1. What is Logistic Regression, and how does it differ from Linear Regression?

- **Logistic Regression** is a supervised learning algorithm used for classification tasks. It predicts the probability that an instance belongs to a particular class.
- **Linear Regression**, on the other hand, is used for regression tasks, where the output is a continuous value.
- **Key Difference:** Logistic regression applies the sigmoid function to transform predictions into probabilities, while linear regression directly predicts numerical values.

2. What is the mathematical equation of Logistic Regression?

The logistic regression model is given by:

$P(y=1|X)=1/1+e^{-(\beta0+\beta1X1+\beta2X2+...+\beta nXn)}$ here,

- $P(y=1|X)$ is the probability of the positive class.
- $\beta0$ is the intercept.
- $\beta1,\beta2,...\beta n$ are the coefficients (weights).
- $X1,X2,...Xn$ are the feature variables.

3. Why do we use the Sigmoid function in Logistic Regression?

The sigmoid function:

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

is used because:

- It maps any real number to a value between 0 and 1, making it suitable for probability estimation.
- It helps interpret predictions as probabilities.
- It provides a smooth gradient, enabling efficient training using gradient descent.

4. What is the cost function of Logistic Regression?

The cost function for logistic regression is the log loss (binary cross-entropy):

$$J(\theta) = -1/m\sum[yilog(h\theta(xi))+(1-yi)log(1-h\theta(xi))]$$

- This function penalizes incorrect predictions more significantly than squared error loss.
- It ensures convexity, allowing gradient descent to find the optimal parameters.

5. What is Regularization in Logistic Regression? Why is it needed?

Regularization prevents overfitting by adding a penalty term to the loss function. It helps in:

- Reducing model complexity.
- Preventing the weights from becoming too large.
- Improving generalization to unseen data.

6. Explain the difference between Lasso, Ridge, and Elastic Net regression.

- **Lasso Regression (L1 Regularization)**: Adds the absolute value of coefficients |λ∑|βjl) to the cost function. It performs feature selection by shrinking some coefficients to zero.
- **Ridge Regression (L2 Regularization)**: Adds the squared value of coefficients (λ∑β^2j) to the cost function. It prevents overfitting without eliminating features.
- **Elastic Net**: A combination of L1 and L2 regularization, controlling both sparsity and shrinkage.

7. When should we use Elastic Net instead of Lasso or Ridge?

- When there are many correlated features, Lasso might randomly pick one and ignore others.
- Ridge keeps all features but doesn't perform feature selection.
- Elastic Net balances both, making it ideal when feature selection and correlated variables are concerns.

8. What is the impact of the regularization parameter (λ) in Logistic Regression?

- Higher λ values → Stronger regularization → Simpler model, less overfitting, but risk of underfitting.

- Lower λ values → Weaker regularization → More complex model, risk of overfitting.

9. What are the key assumptions of Logistic Regression?

- **Linear relationship** between independent variables and the log-odds of the dependent variable.
- **No multicollinearity** among independent variables.
- **Independent observations** (no autocorrelation).
- **Large sample size** for better estimations.

10. What are some alternatives to Logistic Regression for classification tasks?

- Decision Trees
- Random Forest
- Support Vector Machines (SVM)
- Naïve Bayes
- Neural Networks
- Gradient Boosting Methods (XGBoost, LightGBM, CatBoost)

11. What are Classification Evaluation Metrics?

- **Accuracy**: TP+TN / TP+TN+FP+FN
- **Precision**: TP / TP+FP
- **Recall**: TP / TP+FN
- **F1-Score**: Harmonic mean of Precision & Recall
- **ROC-AUC**: Measures model performance across all classification thresholds.
- **Log-Loss**: Measures probability error.

12. How does class imbalance affect Logistic Regression?

- The model may predict the majority class more often.
- Low recall for the minority class.

- Solutions: Class weighting, oversampling, undersampling, or using SMOTE (Synthetic Minority Over-sampling Technique).

13. What is Hyperparameter Tuning in Logistic Regression?

- Adjusting **regularization strength (λ)**, solver choice, and other parameters.
- Methods: Grid Search, Random Search, Bayesian Optimization.

14. What are different solvers in Logistic Regression? Which one should be used?

- **lbfgs**: Default, good for small to medium datasets.
- **saga**: Works with L1, L2, Elastic Net regularization. Good for large datasets.
- **liblinear**: Best for small datasets with L1/L2 regularization.
- **newton-cg, sag**: For large datasets.

15. How is Logistic Regression extended for multiclass classification?

- **One-vs-Rest (OvR)**: Trains a separate binary classifier for each class.
- **Softmax Regression (Multinomial Logistic Regression)**: Generalizes logistic regression for multiple classes.

16. What are the advantages and disadvantages of Logistic Regression?

Advantages:

- Simple, interpretable, fast to train.
- Works well when the relationship between features and target is linear in the log-odds space.

Disadvantages:

- Assumes linear decision boundaries.
- Sensitive to outliers and collinearity.
- Doesn’t handle complex relationships well.

17. What are some use cases of Logistic Regression?

- **Medical Diagnosis** (Disease prediction).
- **Spam Detection** (Email classification).
- **Credit Scoring** (Loan approval).
- **Customer Churn Prediction**.

18. What is the difference between Softmax Regression and Logistic Regression?

- **Logistic Regression**: Used for binary classification.
- **Softmax Regression**: Used for multiclass classification by computing probabilities for all classes.

19. How do we choose between One-vs-Rest (OvR) and Softmax for multiclass classification?

- **OvR** is preferred when classes are imbalanced or the dataset is large.
- **Softmax** is preferred when there are many classes with balanced data.

20. How do we interpret coefficients in Logistic Regression?

- The coefficient βj represents the change in log-odds for a unit increase in Xj.
- **Odds Ratio**: e^βj tells us how much the odds change when Xj increases by 1.

Practical Questions

1) Write a Python program that loads a dataset, splits it into training and testing sets, applies Logistic Regression, and prints the model accuracy

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split into train & test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("Model Accuracy:", accuracy_score(y_test, y_pred))
```

```
Model Accuracy: 1.0
```

2) Write a Python program to apply L1 regularization (Lasso) on a dataset using LogisticRegression(penalty='l1') and print the model accuracy

```
model_l1 = LogisticRegression(penalty='l1', solver='liblinear', C=1.0)
model_l1.fit(X_train, y_train)

y_pred_l1 = model_l1.predict(X_test)
print("L1-Regularized Model Accuracy:", accuracy_score(y_test,
y_pred_l1))
```

```
L1-Regularized Model Accuracy: 1.0
```

3) Write a Python program to train Logistic Regression with L2 regularization (Ridge) using LogisticRegression(penalty='l2'). Print model accuracy and coefficients

```
model_l2 = LogisticRegression(penalty='l2', solver='lbfgs', C=1.0)
model_l2.fit(X_train, y_train)

y_pred_l2 = model_l2.predict(X_test)
print("L2-Regularized Model Accuracy:", accuracy_score(y_test,
y_pred_l2))

# Print coefficients
print("Model Coefficients:", model_l2.coef_)
```

```
L2-Regularized Model Accuracy: 1.0
Model Coefficients: [[-0.39339961  0.96258869 -2.37510705 -0.99874611]
 [ 0.5084024  -0.25486663 -0.21301372 -0.77575531]
 [-0.11500279 -0.70772206  2.58812078  1.77450141]]
```

4) Write a Python program to train Logistic Regression with Elastic Net Regularization (penalty='elasticnet')

```
import warnings
warnings.filterwarnings('ignore')
model_elastic = LogisticRegression(penalty='elasticnet',
solver='saga', l1_ratio=0.5, C=1.0)
model_elastic.fit(X_train, y_train)

y_pred_elastic = model_elastic.predict(X_test)
print("Elastic Net Model Accuracy:", accuracy_score(y_test,
y_pred_elastic))
```

```
Elastic Net Model Accuracy: 1.0
```

5) Write a Python program to train a Logistic Regression model for multiclass classification using multi_class='ovr'

```
model_ovr = LogisticRegression(multi_class='ovr', solver='lbfgs',
C=1.0)
model_ovr.fit(X_train, y_train)

y_pred_ovr = model_ovr.predict(X_test)
print("OvR Logistic Regression Accuracy:", accuracy_score(y_test,
y_pred_ovr))
```

```
OvR Logistic Regression Accuracy: 0.9666666666666667
```

6) Write a Python program to apply GridSearchCV to tune the hyperparameters (C and penalty) of Logistic Regression. Print the best parameters and accuracy

```
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.01, 0.1, 1, 10], 'penalty': ['l1', 'l2']}
grid_search = GridSearchCV(LogisticRegression(solver='liblinear'),
param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Best Parameters: {'C': 10, 'penalty': 'l1'}
Best Accuracy: 0.9583333333333334
```

7) Write a Python program to evaluate Logistic Regression using Stratified K-Fold Cross-Validation. Print the average accuracy

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
model = LogisticRegression()

cv_scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
print("Cross-Validation Accuracy:", cv_scores.mean())
```

```
Cross-Validation Accuracy: 0.9666666666666668
```

8) Write a Python program to load a dataset from a CSV file, apply Logistic Regression, and evaluate its accuracy.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Load dataset
df = sns.load_dataset('tips')

df = df.dropna()  # Drop rows with missing values

label_encoder = LabelEncoder()
y = df.sex
```

```
df.sex = label_encoder.fit_transform(y)
# Define features and target
X = df[["total_bill","tip"]]  # Replace with the actual target column
name
y = df["sex"]

# Encode target variable if categorical
if y.dtype == "object":  # If target contains text labels, encode them
as numbers
    y = LabelEncoder().fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=1)

# Feature Scaling (Optional but recommended for Logistic Regression)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression model
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Make predictions
y_pred = log_reg.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 0.6216216216216216
```

9) Write a Python program to apply RandomizedSearchCV for tuning hyperparameters (C, penalty, solver) in Logistic Regression. Print the best parameters and accuracy

```
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

param_dist = {'C': np.logspace(-4, 4, 20), 'penalty': ['l1', 'l2'],
'solver': ['liblinear', 'saga']}
random_search = RandomizedSearchCV(LogisticRegression(), param_dist,
n_iter=10, cv=5, scoring='accuracy', random_state=42)
random_search.fit(X_train, y_train)

print("Best Parameters:", random_search.best_params_)
print("Best Accuracy:", random_search.best_score_)
```

```
Best Parameters: {'solver': 'liblinear', 'penalty': 'l2', 'C':
0.0006951927961775605}
Best Accuracy: 0.6647058823529411
```

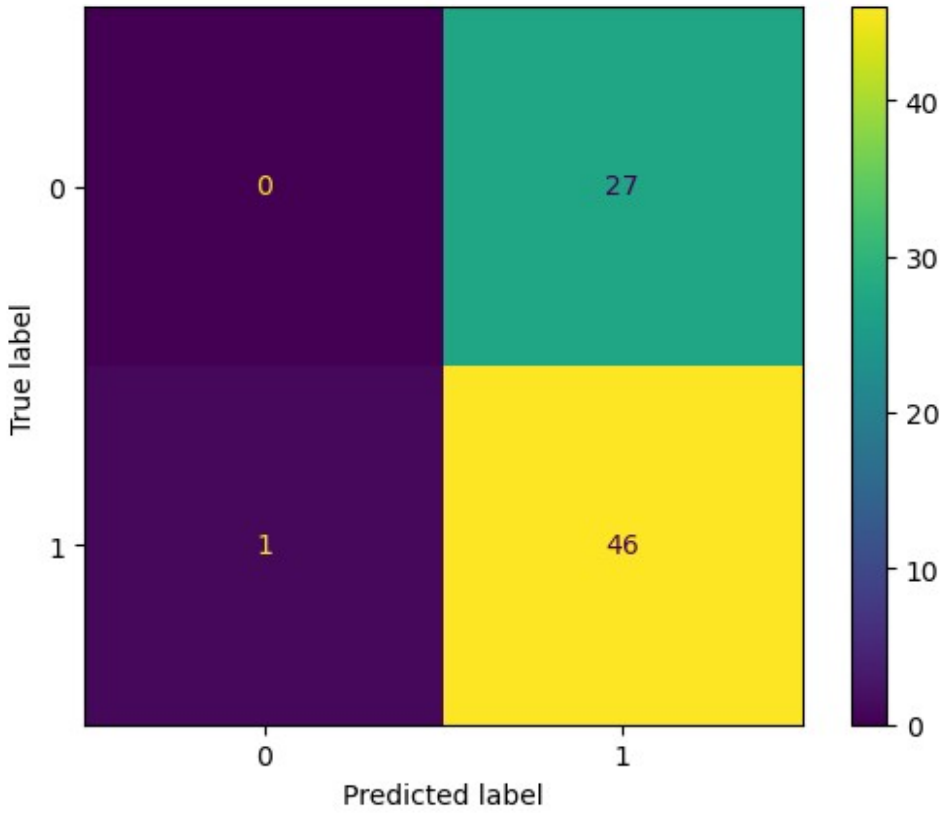
10) Write a Python program to implement One-vs-One (OvO) Multiclass Logistic Regression and print accuracy

```
from sklearn.multiclass import OneVsOneClassifier

ovo_model = OneVsOneClassifier(LogisticRegression())
ovo_model.fit(X_train, y_train)

y_pred_ovo = ovo_model.predict(X_test)
print("OvO Model Accuracy:", accuracy_score(y_test, y_pred_ovo))
```

```
OvO Model Accuracy: 0.6216216216216216
```

11) Write a Python program to train a Logistic Regression model and visualize the confusion matrix for binary classification

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

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

y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```



12) Write a Python program to train a Logistic Regression model and evaluate its performance using Precision, Recall, and F1-Score

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

print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))
```

```
Precision: 0.6301369863013698
Recall: 0.9787234042553191
F1-Score: 0.7666666666666667
```

13) Write a Python program to train a Logistic Regression model on imbalanced data and apply class weights to improve model performance

```
model_weighted = LogisticRegression(class_weight='balanced')
model_weighted.fit(X_train, y_train)

y_pred_weighted = model_weighted.predict(X_test)
```

```
print("Weighted Model Accuracy:", accuracy_score(y_test,
y_pred_weighted))
```

```
Weighted Model Accuracy: 0.527027027027027
```

14) Write a Python program to train Logistic Regression on the Titanic dataset, handle missing values, and evaluate performance

```
df = sns.load_dataset('titanic')
label_encoder = LabelEncoder()
df.sex = label_encoder.fit_transform(df.sex)
df.age = df['age'].fillna(df.age.median())

X = df[['pclass','sex','age','sibsp']]
y = df["survived"]

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

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

y_pred = model.predict(X_test)
print("Titanic Model Accuracy:", accuracy_score(y_test, y_pred))
```

```
Titanic Model Accuracy: 0.7988826815642458
```

15) Write a Python program to apply feature scaling (Standardization) before training a Logistic Regression model. Evaluate its accuracy and compare results with and without scaling

```
from sklearn.preprocessing import StandardScaler

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

model_scaled = LogisticRegression()
model_scaled.fit(X_train_scaled, y_train)

y_pred_scaled = model_scaled.predict(X_test_scaled)
print("Scaled Model Accuracy:", accuracy_score(y_test, y_pred_scaled))
```

```
Scaled Model Accuracy: 0.7988826815642458
```

16) Write a Python program to train Logistic Regression and evaluate its performance using ROC-AUC score

```
from sklearn.metrics import roc_auc_score

y_prob = model.predict_proba(X_test)[:, 1]
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

```
ROC-AUC Score: 0.8775418275418276
```

17) Write a Python program to train Logistic Regression using a custom learning rate (C=0.5) and evaluate accuracy

```
model_custom_C = LogisticRegression(C=0.5)
model_custom_C.fit(X_train, y_train)

y_pred_custom = model_custom_C.predict(X_test)
print("Model Accuracy (C=0.5):", accuracy_score(y_test,
y_pred_custom))
```

```
Model Accuracy (C=0.5): 0.8044692737430168
```

18) Write a Python program to train Logistic Regression and identify important features based on model

coefficients

```
import numpy as np

coefs = model.coef_[0]
features = np.array(data.feature_names)

for feature, coef in sorted(zip(features, coefs), key=lambda x:
abs(x[1]), reverse=True):
    print(f"{feature}: {coef}")
```

```
sepal width (cm): -2.5822331289302056
sepal length (cm): -1.0355191822385603
petal width (cm): -0.32061952669301963
petal length (cm): -0.03136288547896346
```

19) Write a Python program to train Logistic Regression and evaluate its performance using Cohen’s Kappa

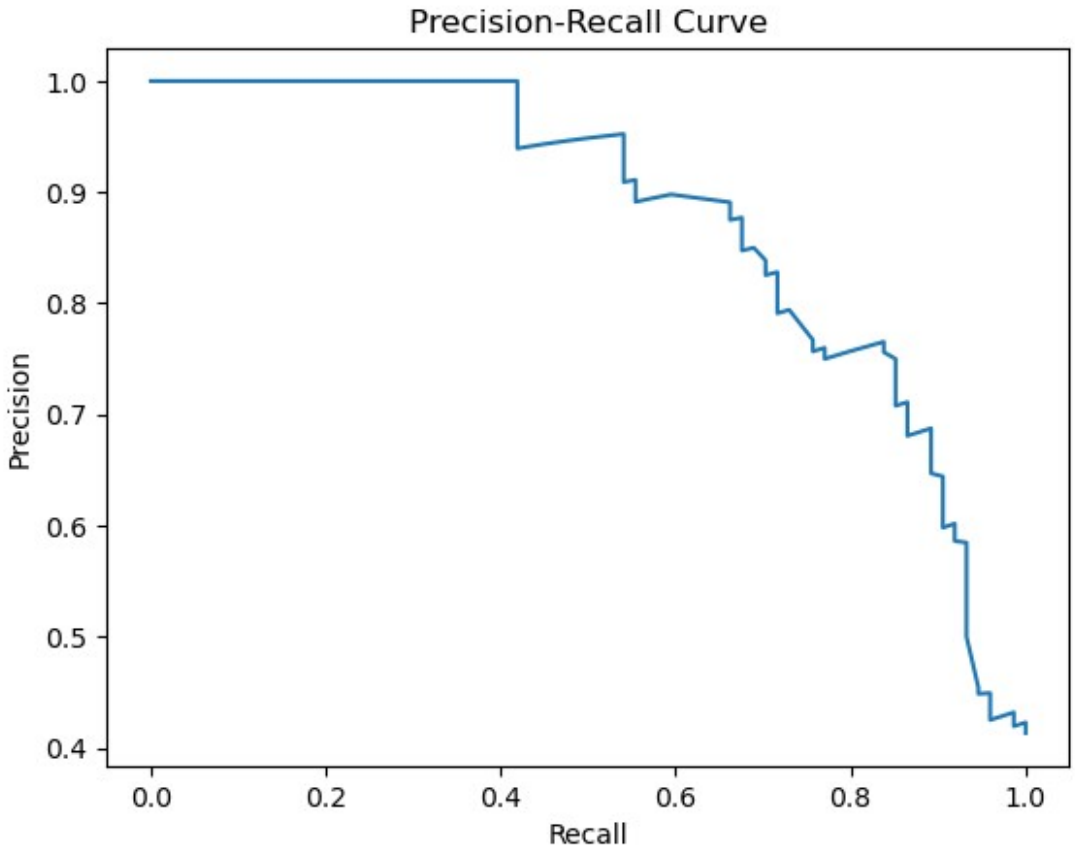
Score

```
from sklearn.metrics import cohen_kappa_score

print("Cohen’s Kappa Score:", cohen_kappa_score(y_test, y_pred))
```

```
Cohen’s Kappa Score: 0.5853281853281853
```

20) Write a Python program to train Logistic Regression and visualize the Precision-Recall Curve for binary classification

```
from sklearn.metrics import precision_recall_curve

precision, recall, _ = precision_recall_curve(y_test, y_prob)
plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.show()
```



21) Write a Python program to train Logistic Regression with different solvers (liblinear, saga, lbfgs) and compare their accuracy

```
solvers = ["liblinear", "saga", "lbfgs"]

for solver in solvers:
    model_solver = LogisticRegression(solver=solver)
    model_solver.fit(X_train, y_train)
    y_pred_solver = model_solver.predict(X_test)
    print(f"Accuracy with {solver}: {accuracy_score(y_test,
y_pred_solver)}")
```

```
Accuracy with liblinear: 0.7988826815642458
Accuracy with saga: 0.7541899441340782
Accuracy with lbfgs: 0.7988826815642458
```

22) Write a Python program to train Logistic Regression and evaluate its performance using Matthews Correlation Coefficient (MCC)

```
from sklearn.metrics import matthews_corrcoef

print("MCC Score:", matthews_corrcoef(y_test, y_pred))
```

```
MCC Score: 0.5853281853281853
```

23) Write a Python program to train Logistic Regression on both raw and standardized data. Compare their accuracy to see the impact of feature scaling

```
model_raw = LogisticRegression()
model_raw.fit(X_train, y_train)
acc_raw = accuracy_score(y_test, model_raw.predict(X_test))

model_scaled = LogisticRegression()
model_scaled.fit(X_train_scaled, y_train)
acc_scaled = accuracy_score(y_test,
model_scaled.predict(X_test_scaled))

print("Raw Data Accuracy:", acc_raw)
print("Standardized Data Accuracy:", acc_scaled)
```

```
Raw Data Accuracy: 0.7988826815642458
Standardized Data Accuracy: 0.7988826815642458
```

24) Write a Python program to train Logistic Regression and find the optimal C (regularization strength) using cross-validation

```
from sklearn.model_selection import cross_val_score

C_values = [0.01, 0.1, 1, 10, 100]
best_C = 0
best_score = 0

for C in C_values:
    model_C = LogisticRegression(C=C)
    scores = cross_val_score(model_C, X, y, cv=5, scoring='accuracy')
    mean_score = scores.mean()
    print(f"C={C}, Accuracy={mean_score}")

    if mean_score > best_score:
        best_score = mean_score
        best_C = C

print("Optimal C:", best_C)
```

```
C=0.01, Accuracy=0.7564810746343607
C=0.1, Accuracy=0.792379637185362
C=1, Accuracy=0.786730274307953
C=10, Accuracy=0.786730274307953
C=100, Accuracy=0.786730274307953
Optimal C: 0.1
```

25) Write a Python program to train Logistic Regression, save the trained model using joblib, and load it again to make predictions.

```
import joblib

# Train and Save Model
joblib.dump(model, "logistic_model.pkl")

# Load Model
loaded_model = joblib.load("logistic_model.pkl")

# Make Predictions
y_pred_loaded = loaded_model.predict(X_test)
print("Loaded Model Accuracy:", accuracy_score(y_test, y_pred_loaded))
```

```
Loaded Model Accuracy: 0.7988826815642458
```