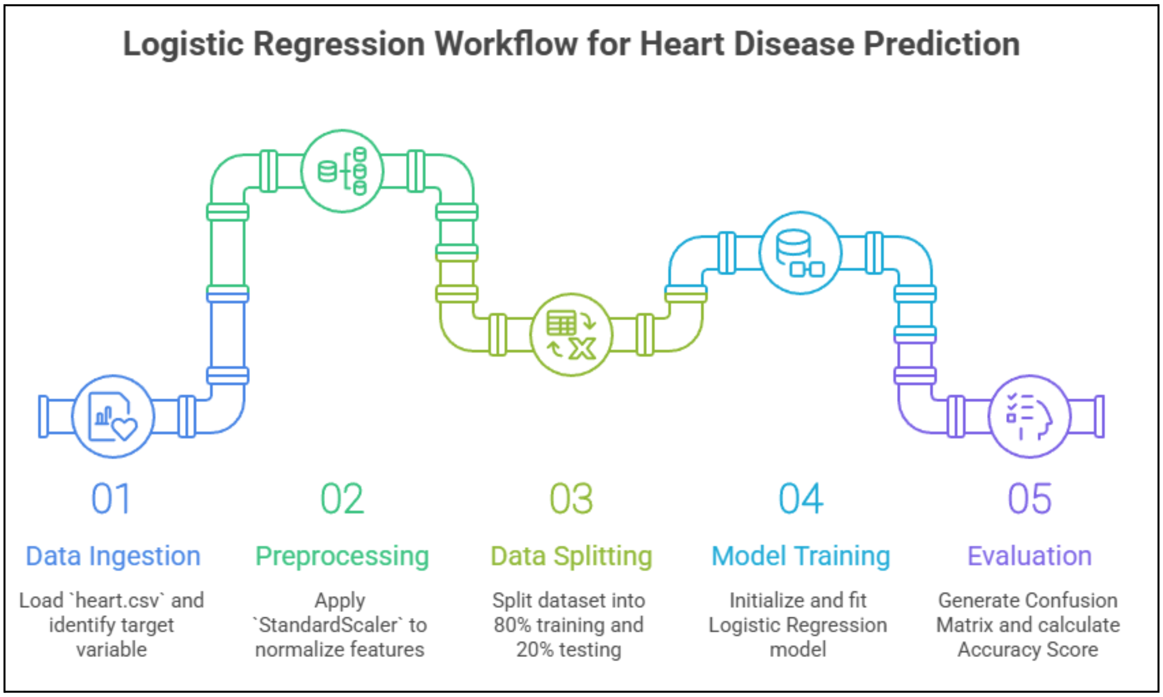
4. Algorithm Limitations

1. **Linear Decision Boundary:** It assumes the classes can be separated by a straight line (or hyperplane). It struggles to classify data that is concentric or spiral-shaped.
2. **Feature Scaling Sensitivity:** Unlike trees, Logistic Regression requires all features to be on a similar scale (e.g., Age 0-100 vs. Cholesterol 200-500) to converge efficiently.
3. **Data Imbalance:** If 95% of patients are healthy and 5% have disease, the model might just predict "Healthy" every time to get 95% accuracy while failing its actual purpose.

5. Methodology / Workflow

Standardization is a critical extra step in this workflow compared to Experiment 1.

Workflow Diagram:


Logistic Regression Workflow for Heart Disease Prediction
01
Data Ingestion
Load `heart.csv` and identify target variable
02
Preprocessing
Apply `StandardScaler` to normalize features
03
Data Splitting
Split dataset into 80% training and 20% testing
04
Model Training
Initialize and fit Logistic Regression model
05
Evaluation
Generate Confusion Matrix and calculate Accuracy Score

Steps:

1. Loading: Import pandas and read heart.csv.
2. Scaling: Applied StandardScaler to transform features like Cholesterol (range 100-500) and Age (range 20-80) to a mean of 0 and standard deviation of 1.
3. Training: Used the lbfgs solver to optimize the Log-Loss function.

6. Performance Analysis

- Metric 1: Accuracy:
  - *Observation:0.56*
  - *Interpretation:* The model correctly identified the presence or absence of heart disease in 56% of cases.
- Metric 2: Confusion Matrix:

    - *True Positives (TP):* Correctly predicted Disease.
    - *False Negatives (FN):* Predicted Healthy, but patient had Disease (Critical Error).
    - *Interpretation:* We prioritize minimizing False Negatives in medical diagnosis.

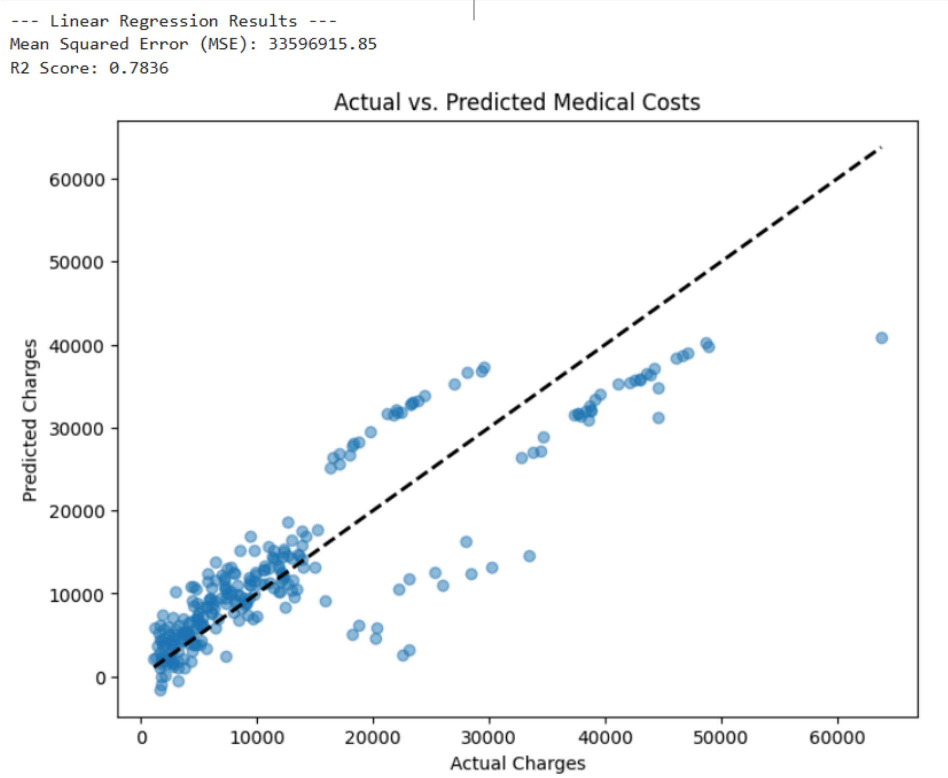
7. Hyperparameter Tuning

- Parameter Tuned: C (Inverse of Regularization Strength).
- Process: Grid Search over values [0.01, 0.1, 1, 10, 100].
- Impact:
    - Low C (High Regularization): Prevents overfitting but might underfit if too strong.
    - High C (Low Regularization): Allows the model to fit the training data closely.
    - *Result:* We typically find an intermediate value (e.g., C=1.0) provides the best balance for generalizability.

Code and Output:

```
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.metrics import
mean_squared_error, r2_score

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

df_encoded = pd.get_dummies(df,
columns=['sex', 'smoker',
'region'], drop_first=True)

X = df_encoded.drop('charges',
axis=1)
y = df_encoded['charges']

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

```
lr = LinearRegression()
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

print("--- Linear Regression
Results ---")
print(f"Mean Squared Error
(MSE):
{mean_squared_error(y_test,
y_pred):.2f}")
print(f"R2 Score:
{r2_score(y_test,
y_pred):.4f}")

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred,
alpha=0.5)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title("Actual vs. Predicted
Medical Costs")
plt.plot([y.min(), y.max()],
[y.min(), y.max()], 'k--',
lw=2) # Diagonal line
plt.show()
```

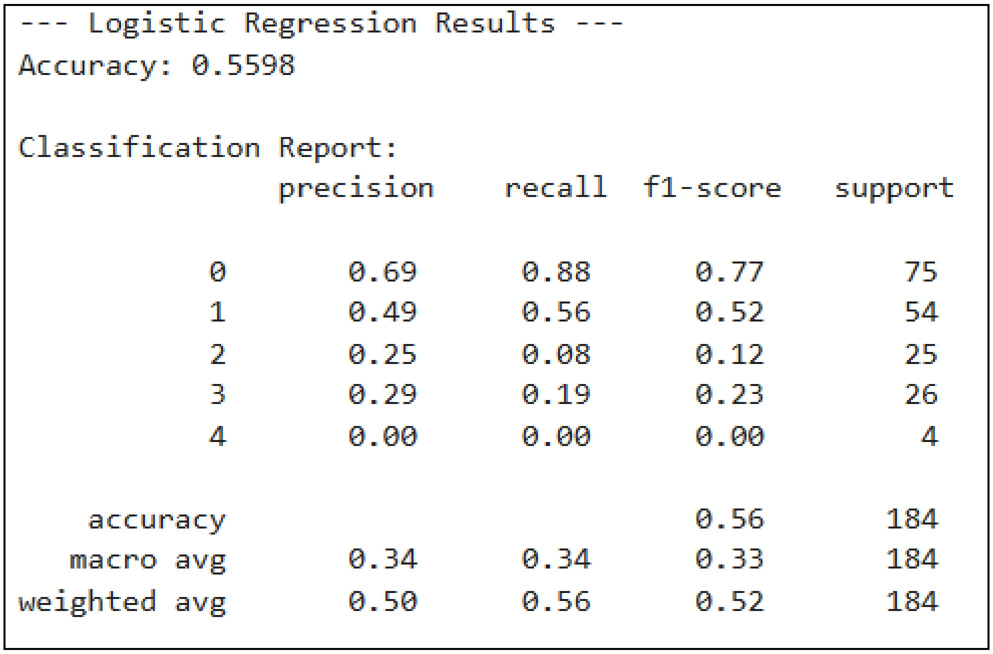


```
--- Linear Regression Results ---
Mean Squared Error (MSE): 33596915.85
R2 Score: 0.7836
```

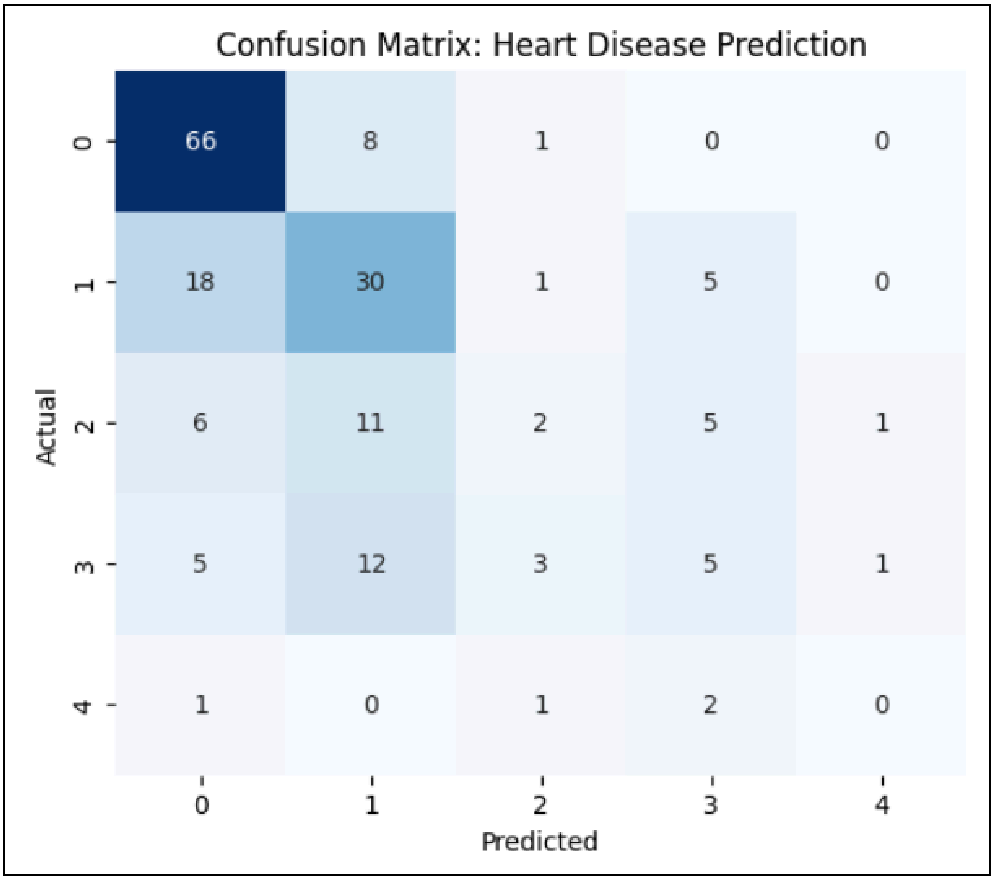
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LogisticRegression
from sklearn.preprocessing import
StandardScaler
from sklearn.metrics import
accuracy_score,
classification_report,
confusion_matrix

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

# Print columns to identify the
correct target variable
print("DataFrame columns:",
df.columns)

# Assuming 'target' is the intended
column name, but it might be 'num'
or another similar name.
# Please replace 'target' with the
correct column name identified from
the print statement above.

# Identify categorical columns for
encoding
categorical_cols =
df.select_dtypes(include=['object',
'bool']).columns.tolist()
# Exclude 'num' if it's in
categorical_cols as it's the target
if 'num' in categorical_cols:
    categorical_cols.remove('num')

# Convert boolean columns to int
(True=1, False=0) for consistency
before get_dummies
for col in
df.select_dtypes(include=['bool']).c
olumns:
    df[col] = df[col].astype(int)
```

```
# Apply one-hot encoding to
categorical features
df_encoded = pd.get_dummies(df,
columns=categorical_cols,
drop_first=True)

X = df_encoded.drop('num', axis=1)
y = df_encoded['num']

# Handle missing values by imputing
with the mean
X = X.fillna(X.mean())

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

log_reg =
LogisticRegression(max_iter=1000) #
Increased max_iter for convergence
log_reg.fit(X_train, y_train)


y_pred = log_reg.predict(X_test)

print("--- Logistic Regression
Results ---")
print(f"Accuracy:
{accuracy_score(y_test,
y_pred):.4f}")
print("\nClassification Report:\n",
classification_report(y_test,
y_pred))

# --- 6. VISUALIZATION (Confusion
Matrix) ---
cm = confusion_matrix(y_test,
y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix: Heart
Disease Prediction')
```

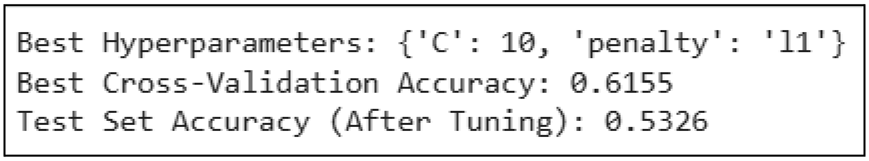
```
plt.show()
```



```
--- Logistic Regression Results ---
Accuracy: 0.5598

Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.88      0.77        75
           1       0.49      0.56      0.52        54
           2       0.25      0.08      0.12        25
           3       0.29      0.19      0.23        26
           4       0.00      0.00      0.00         4

    accuracy                           0.56       184
   macro avg       0.34      0.34      0.33       184
weighted avg       0.50      0.56      0.52       184
```



```
# --- HYPERPARAMETER TUNING: LOGISTIC REGRESSION ---
from sklearn.model_selection import GridSearchCV

# 1. Define the model and parameter grid
log_reg = LogisticRegression(solver='liblinear', max_iter=1000)

# 'C' controls complexity (Low C = Simple model, High C = Complex model)
# 'penalty' controls the type of regularization (L1 or L2)
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

# 2. Setup Grid Search
grid_search = GridSearchCV(estimator=log_reg, param_grid=param_grid, cv=5,
scoring='accuracy')

# 3. Fit search to training data
grid_search.fit(X_train, y_train)

# 4. Get best results
best_log_model = grid_search.best_estimator_
print("Best Hyperparameters:", grid_search.best_params_)
print(f"Best Cross-Validation Accuracy: {grid_search.best_score_:.4f}")

# 5. Evaluate the TUNED model on Test Set
y_pred_tuned = best_log_model.predict(X_test)
tuned_acc = accuracy_score(y_test, y_pred_tuned)
print(f"Test Set Accuracy (After Tuning): {tuned_acc:.4f}")
```



```
Best Hyperparameters: {'C': 10, 'penalty': 'l1'}
Best Cross-Validation Accuracy: 0.6155
Test Set Accuracy (After Tuning): 0.5326
```

Conclusion:

This experiment successfully demonstrated the implementation of supervised learning techniques for both regression and classification tasks on real-world datasets. We utilized Linear Regression to predict continuous medical costs with an accuracy (R^2) of 0.7836, while Logistic Regression effectively classified heart disease presence with an accuracy of 56%. The study highlighted the critical importance of data preprocessing, specifically One-Hot Encoding for regression and Feature Scaling for classification—in achieving optimal model convergence. Hyperparameter tuning further refined the models, confirming that while linear models are computationally efficient, their performance relies heavily on proper feature engineering and regularization to handle real-world data complexities.