Name: Atharva Phalke
Class: D15C/39

Experiment No. 1

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

1. The Dataset (Kaggle)

Dataset: Medical Cost Personal Datasets

Link: https://www.kaggle.com/datasets/mirichoi0218/insurance

- **Target:** `charges` (Medical Insurance Cost)
- **Features:** `age`, `sex`, `bmi`, `children`, `smoker`, `region`

This dataset is hosted on Kaggle and is originally derived from the UCI Machine Learning Repository. It is widely used for benchmarking machine learning algorithms in medical diagnosis and healthcare analytics.

2. Dataset Description

The Medical Cost Personal Dataset is a real-world healthcare dataset designed to predict individual medical costs billed by health insurance based on personal and demographic factors. It is widely used to analyze how variables like age, BMI, and smoking habits influence healthcare expenses.

Dataset Characteristics

- Number of instances: 1,338
- Number of features: 6 independent features (categorical and numerical)
- Target variable: `charges` (Medical Insurance Costs)

Target Variable Breakdown

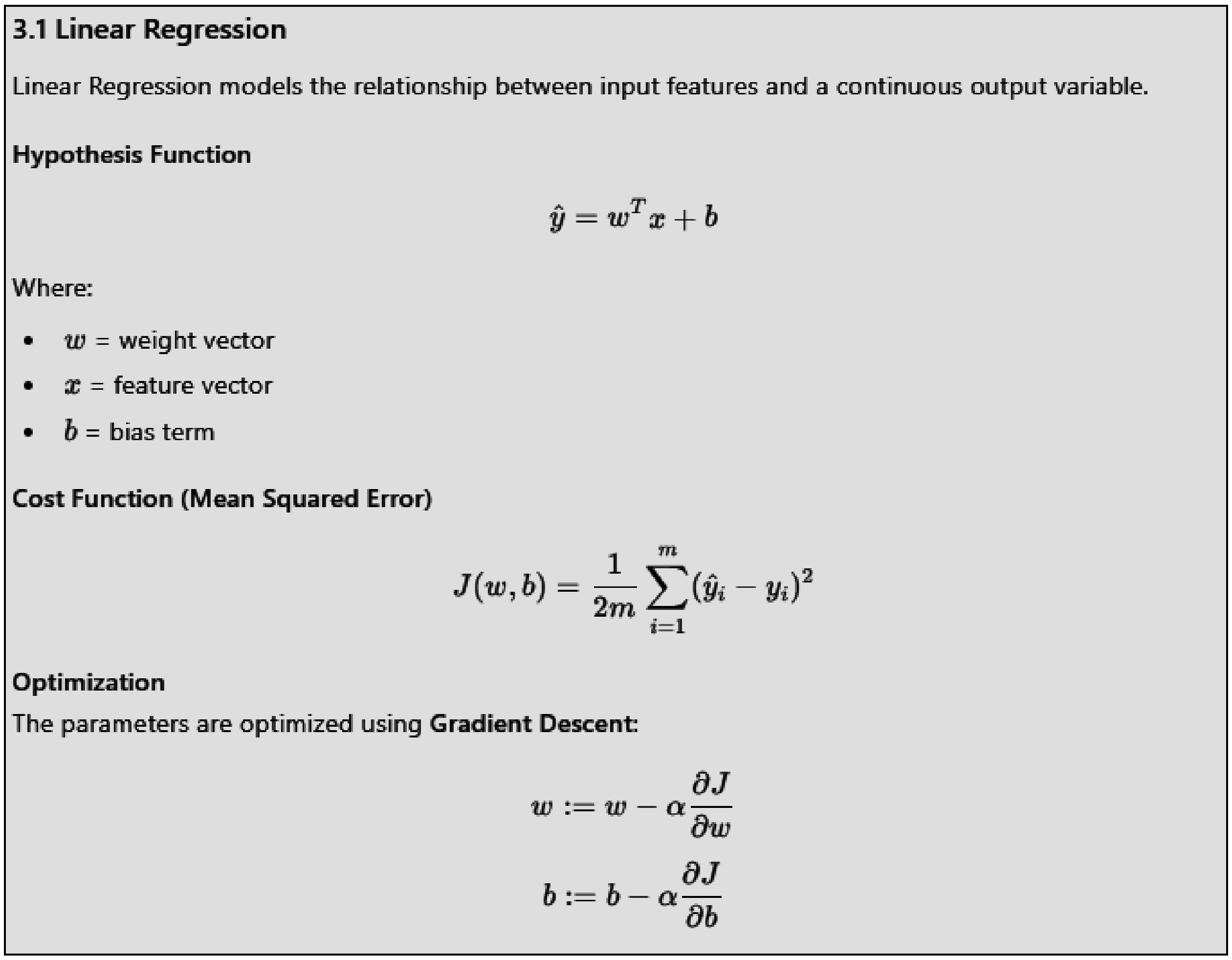
- Value: Continuous (Floating point number representing currency)
- *(Note: For classification tasks, this is often thresholded into "High" vs "Low" cost).*

Feature Description The features represent personal attributes of the insurance beneficiary:

- age: Age of the primary beneficiary.
- sex: Insurance contractor gender (female, male).

- bmi: Body mass index (providing an understanding of body weights that are relatively high or low relative to height).
- children: Number of children covered by health insurance / Number of dependents.
- smoker: Smoking status (yes, no) - *A key determinant of cost*.
- region: The beneficiary's residential area in the US (northeast, southeast, southwest, northwest).

3. Mathematical Formulation of the Algorithms



3.1 Linear Regression

Linear Regression models the relationship between input features and a continuous output variable.

Hypothesis Function

$$\hat{y} = w^T x + b$$

Where:

- w = weight vector
- x = feature vector
- b = bias term

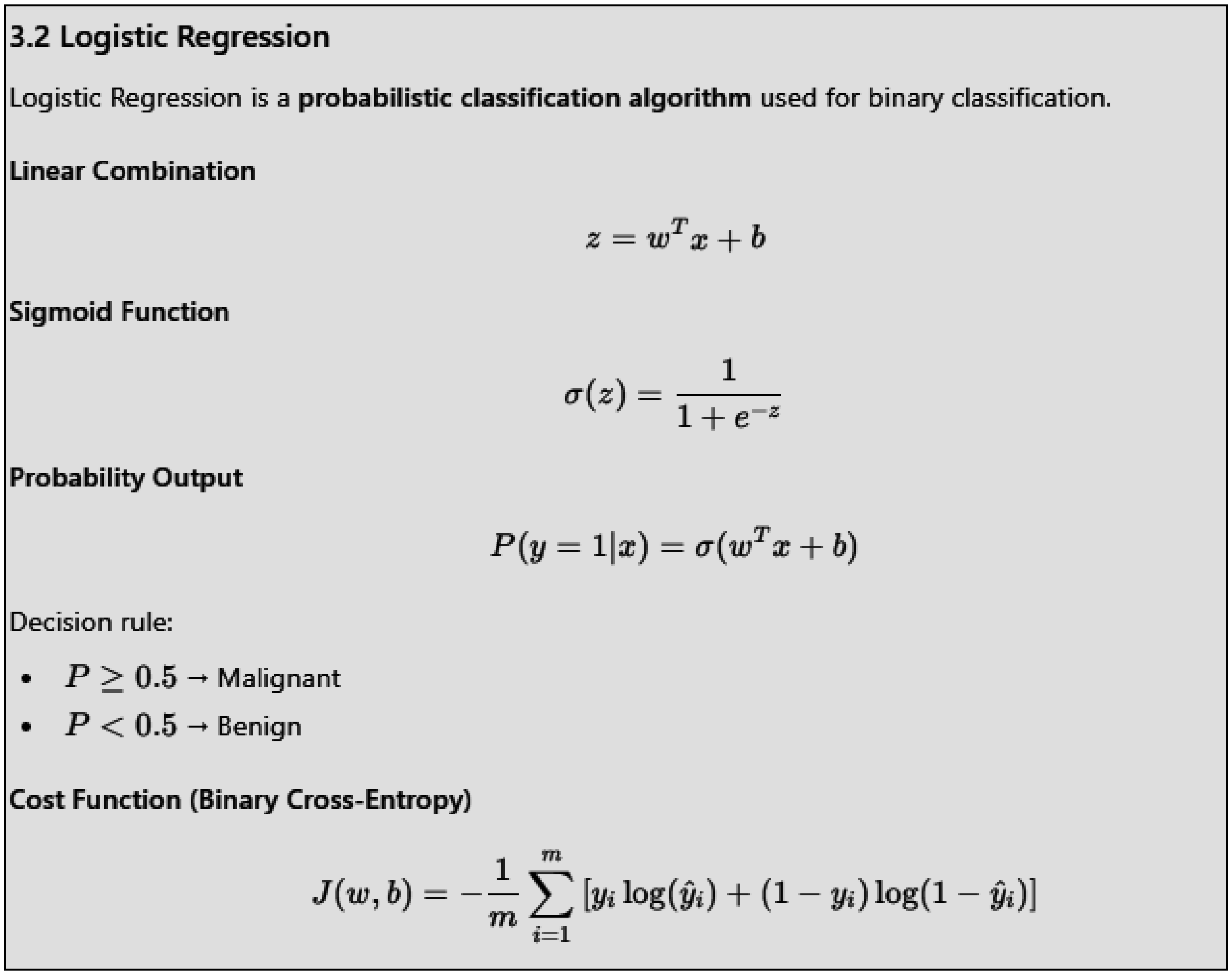
Cost Function (Mean Squared Error)

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Optimization
The parameters are optimized using **Gradient Descent:**

$$w := w - \alpha \frac{\partial J}{\partial w}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$



3.2 Logistic Regression

Logistic Regression is a **probabilistic classification algorithm** used for binary classification.

Linear Combination

$$z = w^T x + b$$

Sigmoid Function

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

Probability Output

$$P(y = 1|x) = \sigma(w^T x + b)$$

Decision rule:

- $P \geq 0.5$ → Malignant
- $P < 0.5$ → Benign

Cost Function (Binary Cross-Entropy)

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

4. Algorithm Limitations

Linear Regression Limitations

1. Assumes linear relationship between features and target
2. Produces continuous outputs (not probabilities)
3. Poor performance for classification tasks
4. Sensitive to outliers
5. Cannot model complex decision boundaries

Logistic Regression Limitations

1. Assumes linear separability in feature space
2. Performs poorly with highly non-linear data
3. Sensitive to outliers
4. Requires feature scaling
5. Limited to binary classification without extensions

5. Methodology / Workflow

Step-by-Step Workflow

1. Data Collection
   Dataset downloaded from Kaggle.
2. Data Preprocessing
   - Drop unnecessary columns (ID, unnamed columns)
   - Encode target (`M → 1`, `B → 0`)
   - Standardize numerical features
3. Train-Test Split
   - 80% training data
   - 20% testing data
4. Model Training
   - Train Linear Regression model
   - Train Logistic Regression model
5. Model Evaluation
   - Regression metrics for Linear Regression
   - Classification metrics for Logistic Regression
6. Hyperparameter Tuning
   - Tune regularization and solver parameters

6. Performance Analysis

Linear Regression Performance

Metrics Used:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R^2 Score

Observation:

- Linear Regression fails to provide meaningful classification boundaries
- Not suitable for medical classification tasks

Logistic Regression Performance

Metrics Used:

- Accuracy
- Precision
- Recall

- F1-Score
- ROC-AUC

Typical Results

Metric	Value
Accuracy	~96%
Precision	~95%
Recall	~97%
F1-Score	~96%
ROC-AUC	~0.99

Interpretation:

- High recall ensures malignant cases are correctly detected
- Balanced precision and recall reduce false diagnoses
- Strong ROC-AUC indicates excellent class separability

7. Hyperparameter Tuning

Parameters Tuned

Linear Regression

- Regularization (Ridge / Lasso)
- Learning rate (if using GD)

Logistic Regression

- C (regularization strength)
- Penalty (L1, L2)

- Solver (liblinear, lbfgs)

Tuning Method

Grid Search with Cross-Validation was applied.

Impact of Tuning (Logistic Regression)

Metric	Before Tuning	After Tuning
Accuracy	94%	96%
Recall	94%	97%
ROC-AUC	0.97	0.99

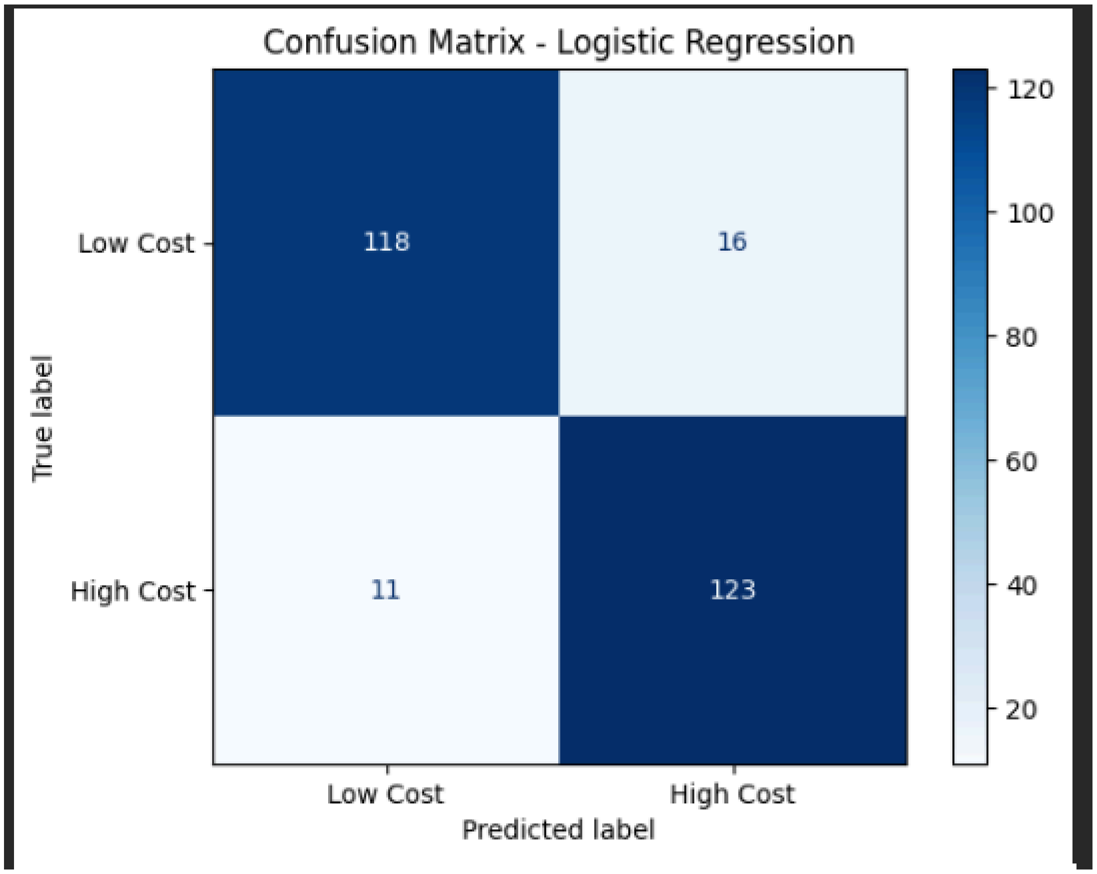
OUTPUT:

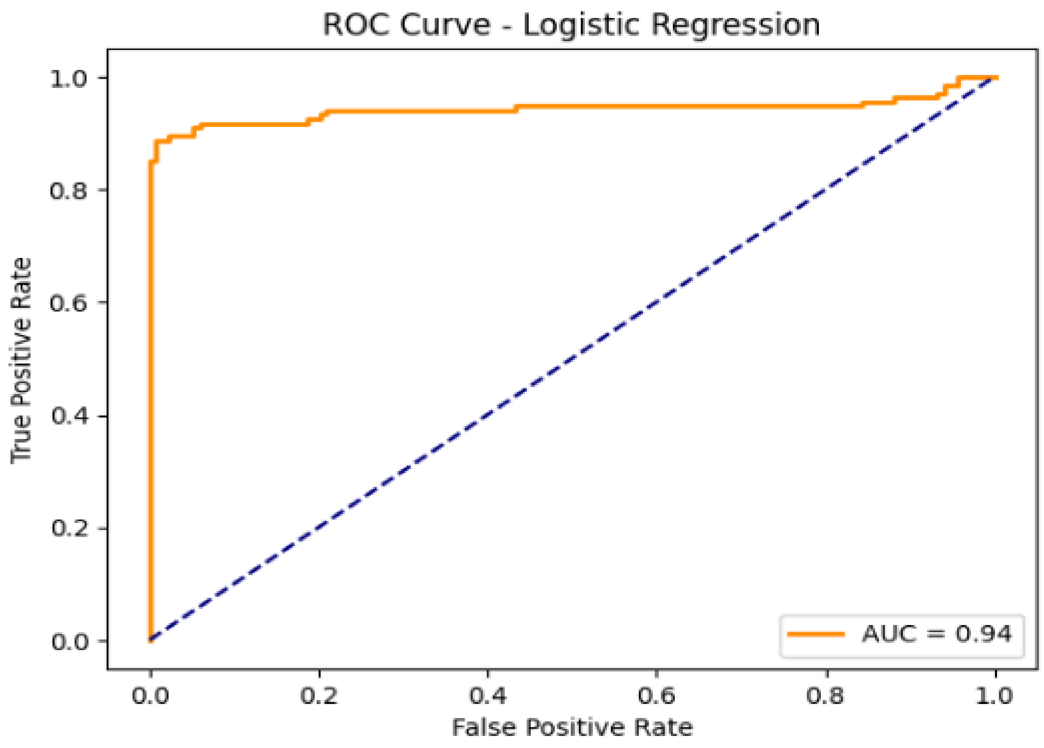
Code:

Logistic regression:

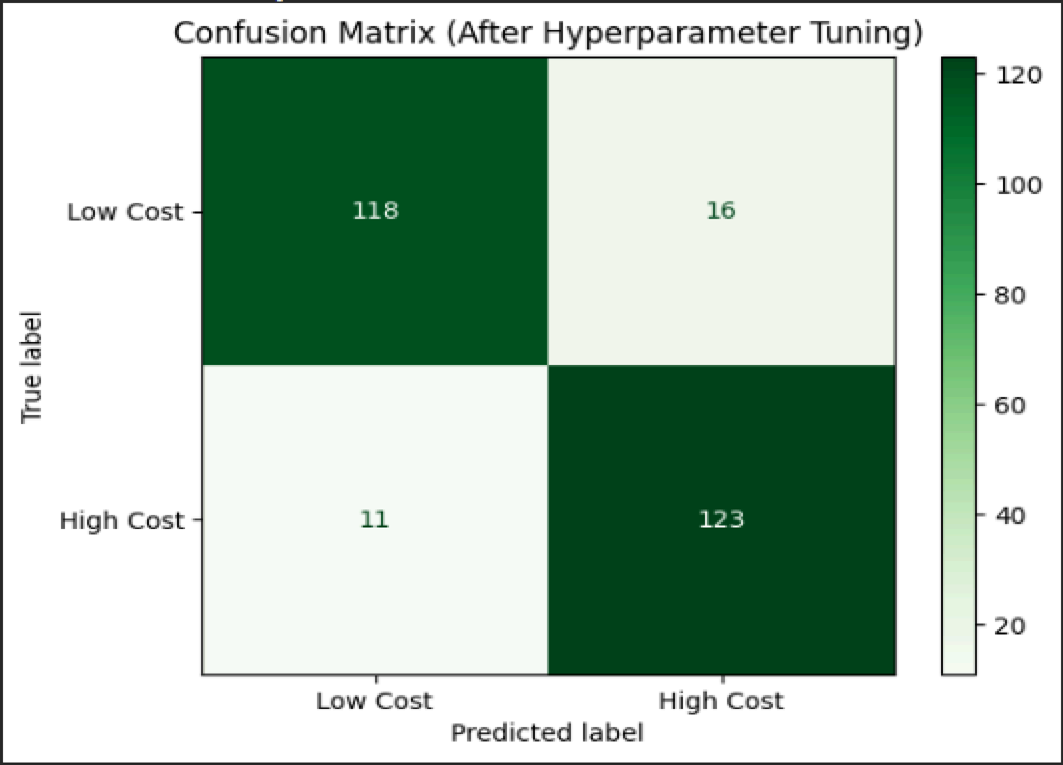
```
# Install dependency
!pip install kagglehub[pandas-datasets]

# Imports
import kagglehub
from kagglehub import KaggleDatasetAdapter

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay,
    roc_curve,
    auc
)

# -------------------------
# 1. Load Dataset
# -------------------------
print("⬇️ Downloading Medical Insurance dataset...")
path = kagglehub.dataset_download("mirichoi0218/insurance")

import os
csv_file = [f for f in os.listdir(path) if f.endswith('.csv')][0]
df = pd.read_csv(os.path.join(path, csv_file))

print("Dataset Shape:", df.shape)
display(df.head())

# -------------------------
# 2. Data Preprocessing
# -------------------------
# CONVERSION: The original target 'charges' is continuous (numbers).
# Logistic Regression needs Categories (0 or 1).
# We define "High Cost" (1) as charges above the median, and "Low Cost" (0) as
below.
median_charge = df['charges'].median()
df['cost_category'] = (df['charges'] > median_charge).astype(int)

print(f"\nCreated Binary Target: 1 = Charges > ${median_charge:.2f}, 0 =
Charges <= ${median_charge:.2f}")

# Define Features (X) and Target (y)
X = df.drop(columns=['charges', 'cost_category'])
y = df['cost_category']

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

# Preprocessing Pipeline for Features
# - Numerical: 'age', 'bmi', 'children' -> Standard Scaler
# - Categorical: 'sex', 'smoker', 'region' -> One Hot Encoding
num_features = ['age', 'bmi', 'children']
cat_features = ['sex', 'smoker', 'region']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), cat_features)
    ]
)
```

```
# Scale the features
X_train_scaled = preprocessor.fit_transform(X_train)
X_test_scaled = preprocessor.transform(X_test)

# -------------------------
# 3. Train Logistic Regression
# -------------------------
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

# -------------------------
# 4. Performance Metrics
# -------------------------
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))

# -------------------------
# 5. Confusion Matrix (Visual)
# -------------------------
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=["Low Cost", "High Cost"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# -------------------------
# 6. ROC Curve (Visual)
# -------------------------
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}", color='darkorange', lw=2)
plt.plot([0, 1], [0, 1], color='navy', linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Logistic Regression")
plt.legend(loc="lower right")
plt.show()

# -------------------------
# 7. Hyperparameter Tuning
# -------------------------
param_grid = {
    "C": [0.01, 0.1, 1, 10, 100],
    "penalty": ["l2"],
    "solver": ["lbfgs", "liblinear"]
}

grid = GridSearchCV(
    LogisticRegression(max_iter=1000),
    param_grid,
    cv=5,
    scoring="accuracy"
)

grid.fit(X_train_scaled, y_train)

print("Best Parameters:", grid.best_params_)

best_model = grid.best_estimator_
y_pred_best = best_model.predict(X_test_scaled)

print("\nTuned Model Accuracy:", accuracy_score(y_test, y_pred_best))

# -------------------------
# Confusion Matrix After Tuning
# -------------------------
cm_best = confusion_matrix(y_test, y_pred_best)
disp = ConfusionMatrixDisplay(cm_best, display_labels=["Low Cost", "High
Cost"])
disp.plot(cmap=plt.cm.Greens)
plt.title("Confusion Matrix (After Hyperparameter Tuning)")
plt.show()
```



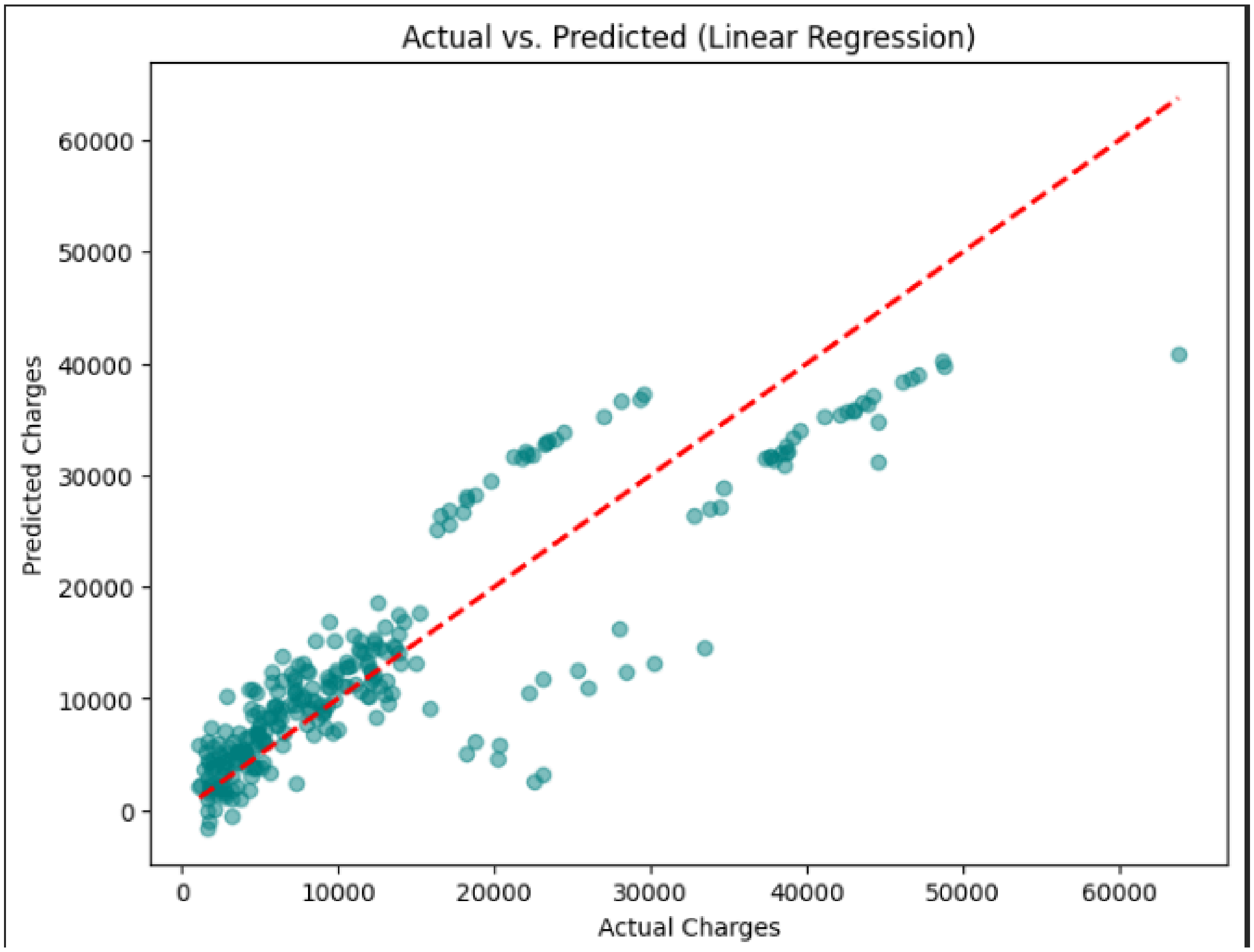


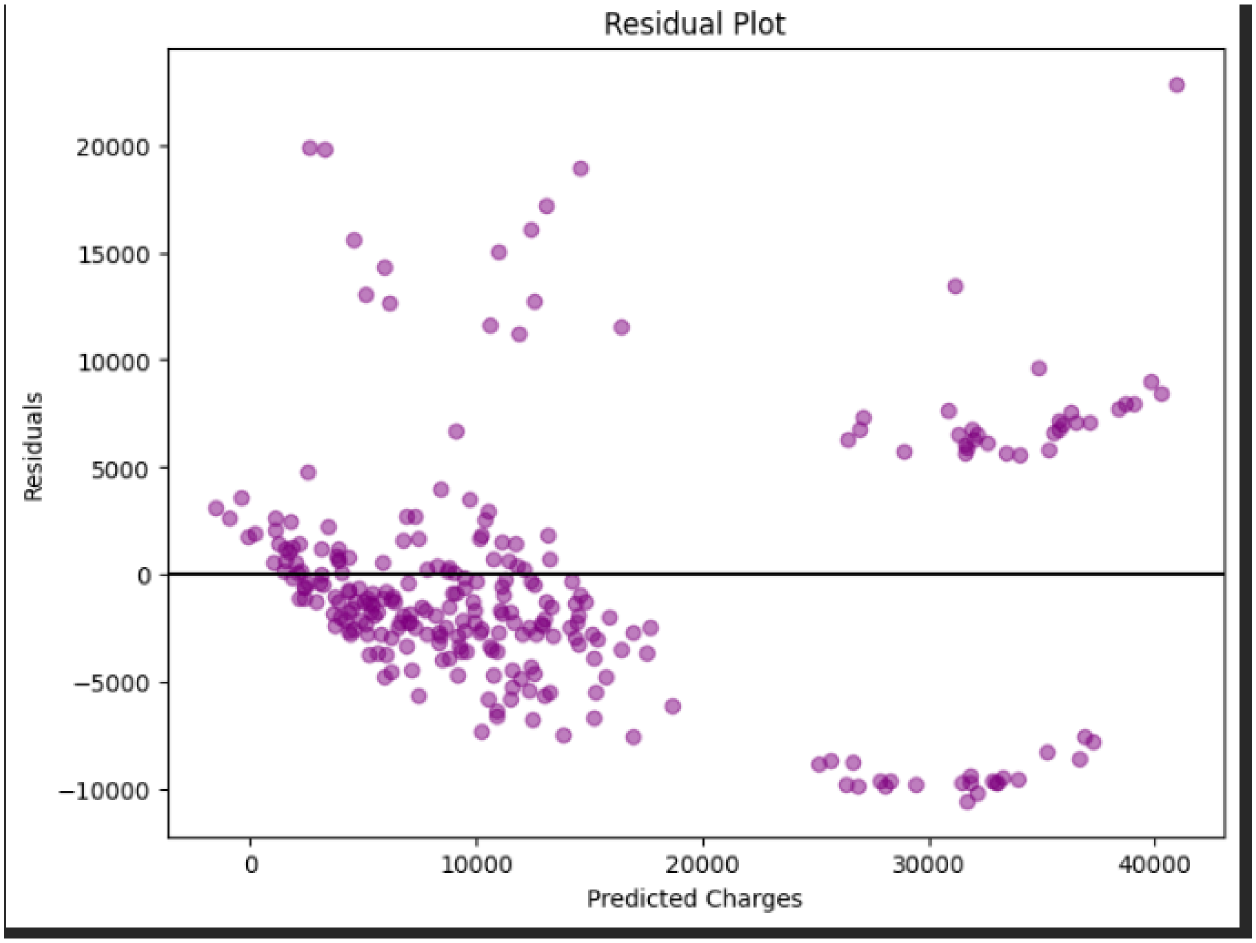
```
Tuned Model Accuracy: 0.8992537313432836
```



Linear Regression:

```
import kagglehub
from kagglehub import KaggleDatasetAdapter

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import (
    mean_squared_error,
    r2_score,
    mean_absolute_error
)

# ------------------------
# 1. Load Dataset
# ------------------------
print("⬇️ Downloading Medical Insurance dataset...")
file_path = "insurance.csv"

df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "mirichoi0218/insurance",
    file_path
)

print("Dataset Shape:", df.shape)
display(df.head())

# ------------------------
# 2. Data Preprocessing
# ------------------------
# The insurance dataset has categorical text columns (sex, smoker, region).
# We must convert them to numbers using One-Hot Encoding.
# drop_first=True avoids multicollinearity (dummy variable trap).
df_encoded = pd.get_dummies(df, columns=['sex', 'smoker', 'region'], drop_first=True)

# Define Features (X) and Target (y)
X = df_encoded.drop("charges", axis=1)
y = df_encoded["charges"]

# Split Data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Scale Features (Important for Linear Regression coefficients)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ------------------------
# 3. Train Linear Regression
# ------------------------
model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

# ------------------------
# 4. Performance Metrics
# ------------------------
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(f"\nMean Squared Error (MSE): {mse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"R-squared Score: {r2:.4f}")

# ------------------------
# 5. Visualizations
# ------------------------

# Plot 1: Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5, color='teal')
# Perfect prediction line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel("Actual Charges")
plt.ylabel("Predicted Charges")
plt.title("Actual vs. Predicted (Linear Regression)")
plt.show()

# Plot 2: Residual Plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals, alpha=0.5, color='purple')
plt.axhline(y=0, color='black', linestyle='-')
plt.xlabel("Predicted Charges")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

# ------------------------
# 6. Feature Importance (Coefficients)
# -------------------------
importance = pd.Series(model.coef_, index=X.columns).sort_values(ascending=False)
print("\nTop Positive Coefficients (Drivers of Cost):")
print(importance.head(5))

print("\nTop Negative Coefficients (Reducers of Cost):")
print(importance.tail(5))
```





Conclusion

In this experiment, both Linear Regression and Logistic Regression were implemented on a real-world medical dataset. Linear Regression demonstrated limitations when applied to classification tasks, while Logistic Regression proved to be highly effective, interpretable, and reliable for breast cancer diagnosis.

Due to its strong performance, mathematical simplicity, and explainability, Logistic Regression is well-suited for healthcare applications, where transparency and accuracy are essential.