```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import GridSearchCV
from statsmodels.stats.outliers_influence import variance_inflation_factor
data=pd.read_csv("/content/heart_statlog_cleveland_hungary_final.csv")
data.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1
4	54	1	3	150	195	0	0	122	0	0.0	1	0

The objective of the analysis is to predict the likelihood of a heart condition based on various health indicators. The dataset provided includes features such as age, sex, chest pain type, resting blood pressure, cholesterol levels, fasting blood sugar, resting electrocardiogram, maximum heart rate, exercise-induced angina, oldpeak, and ST slope. The target variable is the presence or absence of a heart condition, which is represented by a binary value (0 or 1). he goal is to build a logistic regression model that can accurately predict the likelihood of a heart condition given these health indicators. bold text

```
data.head()
```

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	target
0	40	1	2	140	289	0	0	172	0	0.0	1	0
1	49	0	3	160	180	0	0	156	0	1.0	2	1
2	37	1	2	130	283	0	1	98	0	0.0	1	0
3	48	0	4	138	214	0	0	108	1	1.5	2	1
4	54	1	3	150	195	0	0	122	0	0.0	1	0

```
print(data.isnull().sum())
```

```
age                    0
sex                    0
chest pain type        0
resting bp s           0
cholesterol            0
fasting blood sugar    0
resting ecg            0
max heart rate         0
exercise angina        0
oldpeak                0
ST slope               0
```

```
target                 0
dtype: int64
```

```
vif = pd.DataFrame()
vif['feature'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
```

```
               feature       VIF
0                  sex  2.899858
1          cholesterol  2.776460
2  fasting blood sugar  1.278526
3          resting ecg  1.695205
4      exercise angina  1.940553
5              oldpeak  2.024018
```

The features "age," "chest pain type," "resting bp s," and "max heart rate" have high Variance Inflation Factor (VIF) values, indicating multicollinearity issues. Multicollinearity can lead to unstable estimates of the coefficients and reduce the model's interpretability. we have dropped the variables with high VIF values to improve the model's performance and stability.

```
columns_to_drop = ['age', 'chest pain type', 'resting bp s', 'max heart rate', 'ST slope']
data = data.drop(columns=columns_to_drop)

print(data.head())
```

```
   sex  cholesterol  fasting blood sugar  resting ecg  exercise angina  \
0    1          289                    0            0                0
1    0          180                    0            0                0
2    1          283                    0            1                0
3    0          214                    0            0                1
4    1          195                    0            0                0

   oldpeak  target
0      0.0       0
1      1.0       1
2      0.0       0
3      1.5       1
4      0.0       0
```

This code will drop the columns 'age', 'chest pain type', 'resting bp s', 'max heart rate', and 'ST slope' from the dataset

```
print(data.describe())
```

```
               sex  cholesterol  fasting blood sugar  resting ecg  \
count  1190.000000  1190.000000          1190.000000  1190.000000
mean      0.763866   210.363866             0.213445     0.698319
std       0.424884   101.420489             0.409912     0.870359
min       0.000000     0.000000             0.000000     0.000000
25%       1.000000   188.000000             0.000000     0.000000
50%       1.000000   229.000000             0.000000     0.000000
75%       1.000000   269.750000             0.000000     2.000000
max       1.000000   603.000000             1.000000     2.000000

       exercise angina      oldpeak       target
count      1190.000000  1190.000000  1190.000000
mean          0.387395     0.922773     0.528571
std           0.487360     1.086337     0.499393
```

```
min          0.000000    -2.600000     0.000000
25%          0.000000     0.000000     0.000000
50%          0.000000     0.600000     1.000000
75%          1.000000     1.600000     1.000000
max          1.000000     6.200000     1.000000
```

```
X = data.drop('target', axis=1)
y = data['target']
y.head()
```

```
0    0
1    1
2    0
3    1
4    0
Name: target, dtype: int64
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10],
    'penalty': ['l1', 'l2'],
    'max_iter': [100, 200, 300]
}

model = LogisticRegression(solver='liblinear')

grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

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

```
Best Parameters: {'C': 0.1, 'max_iter': 100, 'penalty': 'l2'}
Best Score: 0.7888509231193165
```

Regularization Strength : A lower value of C (0.1) suggests that the model prioritizes simplicity and generalizability over fitting the training data perfectly. This helps prevent overfitting. **Maximum Number of Iterations** : The model converged within 100 iterations, indicating that the optimization process was successful within a reasonable number of iterations. **Penalty ('l2')**: The 'l2' penalty implies that the model uses L2 regularization, which penalizes large coefficients to prevent overfitting and improve the model's generalization ability. **Best Score (0.7888509231193165)**: The best score achieved by the model on the test set is approximately 0.789, indicating that the model's accuracy in predicting heart disease based on the provided dataset is around 78.9%.

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

# Perform cross-validation with the best model
best_model = grid_search.best_estimator_
scores = cross_val_score(best_model, X, y, cv=5, scoring='accuracy')
print("Cross-Validation Scores:", scores)
print("Mean Cross-Validation Score:", scores.mean())
```

```
Best Parameters: {'C': 0.1, 'max_iter': 100, 'penalty': 'l2'}
Best Score: 0.7888509231193165
```

```
Cross-Validation Scores: [0.80672269 0.81092437 0.78991597 0.72689076 0.73109244]
Mean Cross-Validation Score: 0.773109243697479
```

1.The best score achieved by the logistic regression model on the test set is approximately 0.7888509231193165, which represents an accuracy of around 78.9% in predicting the presence of heart disease. **2.Cross-Validation Scores:** The cross-validation scores, which provide a more robust estimate of the model's performance, are: [0.80672269, 0.81092437, 0.78991597, 0.72689076, 0.73109244] The mean cross-validation score is approximately 0.773109243697479, which is slightly lower than the test set score. **3.Model Performance:** The combination of the test set score and the cross-validation scores suggests that the logistic regression model with the specified hyperparameters (C=0.1, max_iter=100, penalty='l2') performs reasonably well in predicting the presence of heart disease based on the data. **4.Generalization Ability:** The relatively high cross-validation scores, with a mean of around 0.773, indicate that the model has a good generalization ability and is likely to perform well on unseen data, not just the test set.

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

```
▾          LogisticRegression
LogisticRegression(solver='liblinear')
```

```
y_pred = model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
print(f"Precision: {precision_score(y_test, y_pred):.2f}")
print(f"Recall: {recall_score(y_test, y_pred):.2f}")
print(f"F1-score: {f1_score(y_test, y_pred):.2f}")
```

```
Accuracy: 0.80
Precision: 0.84
Recall: 0.79
F1-score: 0.81
```

Accuracy (0.80): The accuracy of 80% indicates that the model correctly predicted 80% of the instances in the data. **Precision (0.84)**: The precision of 84% suggests that when the model predicts an individual has heart disease, it is correct 84% of the time. It is a measure of the model's ability to avoid false positives. **Recall (0.79)**: The recall of 79% indicates that the model correctly identifies 79% of the individuals who actually have heart disease. It is a measure of the model's ability to capture all positive instances. **F1-score (0.81)**: The F1-score of 81% is the harmonic mean of precision and recall. It provides a balance between precision and recall, giving a single metric to evaluate the model's performance.