Using logistic regression to predict whether a patient has heart disease

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
```

- These lines import the necessary libraries and functions for data preprocessing, model training, and evaluation. They include tools for handling missing values, scaling data, encoding categorical variables, creating pipelines, splitting datasets, training logistic regression models, and plotting results.

```
# Load the dataset
data = pd.read_csv('heart.csv')
```

```
# Handle missing values (if any)
data = data.dropna()
```

- Load the heart disease dataset from a CSV file into a pandas DataFrame and removes any rows with missing values to ensure the dataset is clean.

```python
# Separate features and target variable
X = data.drop('target', axis=1)
y = data['target']
```

- Separate the dataset into features (`X`) and the target variable (`y`). The target variable is the 'target' column, which indicates the presence of heart disease.

```python
# Encode categorical variables
categorical_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']
numerical_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

```python
# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

```python
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

```python
# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

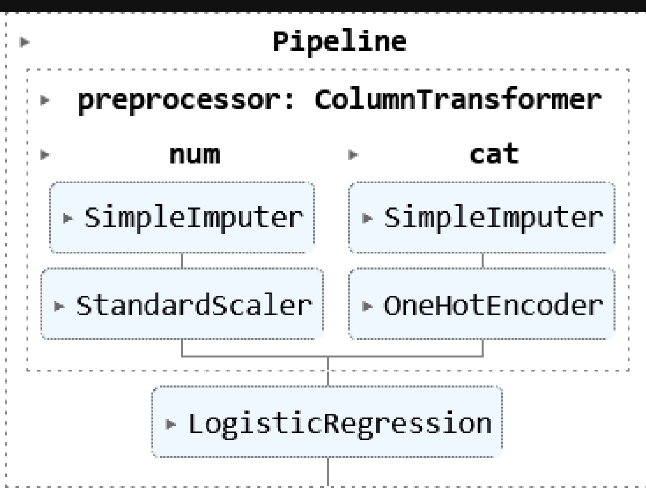
- Define how to preprocess numerical and categorical features. Numerical features are imputed with the mean and scaled, while categorical features are imputed with the most frequent value and one-hot encoded. The `ColumnTransformer` combines these preprocessing steps.

```python
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Split the dataset into training and testing sets, with 80% of the data used for training and 20% for testing. The `random_state=42` ensures reproducibility of the split.

```python
# Create the logistic regression model with regularization
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='liblinear', penalty='l2'))
])
```

```python
# Train the model
model.fit(X_train, y_train)
```



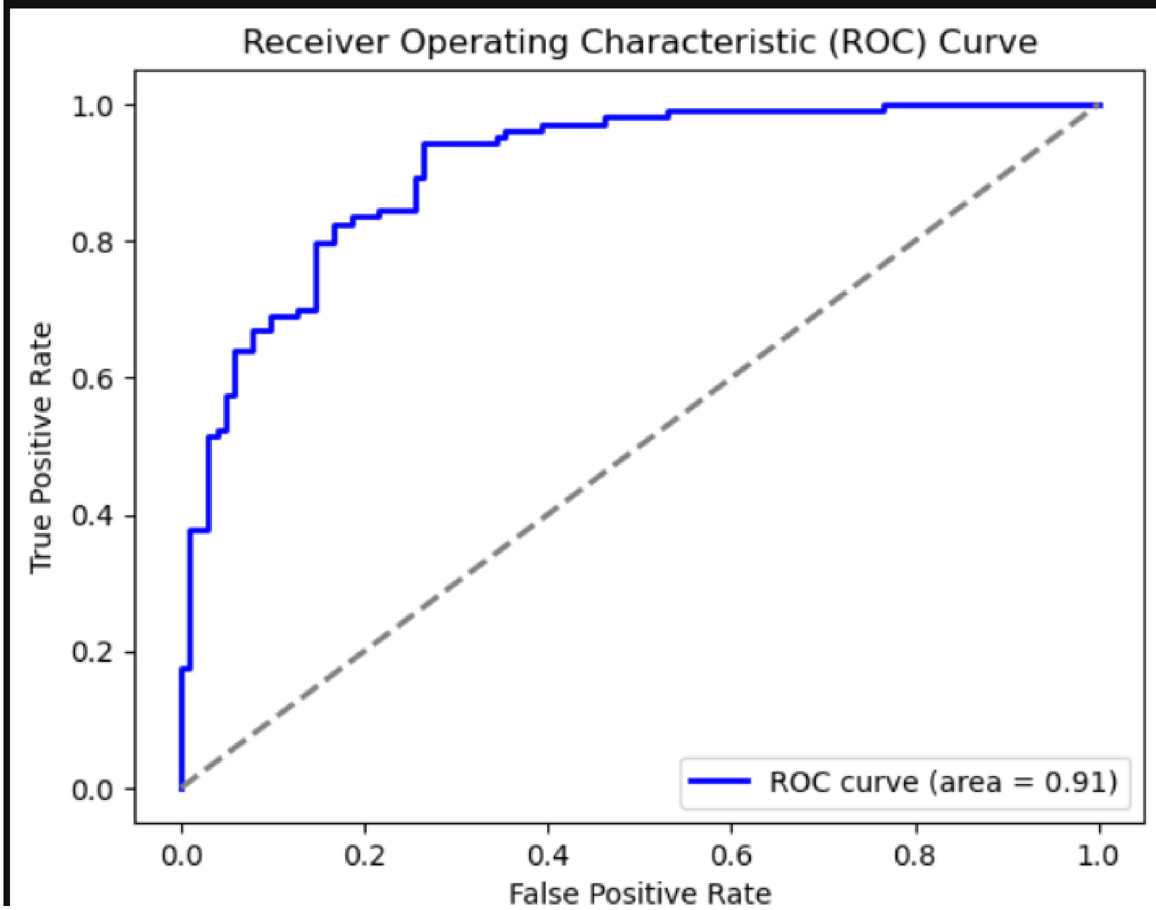
- Create a pipeline that includes preprocessing and logistic regression model training with L2 regularization. It then trains the model using the training data.

```python
# Predict on the test set
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

```python
# Evaluate the model
accuracy = accuracy_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)
roc_auc = roc_auc_score(y_test, y_pred_proba)
```

- Make predictions on the test set and evaluates the model's performance using various metrics, including accuracy, precision, recall, F1 score, and ROC AUC score.

```
# Plot the ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



- Plot the ROC curve, which shows the trade-off between the true positive rate and false positive rate. It also calculates and displays the ROC AUC score.

```python
# Interpret the coefficients
coefficients = model.named_steps['classifier'].coef_[0]
features = numerical_features + list(model.named_steps['preprocessor'].named_transformers_['cat'].named_steps['onehot'].get_feature_names_out(categorical_features))
feature_importance = pd.DataFrame({'Feature': features, 'Coefficient': coefficients})

# Calculate odds ratios
feature_importance['Odds Ratio'] = np.exp(feature_importance['Coefficient'])

# Display feature importance
print("Feature Importance with Odds Ratios")
print(feature_importance)
```

```
Feature Importance with Odds Ratios
      Feature  Coefficient  Odds Ratio
0         age     0.255666    1.291322
1    trestbps    -0.410215    0.663508
2        chol    -0.412019    0.662312
3     thalach     0.531254    1.701063
4     oldpeak    -0.594951    0.551590
5       sex_0     0.876477    2.402421
6       sex_1    -0.836215    0.433348
7        cp_0    -1.279474    0.278184
8        cp_1    -0.098985    0.905756
9        cp_2     0.413950    1.512781
10       cp_3     1.004772    2.731284
11      fbs_0    -0.124305    0.883110
12      fbs_1     0.164567    1.178883
13  restecg_0    -0.000090    0.999910
14  restecg_1     0.209980    1.233653
15  restecg_2    -0.169628    0.843979
16    exang_0     0.352167    1.422145
17    exang_1    -0.311904    0.732052
18    slope_0     0.132314    1.141466
19    slope_1    -0.646984    0.523623
20    slope_2     0.554932    1.741823
21       ca_0     1.560000    4.758819
22       ca_1    -0.591316    0.553598
23       ca_2    -1.561808    0.209757
24       ca_3    -0.786521    0.455426
25       ca_4     1.419907    4.136737
26     thal_0    -0.509038    0.601073
27     thal_1     0.836448    2.308153
28     thal_2     0.533541    1.704959
29     thal_3    -0.820688    0.440129
```

- Extract and interprets the model coefficients, calculating odds ratios to understand the impact of each feature on the likelihood of heart disease. It then prints a DataFrame showing feature importance and odds ratios.

```python
# Display evaluation metrics
evaluation_metrics = {
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1,
    'ROC AUC': roc_auc
}
```

```python
print("\nEvaluation Metrics")
for metric, value in evaluation_metrics.items():
    print(f"{metric}: {value:.2f}")
```

```
Evaluation Metrics
Accuracy: 0.82
Precision: 0.78
Recall: 0.89
F1 Score: 0.83
ROC AUC: 0.91
```

- Create a dictionary of evaluation metrics and prints each metric's name and value, summarizing the model's performance.