Heart Disease Prediction using Logistic Regression

Importing the Required Libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.decomposition import PCA
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report, roc_curve, roc_auc_score
```

Exploratory Data Analysis

Data Description

↳ *1 cell hidden*

EDA

```
columns = ['age', 'sex', 'cp', 'trestbps', 'chol',  'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope','ca', 'thal', 'num']

raw_data = pd.read_csv('/content/drive/MyDrive/MACHINE LEARNING MAY/DAY2_HEART_DISEASE_PREDICTION/processed.cleveland.data', header=None,
df = raw_data.copy()
df.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0

Next steps: Generate code with `df` | View recommended plots

```
df.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    float64
 1   sex       303 non-null    float64
 2   cp        303 non-null    float64
 3   trestbps  303 non-null    float64
 4   chol      303 non-null    float64
 5   fbs       303 non-null    float64
 6   restecg   303 non-null    float64
 7   thalach   303 non-null    float64
 8   exang     303 non-null    float64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    float64
 11  ca        303 non-null    object 
 12  thal      303 non-null    object 
```

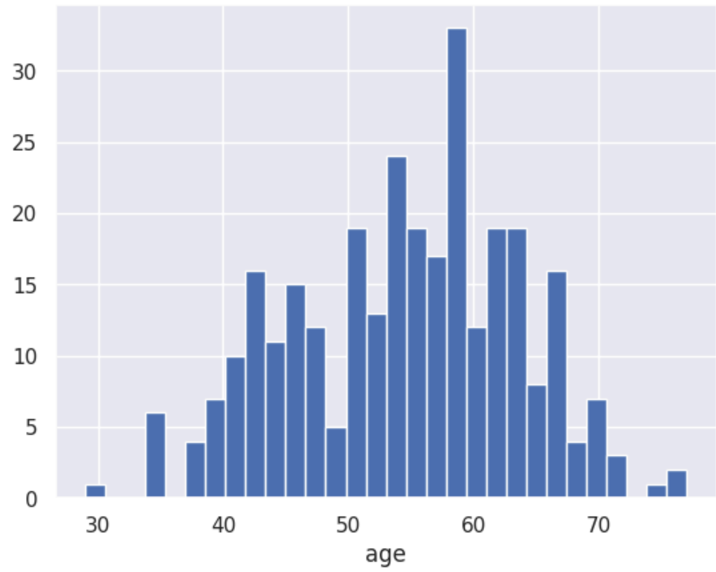
```
 13  num       303 non-null    int64
dtypes: float64(11), int64(1), object(2)
memory usage: 33.3+ KB
```

```
for column in df:
  print(df[column].value_counts())
  print('\n')
```

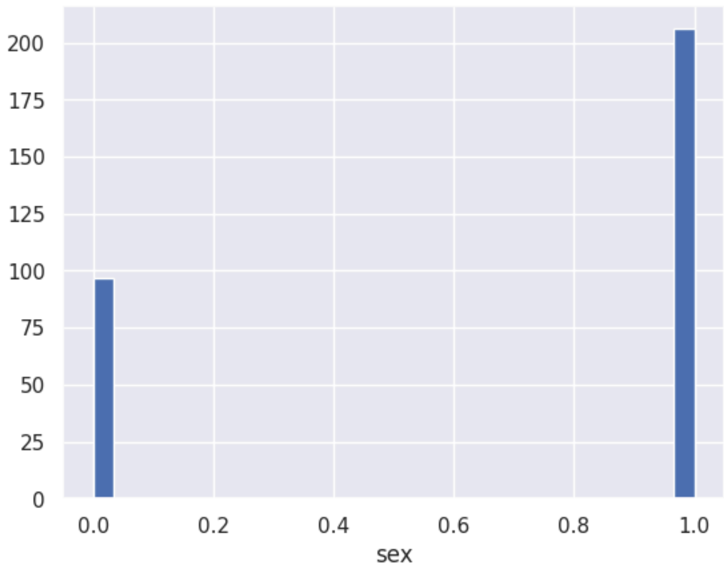
```
3.6     4
2.2     4
3.4     3
0.9     3
2.4     3
0.3     3
4.0     3
1.1     2
4.2     2
2.3     2
2.5     2
3.2     2
5.6     1
2.9     1
6.2     1
2.1     1
1.3     1
3.1     1
3.8     1
0.7     1
3.5     1
4.4     1
Name: count, dtype: int64


slope
1.0    142
2.0    140
3.0     21
Name: count, dtype: int64


ca
0.0    176
1.0     65
2.0     38
3.0     20
?        4
Name: count, dtype: int64


thal
3.0    166
7.0    117
6.0     18
?        2
Name: count, dtype: int64


num
0    164
1     55
2     36
3     35
4     13
Name: count, dtype: int64
```

Visualizing Features

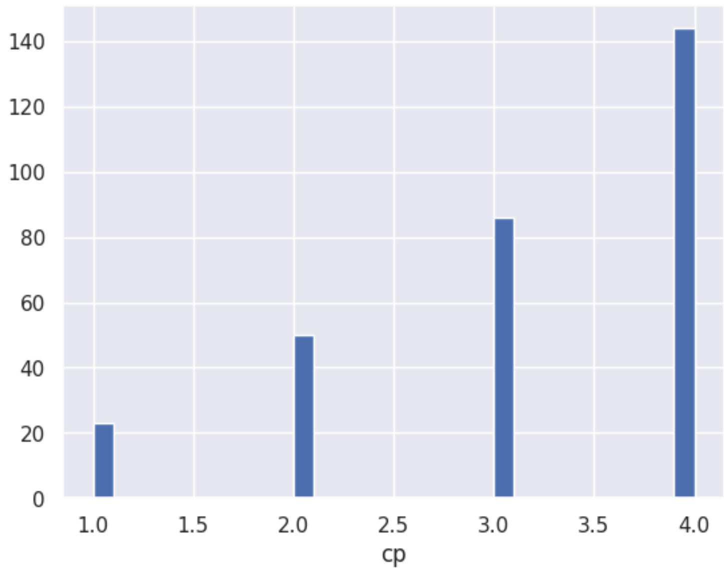
```
sns.set()
for column in df:
  plt.hist(x = df[column], bins = 30)
  plt.xlabel(column)
  plt.show()
  print('\n')
plt.tight_layout()
```



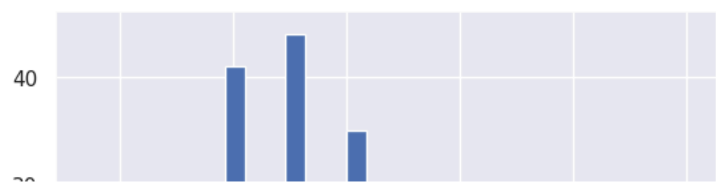
30
25
20
15
10
5
0
30
40
50
60
70
age



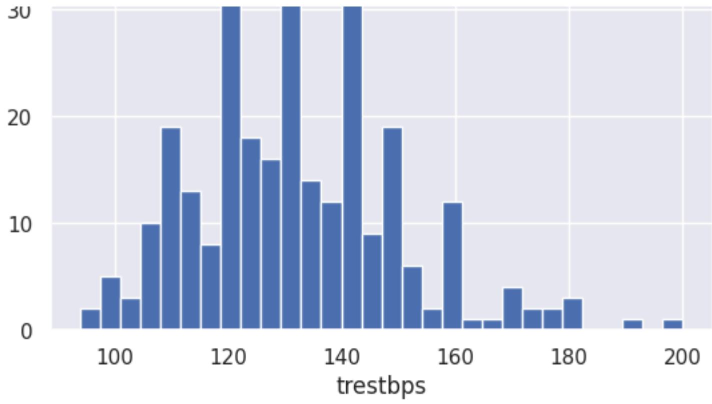
200
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
sex



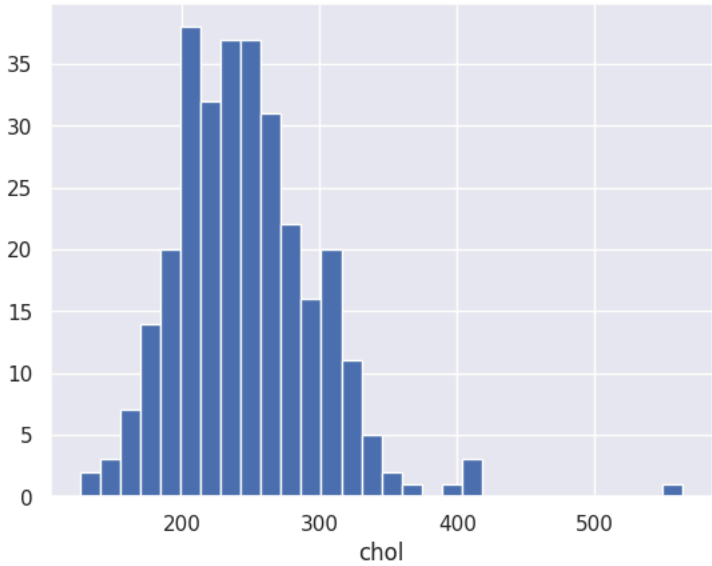
140
120
100
80
60
40
20
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
cp



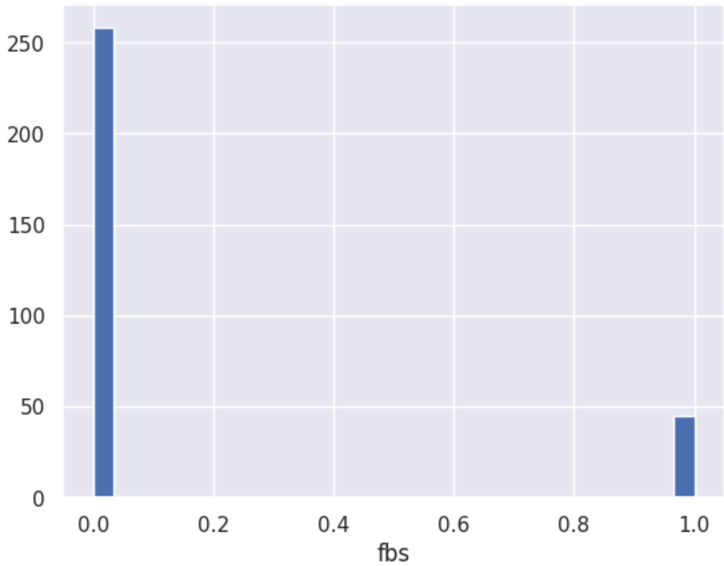
40



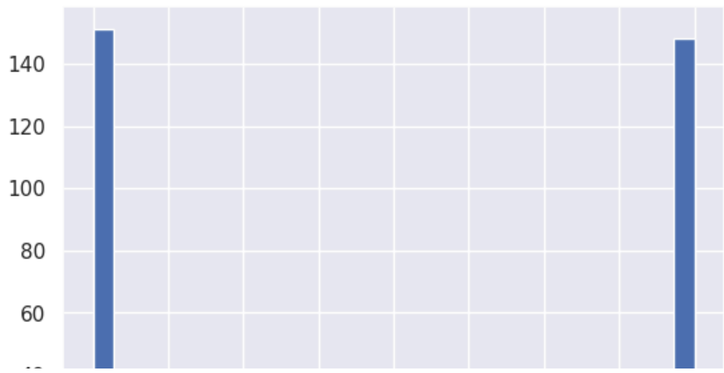
30
20
10
0
100
120
140
160
180
200
trestbps



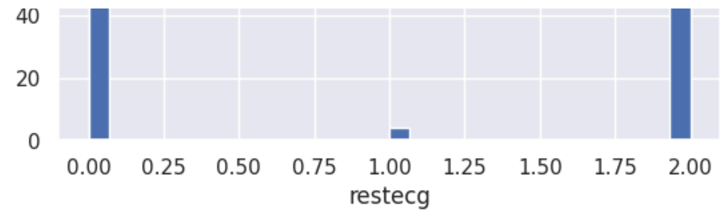
35
30
25
20
15
10
5
0
200
300
400
500
chol

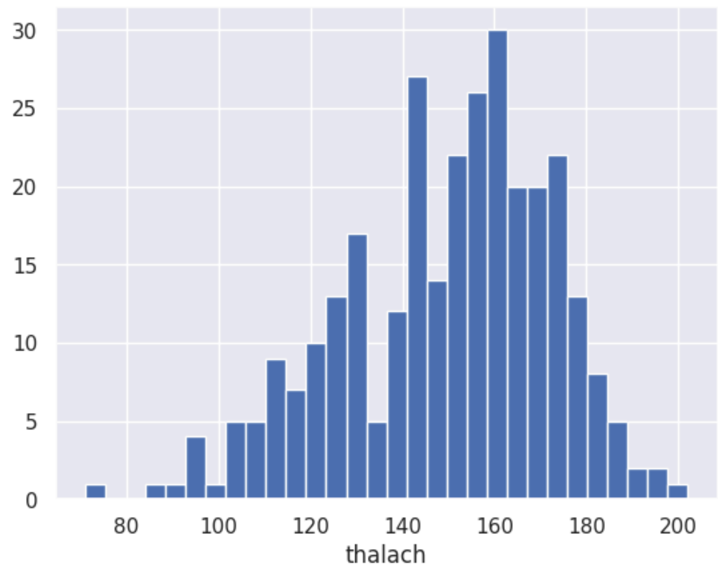


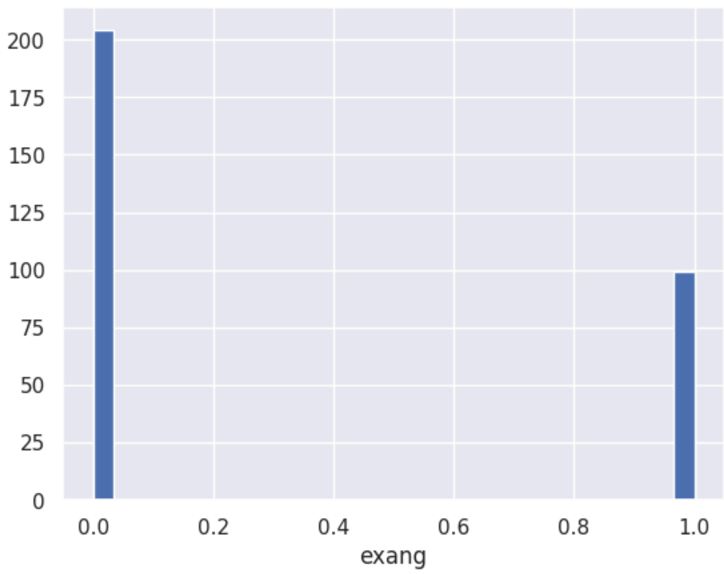
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
fbs

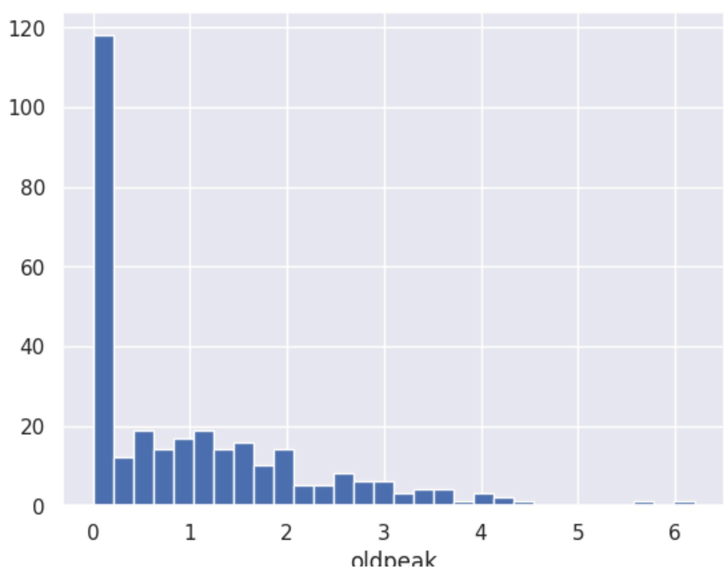


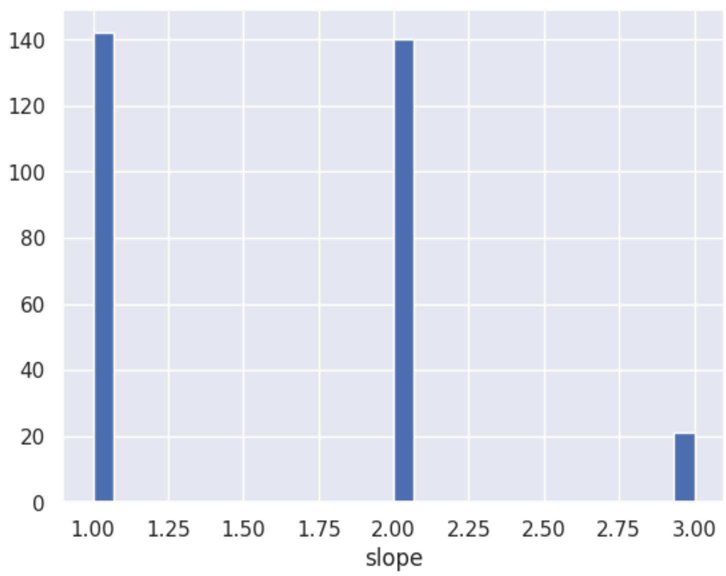
140
120
100
80
60


restecg

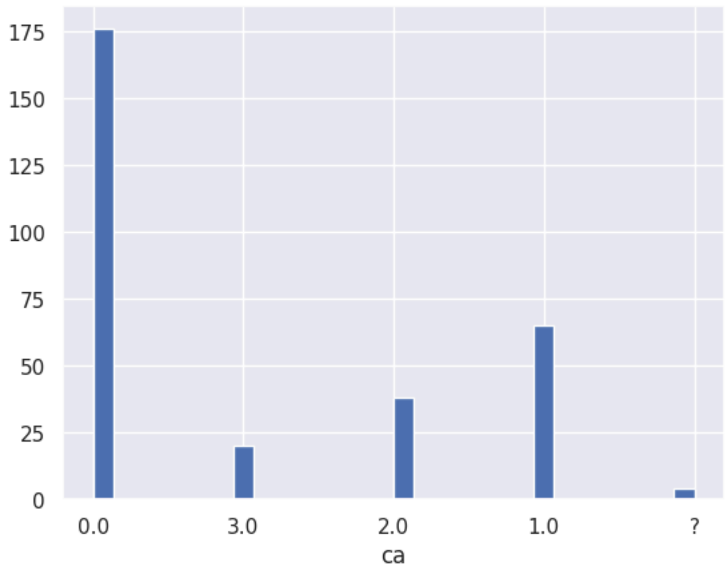

thalach


exang

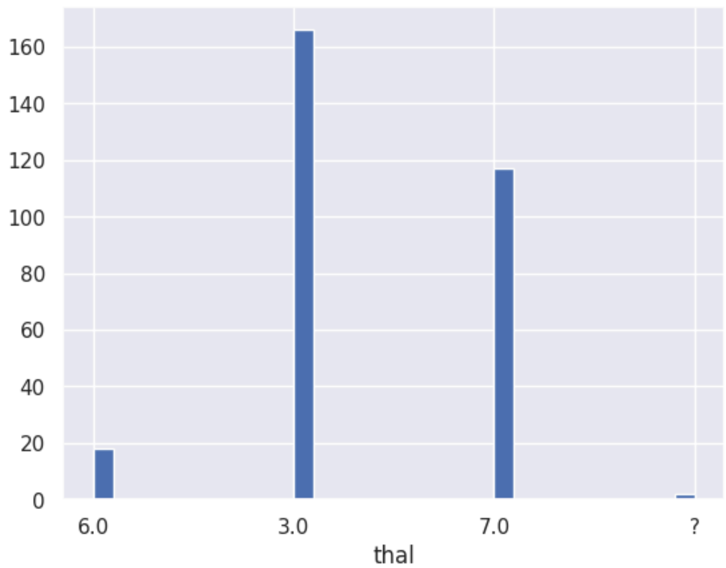

oldpeak



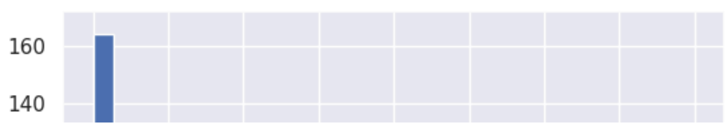
140
120
100
80
60
40
20
0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
slope



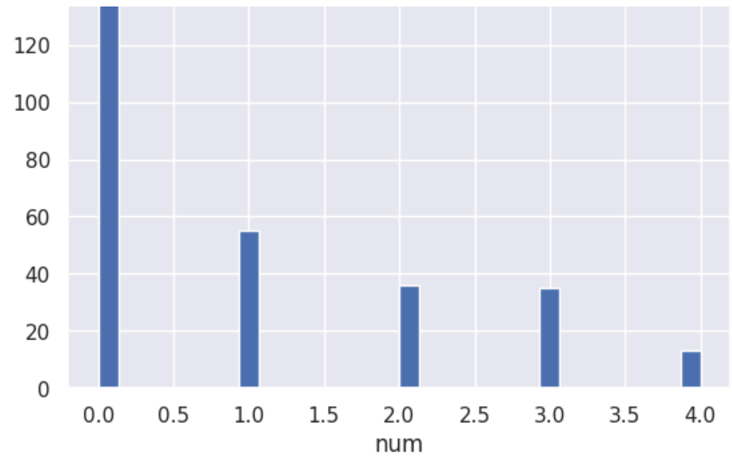
175
150
125
100
75
50
25
0
0.0
3.0
2.0
1.0
?
ca



160
140
120
100
80
60
40
20
0
6.0
3.0
7.0
?
thal



160
140




```
<Figure size 640x480 with 0 Axes>
```

Observations:

1. As we can see, the features 'ca' and 'thal' contain '?' value which represents missing value. We need to impute these values.
2. In the target variable 'num', we are only interested in 0 and 1 values (as given in data description). Hence, labels with values of 2,3,4 will have to be converted to 1.
3. Continuous numerical features: age, trestbps, chol, thalach, oldpeak
4. Discrete numerical features: sex, cp, fbs, restecg, exang, slope, ca, thal

Creating Pipeline

```
class preprocessor:

  def train_test_split(self,df):

    # Replace '?' values with np.nan
    df.replace('?',np.nan,inplace=True)

    # Change values 2,3,4 to 1 in target variable 'num'
    df['num'].replace([2,3,4],1,inplace=True)
    X = df.drop('num',axis=1)
    y = df['num']

    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,shuffle=True,random_state=0)
    return X_train, X_test, y_train, y_test

  def pipeline(self):

    #Continuous and Discrete Numerical Variables
    continuous_numerical_columns = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
    discrete_numerical_columns = ['sex','cp','fbs','restecg','exang','slope','ca','thal']

    continuous_numerical_transformer = Pipeline(steps=[
        ('imputer',SimpleImputer(strategy='median')),
        ('scaler',StandardScaler()),
        ('pca',PCA())
    ])

    discrete_numerical_transformer = Pipeline(steps=[
        ('imputer',SimpleImputer(strategy='most_frequent')),
        ('onehot',OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer(transformers=[
        ('continuous_num',continuous_numerical_transformer,continuous_numerical_columns),
        ('discrete_num',discrete_numerical_transformer,discrete_numerical_columns),
        ])

    pipe = Pipeline(steps=[
    ('preprocessor',preprocessor),
    ('regressor',LogisticRegression(max_iter=1000))
    ])

    return pipe
```

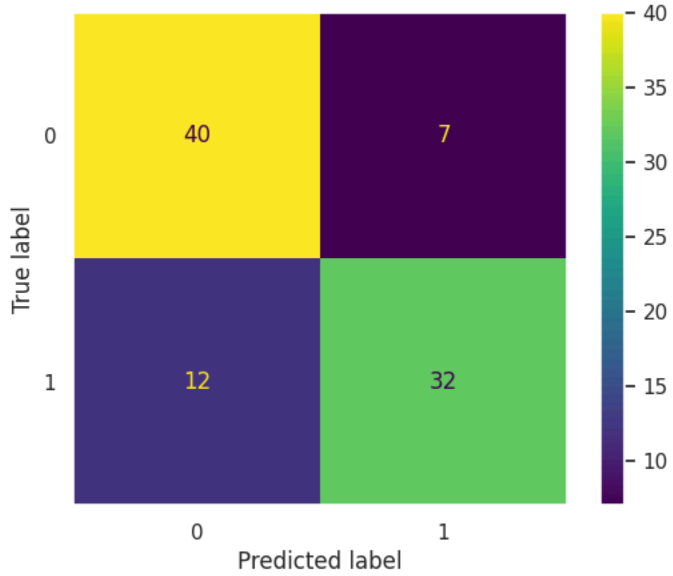
Logistic Regressor without CV

```
pp = preprocessor()
X_train, X_test, y_train, y_test = pp.train_test_split(df)
pipe = pp.pipeline()
pipe.fit(X_train,y_train)
print(f"Score on test data : {pipe.score(X_test,y_test):.3f}")
```

```
Score on test data : 0.791
```

Let us plot the Confusion Matrix.

```
y_test_pred = pipe.predict(X_test)
cm = confusion_matrix(y_test,y_test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[0,1])
disp.plot()
plt.grid(False)
```




```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.85      0.81        47
           1       0.82      0.73      0.77        44

    accuracy                           0.79        91
   macro avg       0.79      0.79      0.79        91
weighted avg       0.79      0.79      0.79        91
```

Logistic Regressor with CV

Since some of the solver functions use only L2 regularization, I have considered two separate param-grids.

CASE1

```
param_grid1 = {'regressor__C' : np.logspace(-4,4,20),
              'regressor__solver' : ['lbfgs','newton-cg','sag','newton-cholesky'],
              'regressor__penalty' : ['l2'],
              'regressor__warm_start' : [True]}

random_search1 = RandomizedSearchCV(pipe, param_grid1, n_jobs=-1, n_iter=80, cv=5, verbose=True, random_state=0)
random_search1.fit(X_train,y_train)
print(f"Best parameters for random_search1 : \n{random_search1.best_params_}")
print(f"Best CV score for random_search1 : {random_search1.best_score_:.3f}")
print(f"Score on test data : {random_search1.score(X_test,y_test):.3f}")
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
Best parameters for random_search1 :
```

```
{'regressor__warm_start': True, 'regressor__solver': 'lbfgs', 'regressor__penalty': 'l2', 'regressor__C': 0.08858667904100823}
Best CV score for random_search1 : 0.873
Score on test data : 0.780
```

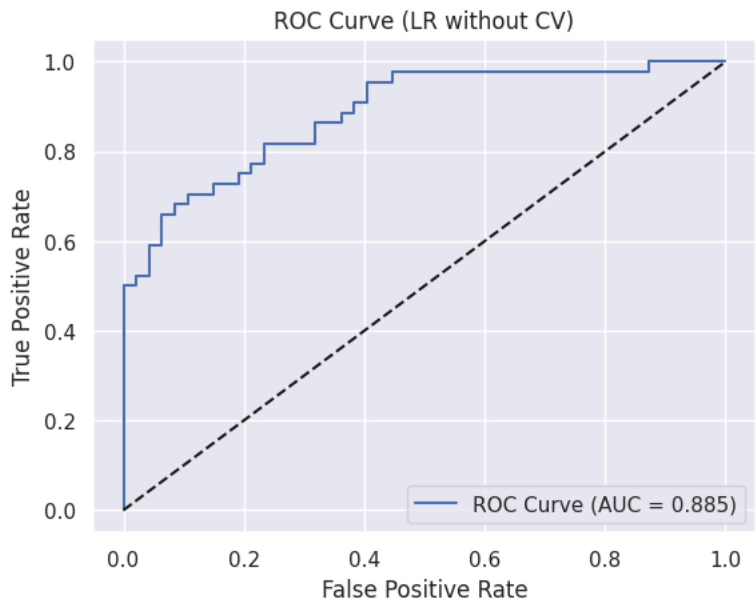
CASE2

```
param_grid2 = {'regressor__C' : np.logspace(-4,4,20),
             'regressor__solver' : ['liblinear','saga'],
             'regressor__penalty' : ['l1','l2'],
             'regressor__warm_start' : [True]}

random_search2 = RandomizedSearchCV(pipe, param_grid2, n_jobs=-1, n_iter=80, cv=5, verbose=True, random_state=0)
random_search2.fit(X_train,y_train)
print(f"Best parameters for random_search2 : \n{random_search2.best_params_}")
print(f"Best CV score for random_search2 : {random_search2.best_score_:.3f}")
print(f"Score on test data : {random_search2.score(X_test,y_test):.3f}")
```

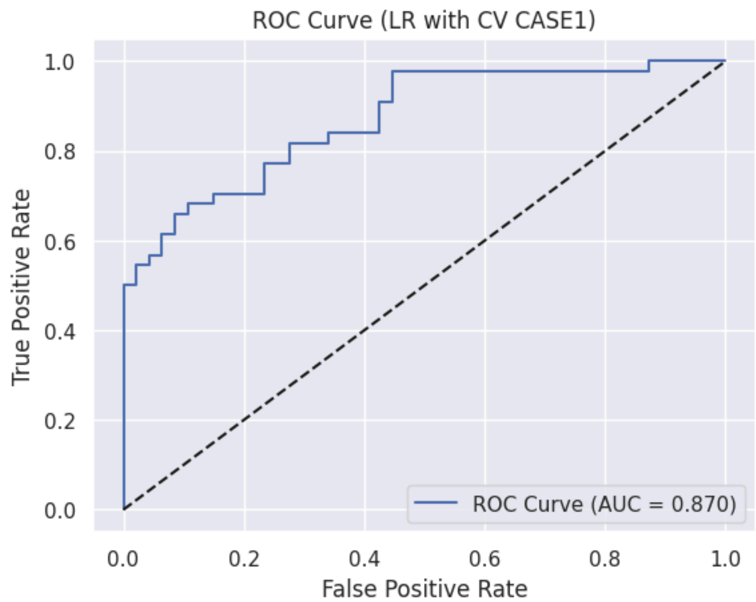
```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
Best parameters for random_search2 :
{'regressor__warm_start': True, 'regressor__solver': 'liblinear', 'regressor__penalty': 'l2', 'regressor__C': 0.08858667904100823}
Best CV score for random_search2 : 0.873
Score on test data : 0.780
```

ROC-AUC Curves

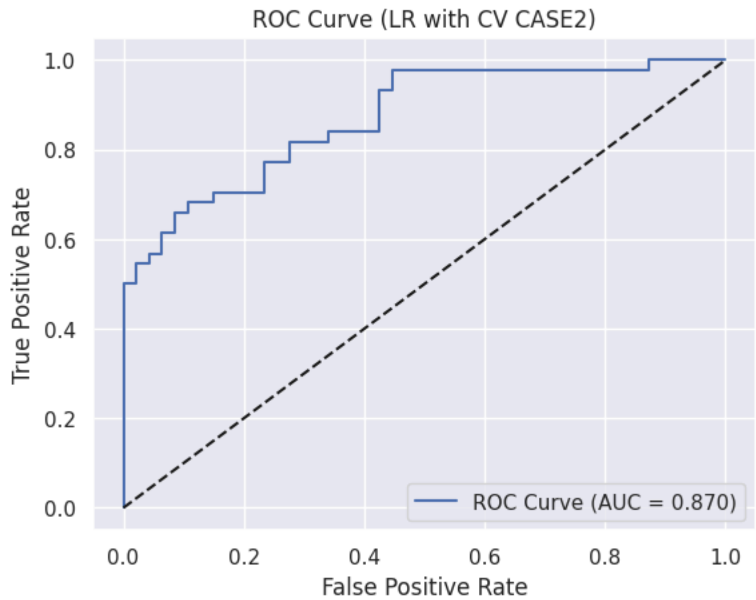
```
y_test_prob = pipe.predict_proba(X_test)[:,1]

fpr, tpr, thresholds = roc_curve(y_test, y_test_prob)
roc_auc = roc_auc_score(y_test, y_test_prob)

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (LR without CV)')
plt.legend(loc='lower right')
plt.show()
```




```
y_test_prob = random_search1.predict_proba(X_test)[:,1]

fpr, tpr, thresholds = roc_curve(y_test, y_test_prob)
roc_auc = roc_auc_score(y_test, y_test_prob)

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (LR with CV CASE1)')
plt.legend(loc='lower right')
plt.show()
```




```
y_test_prob = random_search2.predict_proba(X_test)[:,1]

fpr, tpr, thresholds = roc_curve(y_test, y_test_prob)
roc_auc = roc_auc_score(y_test, y_test_prob)

plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (LR with CV CASE2)')
plt.legend(loc='lower right')
plt.show()
```



From the three ROC Curves, the AUC Score for first model is the highest. One possibility can be that the plain Logistic Regressor could be overfitted, leading to higher AUC score than the one with CV.

```
Start coding or generate with AI.
```