```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


df=pd.read_csv("/content/heart.csv")


df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

```
df.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak

```
# 1) data cleaning
# A. Data cleaning
# Data cleaning involves identifying and handling missing values, duplicate data, outliers, and
# other inconsistencies in the data.
# For the dataset, we will check for missing values and remove any columns that have
# more than 50% missing values.
# # Check for missing values

df.dropna(thresh=0.5 * len(df),axis=1)
# df.dropna() is a method of the DataFrame class in Pandas used to remove rows or columns with missing values (NaN).
# thresh=0.5 * len(df) specifies the threshold for non-null values. In this case, it's set to 50% of the total number of rows in the DataFrame. This means that any column with less than 50% no
# axis=1 indicates that we want to drop columns with missing values.
# inplace=True modifies the DataFrame in-place, meaning the changes will be made directly to the df DataFrame object
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

```
df1=df[['age','sex','cp','ca']].loc[0:15]
df1
```

	age	sex	cp	ca
0	52	1	0	2
1	53	1	0	0
2	70	1	0	0
3	61	1	0	1
4	62	0	0	3
5	58	0	0	0
6	58	1	0	3
7	55	1	0	1
8	46	1	0	0

```
df2=df[['age','sex','cp','ca']].loc[16:30]
df2
```

	age	sex	cp	ca
16	51	0	2	1
17	54	1	0	1
18	50	0	1	0
19	58	1	2	0
20	60	1	2	0
21	67	0	0	2
22	45	1	0	0
23	63	0	2	0
24	42	0	2	0
25	61	0	0	0
26	44	1	2	0
27	58	0	1	2
28	56	1	2	1
29	55	0	0	0
30	44	1	0	0

```
merge=pd.merge(df1,df2,on='age',how='inner')
merge
```

	age	sex_x	cp_x	ca_x	sex_y	cp_y	ca_y
0	61	1	0	1	0	0	0
1	58	0	0	0	1	2	0
2	58	0	0	0	0	1	2
3	58	1	0	3	1	2	0
4	58	1	0	3	0	1	2

```
#data transformation
# Data transformation involves converting data into a different format or structure. In this
# example, we will transform the heart disease dataset by converting the target column into binary
# values (0 and 1).

df['target']=df['target'].apply(lambda x:1 if x>0 else 0)
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
0	52	1	0	125	212	0	1	168	0	1.0	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	
4	62	0	0	138	294	1	1	106	0	1.9	1	
...	...	...	...	...	...	...	...	...	...	...	...	
1020	59	1	1	140	221	0	1	164	1	0.0	2	
1021	60	1	0	125	258	0	0	141	1	2.8	1	
1022	47	1	0	110	275	0	0	118	1	1.0	1	
1023	50	0	0	110	254	0	0	159	0	0.0	2	
1024	54	1	0	120	188	0	1	113	0	1.4	1	

1025 rows × 14 columns

```
#Error Correction
df = df.applymap(lambda x: df.mean() if x < 0 else x)
df
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
0	52	1	0	125	212	0	1	168	0	1.0	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	
4	62	0	0	138	294	1	1	106	0	1.9	1	
...	...	...	...	...	...	...	...	...	...	...	...	
1020	59	1	1	140	221	0	1	164	1	0.0	2	

```
# 5) Model Building
from sklearn.model_selection import train_test_split

X = merge.drop(['age'], axis=1)
y = merge['age']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Build the logistic regression model
from sklearn.linear_model import LogisticRegression

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

# Evaluate the model
from sklearn.metrics import classification_report, confusion_matrix

y_pred = logreg.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))


# train_test_split is a function from scikit-learn that splits the data into training and testing sets. Here, it splits the data X and y into training and testing sets, with 70% of the data 4
# used for training and 30% for testing. The X variable contains the features (all columns except 'age'), and y contains the target variable ('age').

# LogisticRegression is a class from scikit-learn that represents the logistic regression model.
# logreg.fit(X_train, y_train) trains the logistic regression model on the training data.

# y_pred = logreg.predict(X_test) uses the trained logistic regression model to make predictions on the test data (X_test), and stores the predicted values in y_pred.
# confusion_matrix(y_test, y_pred) computes the confusion matrix, which is a table that shows the true positive, false positive, true negative, and false negative values.
# classification_report(y_test, y_pred) generates a text report with precision, recall, F1-score, and support for each class.
```

```
    [[0 0 0 0]
     [1 0 0 0]
     [0 0 1 0]
     [1 0 0 0]]
                  precision    recall  f1-score   support

              54       0.00      0.00      0.00         0
              55       0.00      0.00      0.00         1
              58       1.00      1.00      1.00         1
              61       0.00      0.00      0.00         1
```

```
    accuracy                           0.33         3
   macro avg       0.25      0.25      0.25         3
weighted avg       0.33      0.33      0.33         3

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predi
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sam
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predi
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sam
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predi
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true sam
  _warn_prf(average, modifier, msg_start, len(result))
```