Theoretical

In [ ]:
```
# What is Logistic Regression, and how does it differ from Linear Regression.
'''
Logistic Regression and Linear Regression are both supervised learning algorithms,
Regression is used for classification tasks, where the target variable is categoric
is spam or not. It works by estimating the probability that a given input belongs t
function, which maps predictions to a range between 0 and 1. Based on a threshold (
into categories. It uses log loss (binary cross-entropy) as its cost function to op
On the other hand, Linear Regression is used for regression tasks, where the target
predicting house prices. It assumes a linear relationship between independent and d
line to minimize the error using the Mean Squared Error (MSE) cost function. Unlike
probabilities and classifies data, Linear Regression directly predicts numerical va
The key difference is that Logistic Regression predicts probabilities and is used f
predicts continuous numerical values and is used for regression. Additionally, Logi
to map outputs, whereas Linear Regression relies on a linear function to estimate v
```

In [ ]:
```
#What is the mathematical equation of Logistic Regression.
'''
The mathematical equation for Logistic Regression is derived from the linear regres
continuous values, it predicts probabilities using the sigmoid function.
First, we calculate a linear combination of input features:
z =  β0 + β1X1 + β2X2 +...+ βnXn
Where:
z is the linear combination of weights and input features.
β0 and β1 β2...βn  are learned from data.
X1,X2,...Xn are input features.

To ensure the output is a probability (between 0 and 1), we pass z through the sigm
p = 1/(1+ e^(-z))
Where:
p is probality of the positive class
e is base of natural logarithm
z is linear combination of the input features
```

In [ ]:
```
# Why do we use the Sigmoid function in Logistic Regression?
'''
The sigmoid function is used in Logistic Regression because it transforms any real-
classification tasks. Unlike Linear Regression, which can output values beyond this
probability range. It allows us to interpret the result as the likelihood of belong
threshold (e.g., 0.5) to classify data into categories. Additionally, the sigmoid f
using gradient descent. It also prevents extreme outputs by asymptotically approach
defined as σ(z) = 1/(1+ e^(-z)) where z is the linear combination of input features
large values into probabilities close to 0 or 1, making Logistic Regression effecti
```

In [ ]:
```
#What is the cost function of Logistic Regression?
'''
The cost function used in Logistic Regression is the log loss function, also known
Error (MSE), Logistic Regression requires a different approach because its output i
non-convex function with multiple local minima, making optimization difficult. The
match the actual class labels. Mathematically, it is defined as J(β) = m/1 ∑m/i=l[y
and y^(i) is the predicted probability.This function penalizes incorrect prediction
```

```
classes. Since it is convex, optimization algorithms like gradient descent can effi
Regression effective for classification problems.
```

In [ ]:
```
# What is Regularization in Logistic Regression? Why is it needed?
'''
Regularization in Logistic Regression is a technique used to prevent overfitting by
learns the noise in the training data rather than the actual pattern, leading to po
discouraging excessively large coefficients in the model.
Types of Regularization in Logistic Regression:
L1 Regularization (Lasso Regression) = Adds the absolute value of the coefficients
coefficients become exactly zero), making it useful for feature selection.
L2 Regularization (Ridge Regression) = Adds the square of the coefficients as a pen
Why is Regularization Needed?
Prevents Overfitting: Limits the impact of extreme weights, improving generalizatio
Improves Model Simplicity: L1 regularization removes irrelevant features, making th
Enhances Stability: Helps in handling multicollinearity (when independent variables
```

In [ ]:
```
#Explain the difference between Lasso, Ridge, and Elastic Net regression.
'''
Lasso, Ridge, and Elastic Net are regularization techniques used in regression mode
While they share the same goal, they differ in how they apply regularization.

Lasso Regression (L1 Regularization)
Lasso (Least Absolute Shrinkage and Selection Operator) adds the absolute sum of co
exactly to zero, effectively selecting features.
Effect: Some coefficients shrink to zero, removing less important features.
Best For: Feature selection in high-dimensional datasets.
Weakness: Can cause instability when selecting features if variables are highly cor

Ridge Regression (L2 Regularization)
Ridge Regression adds the sum of squared coefficients as a penalty to the cost func
any features completely.
Effect: Reduces the magnitude of coefficients but retains all features.
Best For: Handling multicollinearity (highly correlated independent variables).
Weakness: Does not perform feature selection since coefficients are only shrunk, no

 Elastic Net Regression (Combination of L1 & L2)
 Elastic Net combines both Lasso (L1) and Ridge (L2) penalties, balancing feature s
 correlated variables and a large number of predictors.
 Effect: Selects important features (like Lasso) while preventing too much shrinkag
Best For: High-dimensional datasets where feature selection and multicollinearity a
Weakness: Requires tuning two parameters
```

In [ ]:
```
#When should we use Elastic Net instead of Lasso or Ridge?
'''
We should use Elastic Net instead of Lasso or Ridge when you have highly correlated
It combines the strengths of both L1 (Lasso) and L2 (Ridge) regularization, making
while also shrinking the coefficients to prevent overfitting. Elastic Net is partic
to balance between shrinkage and feature selection. It is a good choice when both R
```

In [ ]:
```
#What is the impact of the regularization parameter (λ) in Logistic Regression?
'''
The regularization parameter (\( \lambda \)) in Logistic Regression controls the st
model's ability to generalize. A larger \( \lambda \) increases regularization, shr
simpler and less sensitive to noise in the data. However, if \( \lambda \) is too l
```

```
unable to capture important patterns. Conversely, a smaller \( \lambda \) allows th
the risk of overfitting, especially in noisy datasets. Thus, finding the optimal \(
variance, ensuring the model generalizes well to unseen data.
```

In [ ]:
```
#What are the key assumptions of Logistic Regression?
'''
The key assumptions of Logistic Regression are:

1. Binary Dependent Variable:
   Logistic Regression assumes that the dependent variable (target) is binary, mean

2. Linearity of the Log-Odds:
   It assumes that there is a linear relationship between the independent variables
   log-transformed probability of the outcome is a linear combination of the predic

3. Independence of Observations:
   The observations in the dataset must be independent of each other. This means th
   another.

4. No Multicollinearity:
   Logistic Regression assumes that there is no perfect multicollinearity among the
   lead to issues in estimating the model coefficients accurately.

5. Large Sample Size:
   Logistic Regression generally works best with a large sample size because it rel
   lead to unstable or biased estimates.

6. Little or No Outliers:
   Logistic Regression assumes that there are no extreme outliers in the predictor
   coefficients.

7. Homoscedasticity (for large sample sizes):
   While not strictly necessary for Logistic Regression, it is typically assumed th
   variables, especially for larger sample sizes. This assumption helps ensure more
```

In [ ]:
```
#What are some alternatives to Logistic Regression for classification tasks?
'''
Several alternatives to Logistic Regression can be used for classification tasks, d
include:

1. Decision Trees
- Description: Decision trees split the data into subsets based on the most signifi
represents a class label.
- Pros: Easy to interpret, handles both numerical and categorical data, and capture
- Cons: Can overfit easily, especially with deep trees, and is sensitive to small v

2. Random Forest
- Description: An ensemble of decision trees, where each tree is trained on a rando
(for classification).
- Pros: Reduces overfitting compared to individual decision trees, robust, handles
- Cons: Less interpretable than a single decision tree, and can be computationally

3.Support Vector Machines (SVM)
- Description: SVM finds the optimal hyperplane that maximizes the margin between d
to non-linear problems using kernels.
```

```
- Pros: High accuracy, especially for complex or high-dimensional data, and effecti
- Cons: Computationally expensive for large datasets, and the choice of the kernel

 4.K-Nearest Neighbors (KNN)
-Description: KNN is a simple, non-parametric algorithm that assigns a class label
given data point.
- Pros: Simple, intuitive, and non-parametric (no assumption about the data distrib
- Cons: Computationally expensive at prediction time, sensitive to the choice of \(

5. Naive Bayes
- Description: Based on Bayes' theorem, Naive Bayes assumes that the features are i
assigns the class with the highest probability.
- Pros: Fast, efficient, and works well with high-dimensional data, especially when
- Cons: Assumption of feature independence is often unrealistic, and it performs po

 6. Gradient Boosting Machines (GBM)
- Description: GBM is an ensemble technique where models are built sequentially, wi
like XGBoost, LightGBM, and CatBoost.
- Pros: High predictive accuracy, robust to overfitting, handles both numerical and
- Cons: Computationally intensive, complex to tune, and less interpretable.

7.Artificial Neural Networks (ANNs)
- Description: A class of models inspired by the human brain, where multiple layers
particularly when dealing with unstructured data (e.g., images, text).
- Pros: Can model highly complex, non-linear relationships and perform well with la
- Cons: Requires large datasets and significant computational resources, hard to in

8.Linear Discriminant Analysis (LDA)
- Description: LDA is a statistical technique that seeks to find the linear combina
the data is assumed to follow a normal distribution.
- Pros: Works well for normally distributed data, simple to implement, and interpre
- Cons: Assumes normally distributed classes, which might not always hold in practi

9.Quadratic Discriminant Analysis (QDA)
- Description: Similar to LDA, but assumes that each class has its own covariance m
structures for each class.
- Pros: More flexible than LDA, works well with non-linearly separable data.
- Cons: Requires more data for training, as the covariance matrix for each class ne
```

In [ ]:

```
#What are Classification Evaluation Metrics?
'''
Classification evaluation metrics are used to assess the performance of a classific
the most common metric, representing the ratio of correct predictions to the total
measures how many of the predicted positive cases are actually positive, making it
Recall (Sensitivity), on the other hand, evaluates how many actual positive cases w
diagnosis where missing a positive case could be dangerous. The F1-Score is the har
with imbalanced datasets. Specificity assesses how well the model identifies negati
The ROC Curve plots the True Positive Rate against the False Positive Rate, and the
classes, with a higher value signifying better performance. Log Loss (Cross-Entropy
where a lower value indicates better model calibration. Lastly, the Matthews Correl
classification, especially when dealing with imbalanced datasets. Choosing the righ
positives and false negatives.
```

In [ ]:

```
#How does class imbalance affect Logistic Regression?
'''
```

```
Class imbalance affects Logistic Regression in several ways:

Bias Towards Majority Class – Since Logistic Regression aims to minimize overall er
performance on the minority class.

Misleading Accuracy – If a dataset is heavily imbalanced (e.g., 95% class A, 5% cla
in distinguishing class B.

Poorly Calibrated Probabilities – The predicted probabilities may not be well-calib

Skewed Decision Boundary – The model may set the decision boundary too close to the
```

In [ ]:

```
#What is Hyperparameter Tuning in Logistic Regression?
'''
Hyperparameter tuning in Logistic Regression involves optimizing parameters that co
affect model performance, regularization, and convergence speed.
Key Hyperparameters to Tune

Regularization Strength (C)
Controls the inverse of regularization (L1/L2 penalty).
Higher C → Less regularization (risk of overfitting).
Lower C → More regularization (risk of underfitting).
Default in sklearn: C = 1.0.

Penalty Type (penalty)
Regularization technique:
l1 (Lasso) → Useful for feature selection (forces some coefficients to zero).
l2 (Ridge) → Shrinks coefficients but keeps all features.
elasticnet → Combination of L1 & L2.
none → No regularization

Solver (solver)
Controls optimization algorithm:
liblinear – Works with small datasets (supports L1 & L2).
saga – Efficient for large datasets (supports L1, L2, and elasticnet).
lbfgs – Best for L2 regularization & multiclass problems.
newton-cg – Works well for L2 and large datasets.

Class Weight (class_weight)
Adjusts importance of classes (useful for imbalanced datasets).
balanced → Assigns weights inversely proportional to class frequencies.
{0:1, 1:10} → Custom weights.
```

In [ ]:

```
#What are different solvers in Logistic Regression? Which one should be used?
'''
Logistic Regression in machine learning (e.g., in sklearn.linear_model.LogisticRegr
datasets and use cases.

1. liblinear
Type: Coordinate Descent (uses L1 or L2 regularization)
Best for: Small datasets
Supports: L1 & L2 regularization
Pros: Works well for smaller datasets and sparse datasets
Cons: Does not support multinomial loss (only binary classification)
```

```
2. newton-cg (Newton Conjugate Gradient)
Type: Second-order optimization (uses Hessian matrix)
Best for: Large datasets with many features
Supports: L2 regularization
Pros: Good for multiclass problems
Cons: Computationally expensive for very large datasets

3. lbfgs (Limited-memory BFGS)
Type: Quasi-Newton method (approximates Hessian matrix)
Best for: Large datasets
Supports: L2 regularization
Pros: Works well for multinomial classification
Cons: Can be slower for very high-dimensional data

4. sag (Stochastic Average Gradient)
Type: Stochastic Gradient Descent (SGD-like)
Best for: Very large datasets
Supports: L2 regularization
Pros: Faster for large-scale datasets (online learning)
Cons: Works best for large datasets with many features

5. saga (Variant of SAG with L1 support)
Type: Stochastic Gradient Descent (SGD-like)
Best for: Large, high-dimensional datasets
Supports: L1, L2, and Elastic Net regularization
Pros: Works well for sparse data and L1 regularization
Cons: Can be slower for small datasets

Which Solver to Use?
Small dataset = liblinear
Large dataset (few features) =  lbfgs or newton-cg
Large dataset (many features) =         sag or saga
Sparse dataset  =       liblinear or saga
Multiclass classification =     lbfgs, newton-cg, or saga
L1 Regularization        = liblinear or saga
Online learning (streaming data) =  sag or saga
```

In [ ]:

```
#How is Logistic Regression extended for multiclass classification?
'''
Logistic Regression is naturally a binary classifier, but it can be extended for mu
Multinomial (Softmax) Regression. In OvR, separate binary logistic regression model
rest. The class with the highest probability is selected. This approach is simple a
other hand, Softmax Regression (Multinomial Logistic Regression) directly models al
probabilities for each class simultaneously. This approach is theoretically optimal
expensive. In Scikit-learn, OvR is the default when using the "liblinear" solver, w
works well for small datasets, whereas Softmax is preferable for large datasets and
```

In [ ]:

```
#What are the advantages and disadvantages of Logistic Regression?
'''
Advantages and Disadvantages of Logistic Regression

Advantages
1.Simple and Interpretable – Logistic Regression is easy to implement and provides
target.
2.Efficient for Small Datasets – It performs well on small to moderately sized data
```

```
Networks.
3.Probabilistic Output – It provides probability scores, which are useful for ranki
4.Handles Linearly Separable Data Well – If the classes are linearly separable, Log
5.Regularization Support – It supports L1 (Lasso) and L2 (Ridge) regularization, he
6.Works Well for Binary and Multiclass Problems – Using One-vs-Rest (OvR) or Multin
tasks.

Disadvantages
1.Assumes Linearity – Logistic Regression assumes a linear relationship between the
for highly complex, non-linear data.
2.Not Suitable for High-Dimensional Data – When the number of features is very larg
selection or dimensionality reduction.
3.Sensitive to Outliers – Logistic Regression is sensitive to extreme values, which
4.Limited Expressiveness – It struggles with capturing complex relationships and in
5.Not the Best for Large Datasets – When the dataset is extremely large, models lik
6.Feature Engineering Required – Performance heavily depends on properly selecting
```

In [ ]:

```
#What are some use cases of Logistic Regression?
'''
Use Cases of Logistic Regression

1.Medical Diagnosis – Logistic Regression is widely used in predicting diseases, su
medical attributes like age, blood pressure, and cholesterol levels.

2.Credit Scoring & Fraud Detection – Banks and financial institutions use it to pre
analyzing transaction patterns and customer demographics.

3.Marketing & Customer Churn Prediction – Businesses use Logistic Regression to pre
service based on past behavior, usage patterns, and demographics.

4.Spam Email Detection – Email service providers use it to classify emails as spam
links.

5.Employee Attrition Prediction – HR departments use Logistic Regression to analyze
based on salary, job satisfaction, and work environment.

6.Political Campaigning – It is used to predict whether a voter is likely to vote f
survey responses.

7.Image Recognition – Logistic Regression can be used in simple image classificatio

8.Weather Prediction – It can help in predicting binary weather conditions, such as
wind speed.

9.Medical Treatment Effectiveness – It is used in clinical trials to determine whet
their medical history and other attributes.

10.Social Media Sentiment Analysis – Logistic Regression can classify social media
understand customer sentiment.
```

In [ ]:

```
#What is the difference between Softmax Regression and Logistic Regression?
'''
Logistic Regression is primarily used for binary classification, while Softmax Regr
Regression applies the sigmoid function to produce a probability for one class, wit
contrast, Softmax Regression uses the softmax function, which assigns probabilities
```

```
Logistic Regression is trained using binary cross-entropy loss, whereas Softmax Reg
Regression can be extended to multiclass problems using One-vs-Rest (OvR), while So
If a task involves two classes, Logistic Regression is sufficient, but for multicla
all classes simultaneously.
```

In [ ]:
```
#How do we choose between One-vs-Rest (OvR) and Softmax for multiclass classificati
'''
Choosing between One-vs-Rest (OvR) and Softmax (Multinomial Logistic Regression) fo
interpretability, and computational efficiency.

One-vs-Rest (OvR) is suitable when the dataset is small or when interpretability is
making it easier to analyze individual class predictions. It is computationally les
classes is very large, as it requires training K binary models for K classes.

On the other hand, Softmax Regression (Multinomial Logistic Regression) is more eff
multiple categories simultaneously, leading to better probability estimates. It als
However, it is computationally more expensive and may not always be necessary when

In Scikit-learn, if the solver is "liblinear", only OvR is supported, whereas "lbfg
For small datasets or when interpretability matters, OvR is a good choice, while fo
Softmax is preferred.
```

In [ ]:
```
#How do we interpret coefficients in Logistic Regression?
'''
To interpret the coefficients in a more intuitive way, we exponentiate them to get
increases the likelihood of the event occurring, whereas if it is less than 1, the
coefficient of 0.03, its odds ratio is e 0.03 ≈ 1.03, meaning that a one-unit incre
one category is treated as a reference, and the coefficients show how much more or
the reference. In multiclass logistic regression (Softmax Regression), coefficients
set of coefficients comparing it to the reference. Overall, interpreting logistic r
understand their impact on the probability of an event occurring.
```

Practical

In [4]:
```
#Write a Python program that loads a dataset, splits it into training and testing s
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris


iris = load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200)
model.fit(X_train, y_train)


y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.93
```

In [8]:
```
#Write a Python program to apply L1 regularization (Lasso) on a dataset using Logis


iris = load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression model with L1 Regularization (Lasso)
model = LogisticRegression(penalty='l1', solver='saga', multi_class='multinomial',
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy with L1 Regularization: {accuracy:.2f}")
```

```
Model Accuracy with L1 Regularization: 0.93
```

```
/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklearn/linea
r_model/_sag.py:350: ConvergenceWarning: The max_iter was reached which means the co
ef_ did not converge
  warnings.warn(
```

In [1]:
```
#Write a Python program to train Logistic Regression with L2 regularization (Ridge)

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

np.random.seed(42)
X = np.random.rand(500, 5)
y = np.random.randint(0, 2, 500)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression(penalty='l2', solver='liblinear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")

print("Coefficients:")
print(model.coef_)
```

```
Model Accuracy: 0.4800
Coefficients:
[[-0.13769375 -0.08245199  0.04618936 -0.12626757  0.02946699]]
```

In [4]:

```
#Write a Python program to train Logistic Regression with Elastic Net Regularizatio

np.random.seed(42)
X = np.random.rand(500, 5)
y = np.random.randint(0, 2, 500)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression(penalty='elasticnet', solver='saga', l1_ratio=0.5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")

print("Coefficients:")
print(model.coef_)
```

```
Model Accuracy: 0.4800
Coefficients:
[[-0.13297664 -0.0773236   0.04093348 -0.12203169  0.02368321]]
```

In [6]:

```
#Write a Python program to train a Logistic Regression model for multiclass classif

np.random.seed(42)
X = np.random.rand(500, 5)
y = np.random.randint(0, 3, 500)  # Multiclass classification with 3 classes

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression(penalty='l2', solver='liblinear', multi_class='ovr')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")

print("Coefficients:")
print(model.coef_)
```

```
Model Accuracy: 0.3100
Coefficients:
[[ 0.15589535  0.08428401 -0.0017287   0.01269038 -0.00376156]
 [-0.17100577 -0.10926698 -0.06751772 -0.06974533 -0.02592043]
 [ 0.00272837  0.01797608  0.0681597   0.05659553  0.03026593]]
```

In [10]:
```
#Write a Python program to apply GridSearchCV to tune the hyperparameters (C and pe
from sklearn.model_selection import  GridSearchCV
np.random.seed(42)
X = np.random.rand(500, 5)
y = np.random.randint(0, 3, 500)  # Multiclass classification with 3 classes

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}

model = LogisticRegression(solver='liblinear', multi_class='ovr')
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Model Accuracy: {accuracy:.4f}")

print("Coefficients:")
print(best_model.coef_)
```

```
Best Parameters: {'C': 0.01, 'penalty': 'l1'}
Model Accuracy: 0.2500
Coefficients:
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

In [12]:

```python
#Write a Python program to evaluate Logistic Regression using Stratified K-Fold Cro

from sklearn.model_selection import StratifiedKFold, cross_val_score


np.random.seed(42)
X = np.random.rand(500, 5)
y = np.random.randint(0, 3, 500)

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

model = LogisticRegression(solver='liblinear', multi_class='ovr')
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores = cross_val_score(model, X, y, cv=skf, scoring='accuracy')

print(f"Average Accuracy: {scores.mean():.4f}")
```

```
Average Accuracy: 0.3180
```

In [19]:

```python
#Write a Python program to load a dataset from a CSV file, apply Logistic Regressio
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


file_path = 'spotify.csv'
df = pd.read_csv(file_path)


non_numeric_columns = df.select_dtypes(include=['object']).columns
if not non_numeric_columns.empty:
    print(f"Non-numeric columns found: {list(non_numeric_columns)}")


label_encoders = {}
for col in non_numeric_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))
    label_encoders[col] = le


y = df.iloc[:, -1]
X = df.iloc[:, :-1]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = LogisticRegression(solver='liblinear', multi_class='ovr')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
```

```
Non-numeric columns found: ['Artist', 'Track Name', 'Track ID']
Model Accuracy: 0.0000
```

In [15]:
```
#Write a Python program to apply RandomizedSearchCV for tuning hyperparameters (C,
#Print the best parameters and accuracy
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsOneClassifier
from sklearn.metrics import accuracy_score


iris = datasets.load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


param_grid = {
    'estimator__C': [0.01, 0.1, 1, 10, 100],
    'estimator__penalty': ['l1', 'l2'],
    'estimator__solver': ['liblinear', 'lbfgs']
}


ovo_clf = OneVsOneClassifier(LogisticRegression(max_iter=5000))
grid_search = GridSearchCV(ovo_clf, param_grid, cv=5, scoring='accuracy', n_jobs=-1
grid_search.fit(X_train, y_train)
```

```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f'Best Parameters: {best_params}')
print(f'Accuracy: {accuracy:.4f}')
```

```
Best Parameters: {'estimator__C': 1, 'estimator__penalty': 'l2', 'estimator__solve
r': 'liblinear'}
Accuracy: 1.0000
```

```
/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklearn/model
_selection/_validation.py:425: FitFailedWarning:
25 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
25 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/model_selection/_validation.py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/base.py", line 1151, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/multiclass.py", line 704, in fit
    Parallel(n_jobs=self.n_jobs)(
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/utils/parallel.py", line 65, in __call__
    return super().__call__(iterable_with_config)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/parallel.py", line 1085, in __call__
    if self.dispatch_one_batch(iterator):
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/parallel.py", line 901, in dispatch_one_batch
    self._dispatch(tasks)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/parallel.py", line 819, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/_parallel_backends.py", line 208, in apply_async
    result = ImmediateResult(func)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/_parallel_backends.py", line 597, in __init__
    self.results = batch()
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/parallel.py", line 288, in __call__
    return [func(*args, **kwargs)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/jobli
b/parallel.py", line 288, in <listcomp>
    return [func(*args, **kwargs)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/utils/parallel.py", line 127, in __call__
    return self.function(*args, **kwargs)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/multiclass.py", line 560, in _fit_ovo_binary
    _fit_binary(
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/multiclass.py", line 90, in _fit_binary
    estimator.fit(X, y)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/base.py", line 1151, in wrapper
    return fit_method(estimator, *args, **kwargs)
```

```
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/linear_model/_logistic.py", line 1168, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklea
rn/linear_model/_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklearn/model
_selection/_search.py:976: UserWarning: One or more of the test scores are non-finit
e: [0.33333333        nan 0.81666667 0.825      0.73333333        nan
 0.875      0.90833333 0.95              nan 0.95833333 0.95
 0.95              nan 0.95833333 0.95       0.95              nan
 0.95833333 0.95      ]
  warnings.warn(
```

In [1]:

```python
# Write a Python program to implement One-vs-One (OvO) Multiclass Logistic Regressi
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsOneClassifier
from sklearn.metrics import accuracy_score


iris = datasets.load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


ovo_clf = OneVsOneClassifier(LogisticRegression(max_iter=200))
ovo_clf.fit(X_train, y_train)


y_pred = ovo_clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')
```

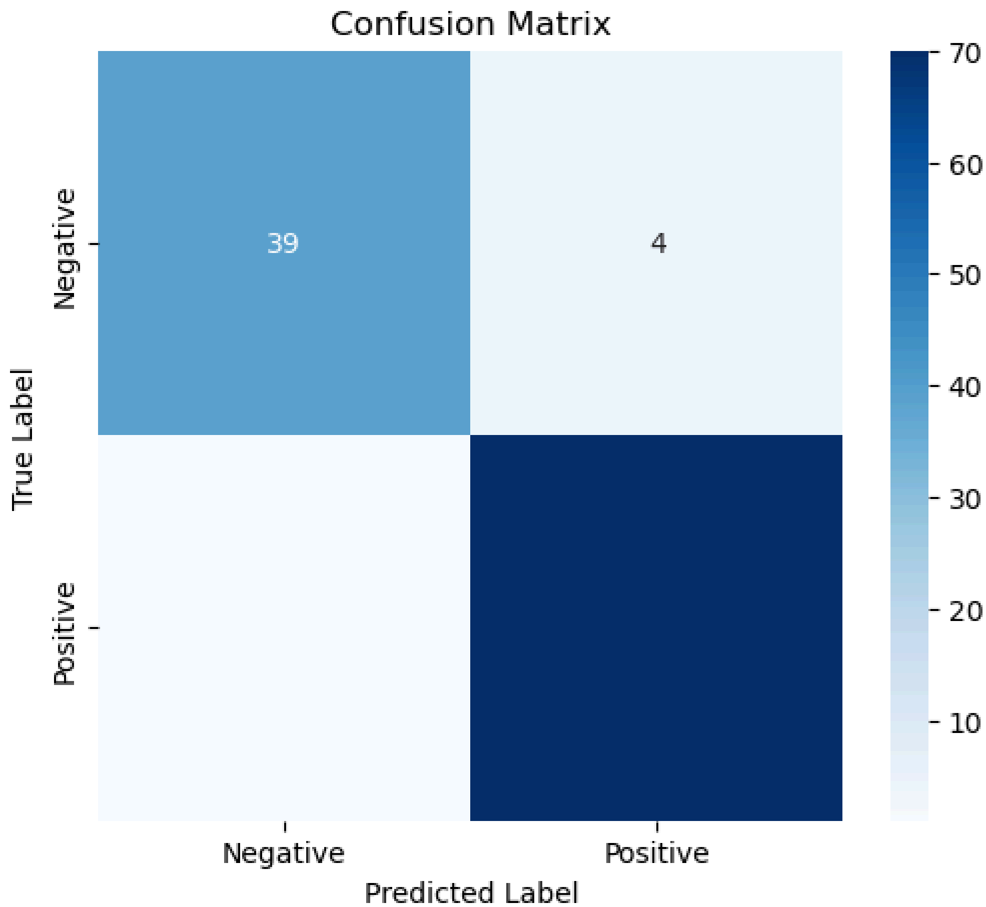
```
Accuracy: 1.0000
```

In [17]:

```python
#Write a Python program to train a Logistic Regression model and visualize the conf
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 LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.datasets import load_breast_cancer


data = load_breast_cancer()
X = data.data
y = data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)


y_pred = model.predict(X_test)


cm = confusion_matrix(y_test, y_pred)


plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Negative", "Positi
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()


print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
/opt/conda/envs/anaconda-ai-2024.04-py310/lib/python3.10/site-packages/sklearn/linea
r_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

In [20]:

```
#Write a Python program to train a Logistic Regression model and evaluate its perfo
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score


data = load_iris()
X, y = data.data, data.target


y = (y == 0).astype(int)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = LogisticRegression()
model.fit(X_train, y_train)


y_pred = model.predict(X_test)


precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
```

In [23]:
```
#Write a Python program to train a Logistic Regression model on imbalanced data and
#performance
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.utils import class_weight
import numpy as np


data = load_iris()
X, y = data.data, data.target


y = (y == 0).astype(int)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


class_weights = dict(enumerate(class_weight.compute_class_weight('balanced', classe


model = LogisticRegression(class_weight=class_weights)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)


precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Precision: 1.00
Recall: 1.00
F1-Score: 1.00
```

In [25]:

```
#Write a Python program to train Logistic Regression on the Titanic dataset, handle
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.utils import class_weight
import numpy as np


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


class_weights = dict(enumerate(class_weight.compute_class_weight('balanced', classe


model = LogisticRegression(class_weight=class_weights)
```

```
model.fit(X_train, y_train)


y_pred = model.predict(X_test)


precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Precision: 0.73
Recall: 0.78
F1-Score: 0.76
```

In [29]:

```
#Write a Python program to apply feature scaling (Standardization) before training
#Evaluate its accuracy and compare results with and without scaling
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.utils import class_weight
import numpy as np


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


model_no_scaling = LogisticRegression(class_weight='balanced', max_iter=500)
model_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = model_no_scaling.predict(X_test)
accuracy_no_scaling = accuracy_score(y_test, y_pred_no_scaling)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model_with_scaling = LogisticRegression(class_weight='balanced', max_iter=500)
model_with_scaling.fit(X_train_scaled, y_train)
y_pred_with_scaling = model_with_scaling.predict(X_test_scaled)
accuracy_with_scaling = accuracy_score(y_test, y_pred_with_scaling)


precision = precision_score(y_test, y_pred_with_scaling)
recall = recall_score(y_test, y_pred_with_scaling)
f1 = f1_score(y_test, y_pred_with_scaling)

print(f"Accuracy without Scaling: {accuracy_no_scaling:.2f}")
print(f"Accuracy with Scaling: {accuracy_with_scaling:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy without Scaling: 0.80
Accuracy with Scaling: 0.80
Precision: 0.72
Recall: 0.78
F1-Score: 0.75
```

In [31]:

```
#Write a Python program to train Logistic Regression and evaluate its performance u
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.utils import class_weight
import numpy as np


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
model_no_scaling = LogisticRegression(class_weight='balanced', max_iter=500)
model_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = model_no_scaling.predict(X_test)
y_pred_proba_no_scaling = model_no_scaling.predict_proba(X_test)[:, 1]
accuracy_no_scaling = accuracy_score(y_test, y_pred_no_scaling)
roc_auc_no_scaling = roc_auc_score(y_test, y_pred_proba_no_scaling)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model_with_scaling = LogisticRegression(class_weight='balanced', max_iter=500)
model_with_scaling.fit(X_train_scaled, y_train)
y_pred_with_scaling = model_with_scaling.predict(X_test_scaled)
y_pred_proba_with_scaling = model_with_scaling.predict_proba(X_test_scaled)[:, 1]
accuracy_with_scaling = accuracy_score(y_test, y_pred_with_scaling)
roc_auc_with_scaling = roc_auc_score(y_test, y_pred_proba_with_scaling)


precision = precision_score(y_test, y_pred_with_scaling)
recall = recall_score(y_test, y_pred_with_scaling)
f1 = f1_score(y_test, y_pred_with_scaling)

print(f"Accuracy without Scaling: {accuracy_no_scaling:.2f}")
print(f"ROC-AUC without Scaling: {roc_auc_no_scaling:.2f}")
print(f"Accuracy with Scaling: {accuracy_with_scaling:.2f}")
print(f"ROC-AUC with Scaling: {roc_auc_with_scaling:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy without Scaling: 0.80
ROC-AUC without Scaling: 0.85
Accuracy with Scaling: 0.80
ROC-AUC with Scaling: 0.84
Precision: 0.72
Recall: 0.78
F1-Score: 0.75
```

In [33]:

```
#Write a Python program to train Logistic Regression using a custom learning rate (


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master

features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]

df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)
```

```
X = df.drop(columns=['Survived'])
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

model_no_scaling = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5)
model_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = model_no_scaling.predict(X_test)
y_pred_proba_no_scaling = model_no_scaling.predict_proba(X_test)[:, 1]
accuracy_no_scaling = accuracy_score(y_test, y_pred_no_scaling)
roc_auc_no_scaling = roc_auc_score(y_test, y_pred_proba_no_scaling)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model_with_scaling = LogisticRegression(class_weight='balanced', max_iter=500, C=0.
model_with_scaling.fit(X_train_scaled, y_train)
y_pred_with_scaling = model_with_scaling.predict(X_test_scaled)
y_pred_proba_with_scaling = model_with_scaling.predict_proba(X_test_scaled)[:, 1]
accuracy_with_scaling = accuracy_score(y_test, y_pred_with_scaling)
roc_auc_with_scaling = roc_auc_score(y_test, y_pred_proba_with_scaling)

precision = precision_score(y_test, y_pred_with_scaling)
recall = recall_score(y_test, y_pred_with_scaling)
f1 = f1_score(y_test, y_pred_with_scaling)

print(f"Accuracy without Scaling (C=0.5): {accuracy_no_scaling:.2f}")
print(f"ROC-AUC without Scaling (C=0.5): {roc_auc_no_scaling:.2f}")
print(f"Accuracy with Scaling (C=0.5): {accuracy_with_scaling:.2f}")
print(f"ROC-AUC with Scaling (C=0.5): {roc_auc_with_scaling:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy without Scaling (C=0.5): 0.81
ROC-AUC without Scaling (C=0.5): 0.85
Accuracy with Scaling (C=0.5): 0.80
ROC-AUC with Scaling (C=0.5): 0.84
Precision: 0.72
Recall: 0.78
F1-Score: 0.75
```

In [35]:

```
#Write a Python program to train Logistic Regression and identify important feature

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master

features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)
```

```
X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)


feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_
feature_importance = feature_importance.sort_values(by='Coefficient', ascending=Fal

print(f"Accuracy (C=0.5): {accuracy:.2f}")
print(f"ROC-AUC (C=0.5): {roc_auc:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("\nFeature Importance:")
print(feature_importance)
```

```
Accuracy (C=0.5): 0.80
ROC-AUC (C=0.5): 0.84
Precision: 0.72
Recall: 0.78
F1-Score: 0.75

Feature Importance:
      Feature  Coefficient
4        Fare     0.137005
6  Embarked_Q     0.056581
3       Parch    -0.071481
7  Embarked_S    -0.179506
2       SibSp    -0.259174
1         Age    -0.495355
0      Pclass    -0.854869
5    Sex_male    -1.221190
```

In [37]:
```
#Write a Python program to train Logistic Regression and evaluate its performance u

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

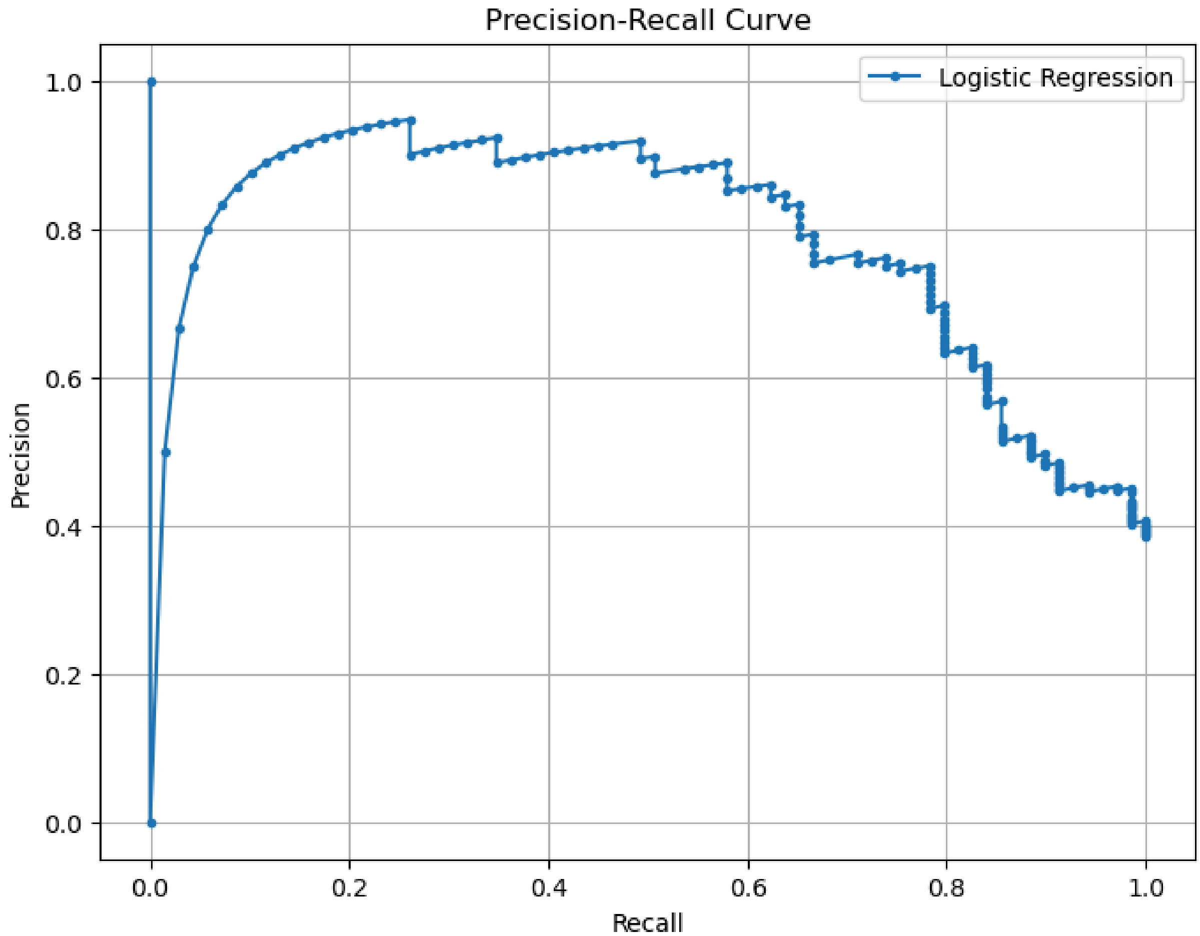
```
df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master

features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]

df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)

X = df.drop(columns=['Survived'])
y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)
kappa = cohen_kappa_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_
feature_importance = feature_importance.sort_values(by='Coefficient', ascending=Fal

print(f"Accuracy (C=0.5): {accuracy:.2f}")
print(f"ROC-AUC (C=0.5): {roc_auc:.2f}")
print(f"Cohen's Kappa Score: {kappa:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("\nFeature Importance:")
print(feature_importance)
```

```
Accuracy (C=0.5): 0.80
ROC-AUC (C=0.5): 0.84
Cohen's Kappa Score: 0.58
Precision: 0.72
Recall: 0.78
F1-Score: 0.75

Feature Importance:
      Feature  Coefficient
4        Fare     0.137005
6  Embarked_Q     0.056581
3       Parch    -0.071481
7  Embarked_S    -0.179506
2       SibSp    -0.259174
1         Age    -0.495355
0      Pclass    -0.854869
5    Sex_male    -1.221190
```

In [39]:

```python
#Write a Python program to train Logistic Regression and visualize the Precision-Re

import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

```
kappa = cohen_kappa_score(y_test, y_pred)

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)


feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': model.coef_
feature_importance = feature_importance.sort_values(by='Coefficient', ascending=Fal


precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba)
plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()

print(f"Accuracy (C=0.5): {accuracy:.2f}")
print(f"ROC-AUC (C=0.5): {roc_auc:.2f}")
print(f"Cohen's Kappa Score: {kappa:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("\nFeature Importance:")
print(feature_importance)
```



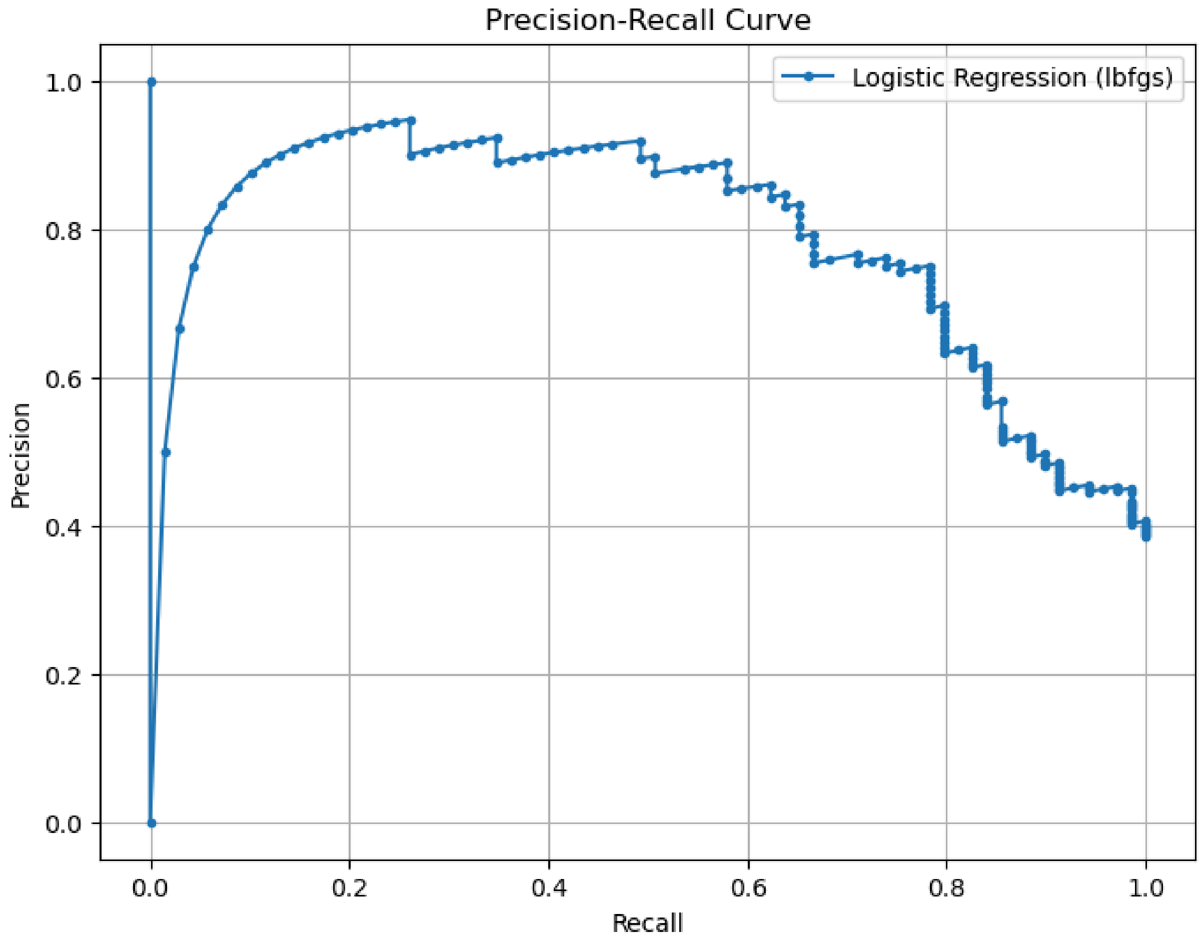

```
Accuracy (C=0.5): 0.80
ROC-AUC (C=0.5): 0.84
Cohen's Kappa Score: 0.58
Precision: 0.72
Recall: 0.78
F1-Score: 0.75

Feature Importance:
      Feature  Coefficient
4        Fare     0.137005
6  Embarked_Q     0.056581
3       Parch    -0.071481
7  Embarked_S    -0.179506
2       SibSp    -0.259174
1         Age    -0.495355
0      Pclass    -0.854869
5    Sex_male    -1.221190
```

In [41]:

```
#Write a Python program to train Logistic Regression with different solvers (liblin

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]
```

```
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


solvers = ['liblinear', 'saga', 'lbfgs']
results = {}

for solver in solvers:
    model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, solver
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    results[solver] = accuracy

    print(f"Solver: {solver}, Accuracy: {accuracy:.2f}")


model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, solver='lb
model.fit(X_train_scaled, y_train)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression (lbfgs
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()

print("\nComparison of Solvers:")
print(pd.DataFrame.from_dict(results, orient='index', columns=['Accuracy']))
```

```
Solver: liblinear, Accuracy: 0.80
Solver: saga, Accuracy: 0.80
Solver: lbfgs, Accuracy: 0.80
```



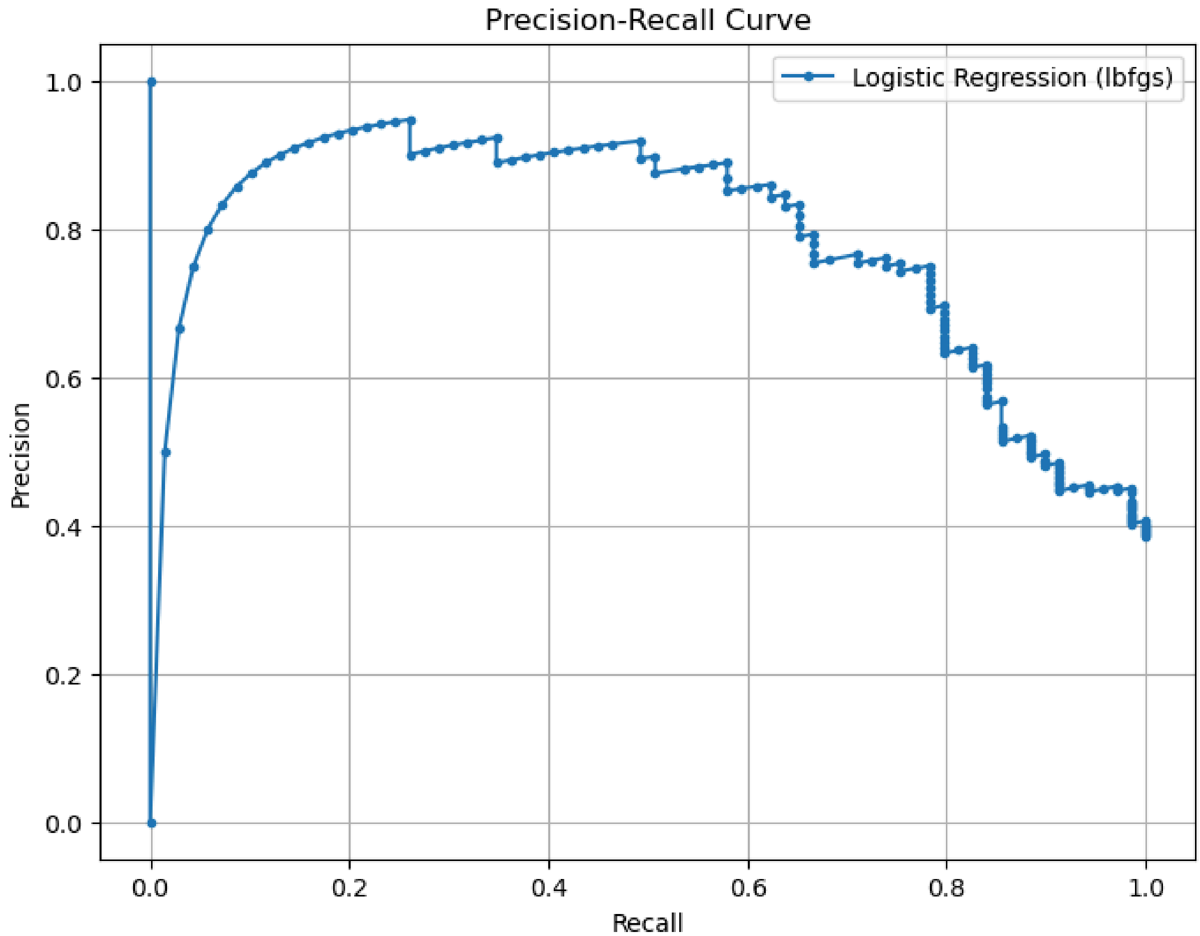

```
Comparison of Solvers:
           Accuracy
liblinear  0.798883
saga       0.798883
lbfgs      0.798883
```

In [48]:

```
#Write a Python program to train Logistic Regression and evaluate its performance u
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

solvers = ['liblinear', 'saga', 'lbfgs']
results = {}

for solver in solvers:
    model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, solver
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    mcc = matthews_corrcoef(y_test, y_pred)
    results[solver] = {'Accuracy': accuracy, 'MCC': mcc}

    print(f"Solver: {solver}, Accuracy: {accuracy:.2f}, MCC: {mcc:.2f}")


model = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, solver='lb
model.fit(X_train_scaled, y_train)
y_pred_proba = model.predict_proba(X_test_scaled)[:, 1]
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba)

plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression (lbfgs
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()

print("\nComparison of Solvers:")
print(pd.DataFrame.from_dict(results, orient='index'))
```

```
Solver: liblinear, Accuracy: 0.80, MCC: 0.58
Solver: saga, Accuracy: 0.80, MCC: 0.58
Solver: lbfgs, Accuracy: 0.80, MCC: 0.58
```

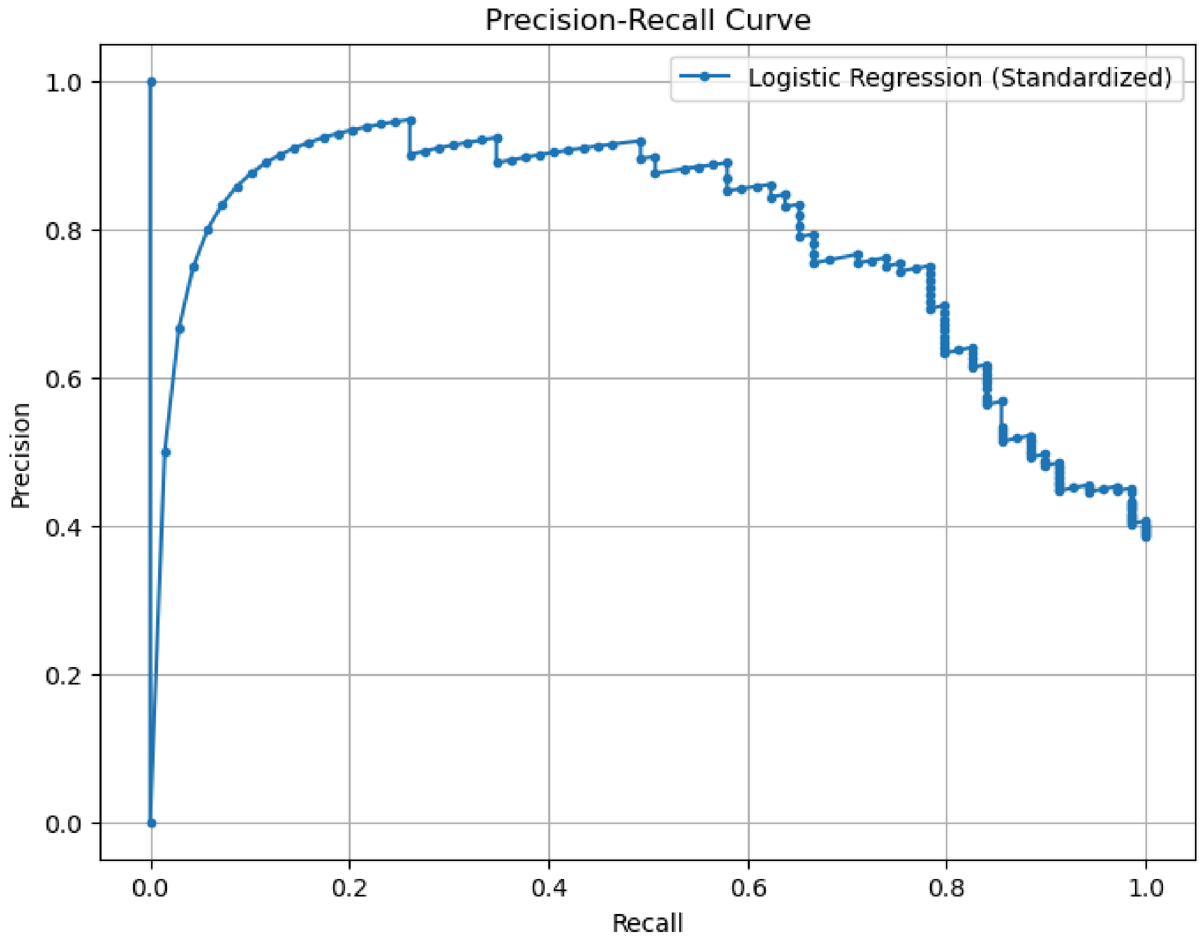



```
Comparison of Solvers:
           Accuracy      MCC
liblinear  0.798883  0.58368
saga       0.798883  0.58368
lbfgs      0.798883  0.58368
```

In [50]:
```
#Write a Python program to train Logistic Regression on both raw and standardized d
#impact of feature scaling

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X  = df.drop(columns=['Survived'])
y  = df['Survived']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


model_raw = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, solver
model_raw.fit(X_train, y_train)
y_pred_raw = model_raw.predict(X_test)
accuracy_raw = accuracy_score(y_test, y_pred_raw)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model_scaled = LogisticRegression(class_weight='balanced', max_iter=500, C=0.5, sol
model_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = model_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)


print(f"Accuracy on Raw Data: {accuracy_raw:.2f}")
print(f"Accuracy on Standardized Data: {accuracy_scaled:.2f}")


y_pred_proba_scaled = model_scaled.predict_proba(X_test_scaled)[:, 1]
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba_scaled

plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression (Stand
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()
```

```
Accuracy on Raw Data: 0.81
Accuracy on Standardized Data: 0.80
```

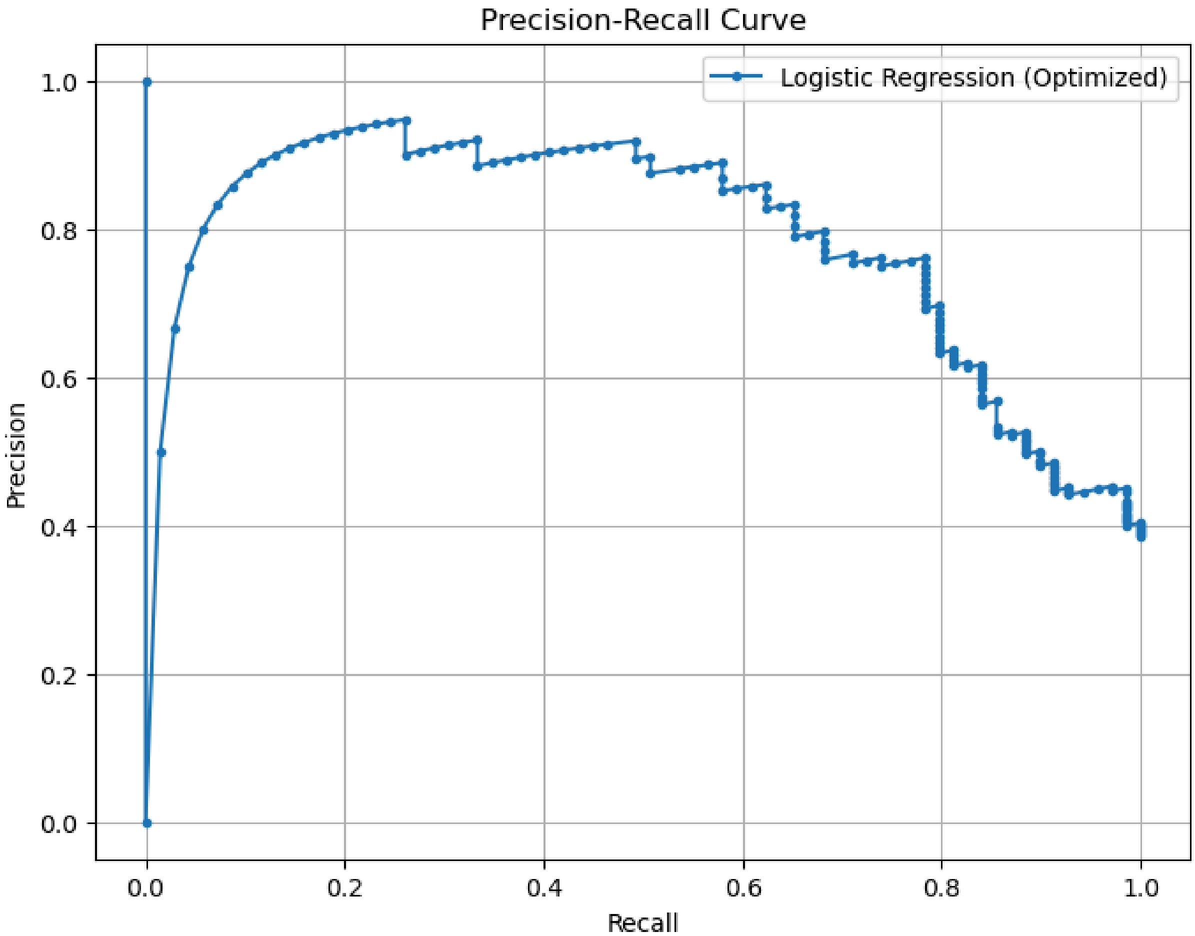


In [52]:

```
#Write a Python program to train Logistic Regression and find the optimal C (regula

df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
param_grid = {'C': np.logspace(-4, 4, 10)}
model = LogisticRegression(class_weight='balanced', max_iter=500, solver='lbfgs')
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)


best_C = grid_search.best_params_['C']
print(f"Optimal C value: {best_C}")


model_optimized = LogisticRegression(class_weight='balanced', max_iter=500, solver=
model_optimized.fit(X_train_scaled, y_train)
y_pred_optimized = model_optimized.predict(X_test_scaled)
accuracy_optimized = accuracy_score(y_test, y_pred_optimized)


print(f"Accuracy with Optimal C ({best_C}): {accuracy_optimized:.2f}")


y_pred_proba_optimized = model_optimized.predict_proba(X_test_scaled)[:, 1]
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba_optimi

plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression (Optim
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()
```

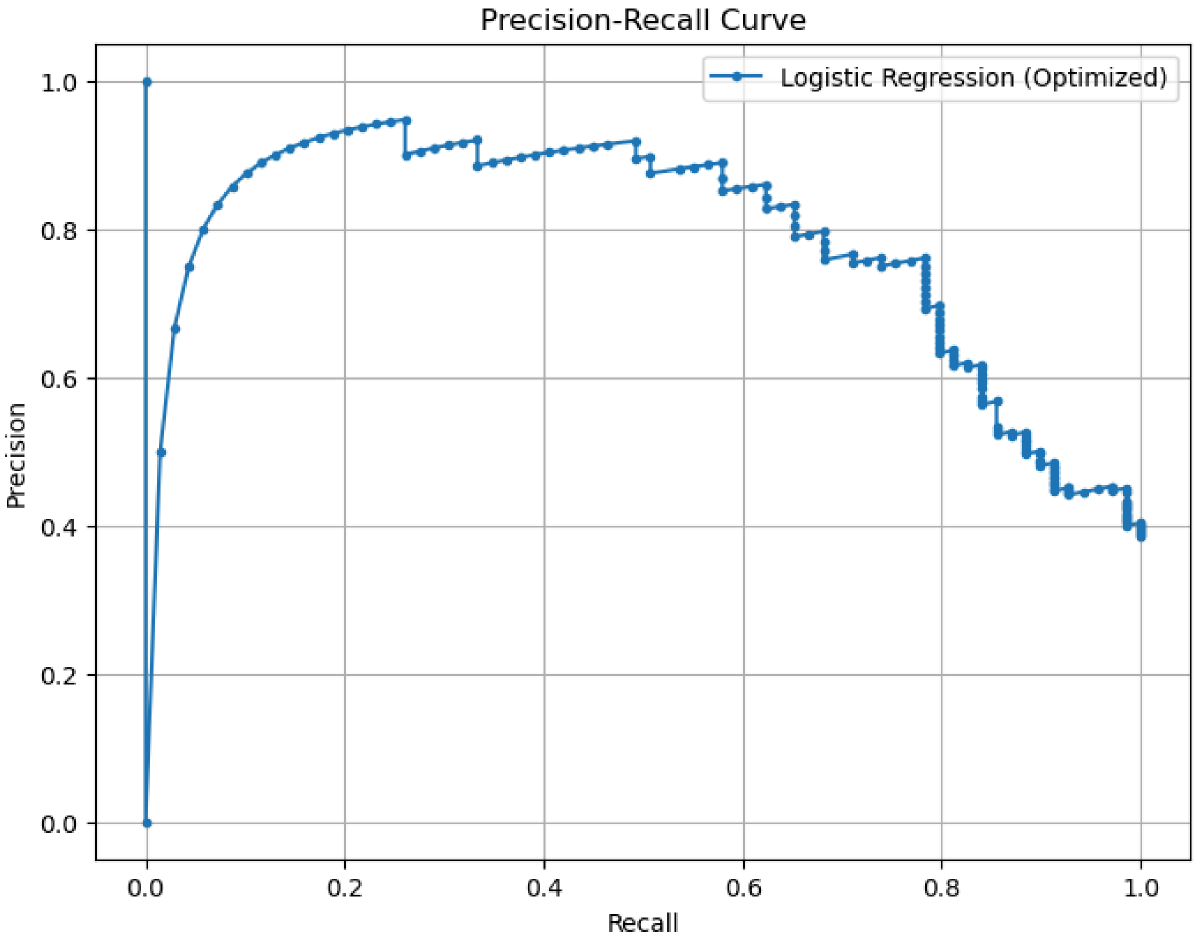
```
Optimal C value: 21.54434690031882
Accuracy with Optimal C (21.54434690031882): 0.80
```



In [54]:
```
#Write a Python program to train Logistic Regression, save the trained model using
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score


df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master


features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
df = df[features + ['Survived']]


df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)


df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)


X = df.drop(columns=['Survived'])
```

```
y = df['Survived']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


param_grid = {'C': np.logspace(-4, 4, 10)}
model = LogisticRegression(class_weight='balanced', max_iter=500, solver='lbfgs')
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)


best_C = grid_search.best_params_['C']
print(f"Optimal C value: {best_C}")


model_optimized = LogisticRegression(class_weight='balanced', max_iter=500, solver=
model_optimized.fit(X_train_scaled, y_train)


joblib.dump(model_optimized, 'logistic_regression_model.pkl')
print("Model saved successfully.")


loaded_model = joblib.load('logistic_regression_model.pkl')
print("Model loaded successfully.")


y_pred_optimized = loaded_model.predict(X_test_scaled)
accuracy_optimized = accuracy_score(y_test, y_pred_optimized)
print(f"Accuracy with Optimal C ({best_C}): {accuracy_optimized:.2f}")


y_pred_proba_optimized = loaded_model.predict_proba(X_test_scaled)[:, 1]
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_pred_proba_optimi

plt.figure(figsize=(8, 6))
plt.plot(recall_vals, precision_vals, marker='.', label='Logistic Regression (Optim
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid()
plt.show()
```

```
Optimal C value: 21.54434690031882
Model saved successfully.
Model loaded successfully.
Accuracy with Optimal C (21.54434690031882): 0.80
```


Precision-Recall Curve
Logistic Regression (Optimized)
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Thank You