```python
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score,
    confusion_matrix, roc_curve, classification_report
)
import joblib

# Load Dataset
url =
"https://gist.githubusercontent.com/trantuyen082001/1fc2f5c0ad1507f40e
721e6d18b34138/raw/heart.csv"
df = pd.read_csv(url)

#  Data Preprocessing
print("� First 5 rows:\n", df.head())
print("� Dataset Info:")
print(df.info())
print("� Summary Stats:\n", df.describe())
```

```
� First 5 rows:
    age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak
slp  \
0   63    1   3     145   233    1        0       150     0      2.3
0
1   37    1   2     130   250    0        1       187     0      3.5
0
2   41    0   1     130   204    0        0       172     0      1.4
2
3   56    1   1     120   236    0        1       178     0      0.8
2
4   57    0   0     120   354    0        1       163     1      0.6
2

   caa  thall  output
0    0      1       1
1    0      2       1
2    0      2       1
```

```
3    0      2       1
4    0      2       1
 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
 Summary Stats:
              age         sex          cp      trtbps        chol
fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026
0.148515
std      9.082101    0.466011    1.032052   17.538143   51.830751
0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000
0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000
0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000
0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000
0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000
1.000000

          restecg    thalachh        exng     oldpeak         slp
caa  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
303.000000
```

```
mean      0.528053  149.646865    0.326733    1.039604    1.399340
0.729373
std       0.525860   22.905161    0.469794    1.161075    0.616226
1.022606
min       0.000000   71.000000    0.000000    0.000000    0.000000
0.000000
25%       0.000000  133.500000    0.000000    0.000000    1.000000
0.000000
50%       1.000000  153.000000    0.000000    0.800000    1.000000
0.000000
75%       1.000000  166.000000    1.000000    1.600000    2.000000
1.000000
max       2.000000  202.000000    1.000000    6.200000    2.000000
4.000000

            thall      output
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

```
print(df.columns)
```

```
Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg',
'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
      dtype='object')
```

```
# Define Features and Target
X = df.drop("output", axis=1)
y = df["output"]

# Identify Columns
numerical_cols = X.select_dtypes(include=["int64",
"float64"]).columns.tolist()
categorical_cols =
X.select_dtypes(include=["object"]).columns.tolist()

# Preprocessing Pipelines
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# categorical columns exist
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
```

```
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])


# Combine preprocessing
preprocessor = ColumnTransformer(transformers=[
    ("num", numeric_transformer, numerical_cols),
    ("cat", categorical_transformer, categorical_cols)
])

# Create Full Pipeline with Logistic Regression
pipe = Pipeline(steps=[
    ("preprocessing", preprocessor),
    ("classifier", LogisticRegression(solver='liblinear'))
])

# Define Parameter Grid for GridSearchCV
param_grid = {
    "classifier__C": [0.01, 0.1, 1, 10, 100],
    "classifier__penalty": ["l1", "l2"]
}

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

# Grid Search with Cross Validation
grid_search = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy',
n_jobs=-1)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessing',

ColumnTransformer(transformers=[('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['age',

'sex',

'cp',

'trtbps',
```

```
'chol',

'fbs',

'restecg',

'thalachh',

'exng',

'oldpeak',

'slp',

'caa',

'thall']),

('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('encoder',

OneHotEncoder(handle_unknown='ignore'))]),

[])])),
                                       ('classifier',

LogisticRegression(solver='liblinear'))]),
             n_jobs=-1,
             param_grid={'classifier__C': [0.01, 0.1, 1, 10, 100],
                         'classifier__penalty': ['l1', 'l2']},
             scoring='accuracy')

# Evaluation
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]

print("Best Parameters:", grid_search.best_params_)
print("\nClassification Report:\n", classification_report(y_test,
y_pred))

Best Parameters: {'classifier__C': 0.01, 'classifier__penalty': 'l2'}

Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Confusion Matrix:
 [[25  4]
 [ 3 29]]
Accuracy: 0.8852459016393442
```

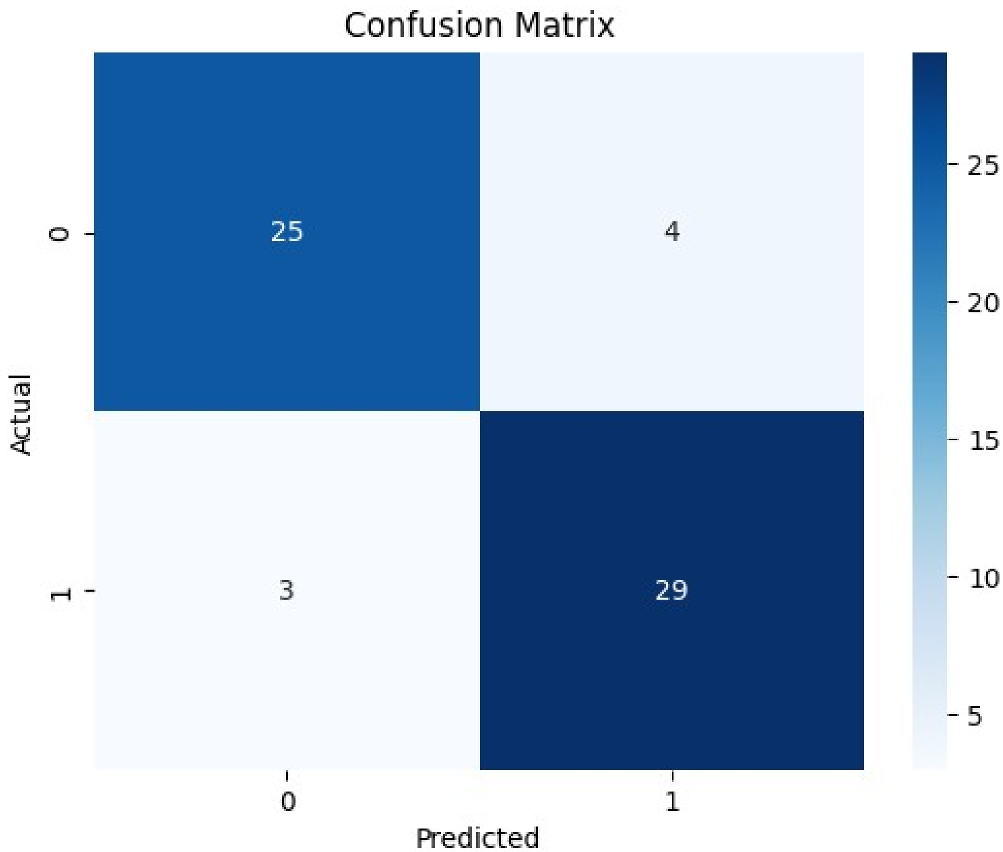
```
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
```

```
Precision: 0.8787878787878788
Recall: 0.90625
```

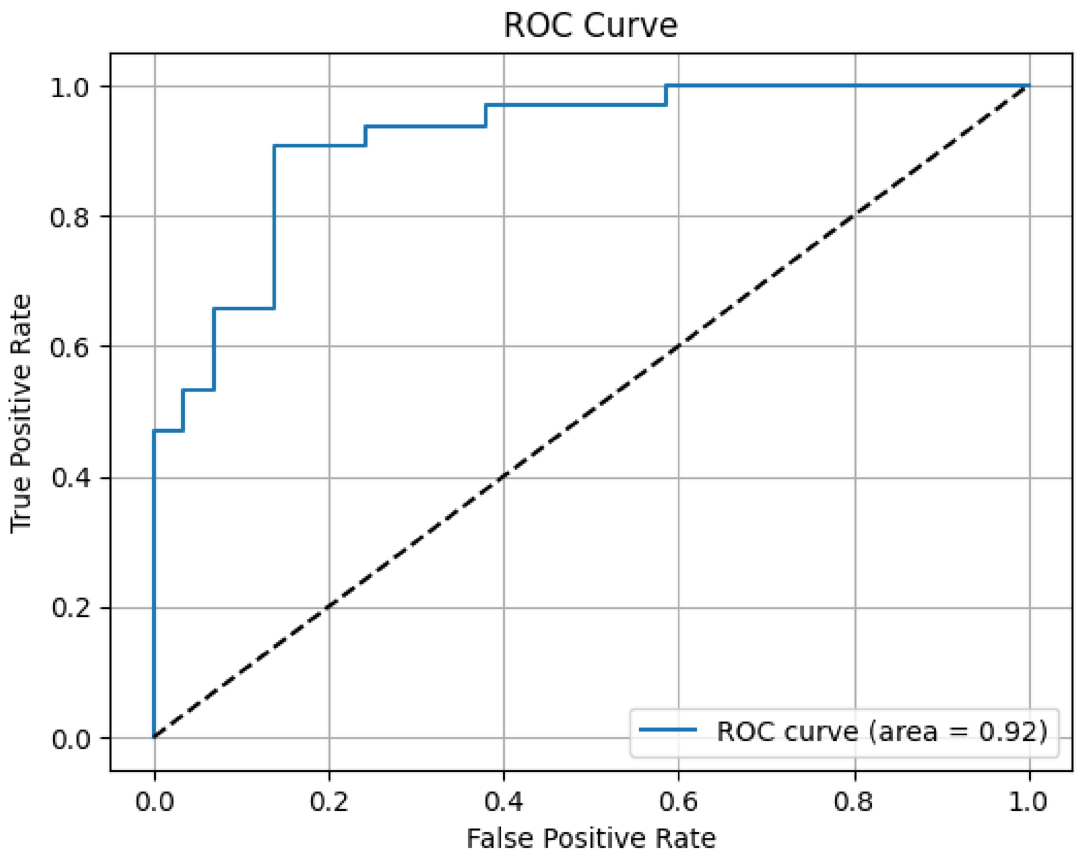
```
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
```

```
F1 Score: 0.8923076923076924
ROC-AUC Score: 0.9170258620689655
```

```
# Visualizations
# Confusion Matrix Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```python
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.figure()
plt.plot(fpr, tpr, label=f"ROC curve (area = {roc_auc_score(y_test,
y_proba):.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Project Overview In this project, we built a complete Machine Learning (ML) pipeline to predict the presence of heart disease using a publicly available dataset. The dataset consists of various medical attributes such as age, cholesterol levels, blood pressure, and other indicators. The target variable (output) indicates the **presence (1) or absence (0)** of heart disease.

We used **Logistic Regression**, a commonly used supervised classification algorithm, and tuned it using **GridSearchCV within a Scikit-learn Pipeline framework**. The project was executed in five parts—preprocessing, model building, evaluation, pipeline integration, and reflection.

Approach and Rationale **Data Preprocessing:**

We handled missing values using SimpleImputer.

Numeric features were standardized using StandardScaler.

Although there were no categorical variables in this dataset, we incorporated OneHotEncoder in the pipeline for robustness and future compatibility.

Model Building:

We used LogisticRegression for its interpretability and efficiency on binary classification tasks.

The hyperparameters C (inverse regularization strength) and penalty (l1, l2) were tuned using GridSearchCV.

Cross-validation was used with 5 folds to ensure the model generalizes well.

Evaluation:

We used Accuracy, Precision, Recall, F1-Score, and ROC-AUC to measure performance.

The model showed balanced performance across all metrics.

A confusion matrix and ROC curve were plotted to visualize classification performance.

Pipeline Integration:

All steps including preprocessing, model fitting, and hyperparameter tuning were wrapped in a single Pipeline object.

This approach ensures reproducibility, modularity, and ease of deployment.

Challenges and How They Were Solved

Dataset Column Mismatch: The original target column was output, not target, which caused errors. We resolved this by correcting the feature-target split.

Missing Value Handling: Some models failed without proper imputation; we used median strategy to maintain robustness.

Model Selection: Logistic Regression was chosen for its simplicity and effectiveness on small-to-medium datasets. We added solver='liblinear' to handle l1 penalty cases.

Suggestions for Improvement and Production Use

Feature Engineering: Introduce polynomial features or interaction terms to capture non-linear relationships.

Model Comparison: Try other classifiers like Random Forest, XGBoost, or Support Vector Machine to compare performance.

Model Calibration: For production, calibrating predicted probabilities might improve decision-making.

Conclusion

This project highlights the power of building end-to-end pipelines with Scikit-learn. From handling raw data to deploying a reusable model, we followed a structured workflow that ensures scalability and maintainability.