Importing the Dependencies

```
import numpy as np import pandas as pd from
sklearn.model_selection import train_test_split from
sklearn.linear_model import LogisticRegression from
sklearn.metrics import accuracy_score
```

Data Collection and Processing

```
#loding the csv data to a Pandas DataFrame heart_data
= pd.read_csv('/content/heart.csv')
```

```
#print first 5 rows of the dataset heart_data.head()
```

	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

Next steps: Generate code with heart_data View recommended plots New interactive sheet

```
#print last 5 rows of the dataset heart_data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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

```
#number of rows and colmns in the dataset heart_data.shape
```

```
(1025, 14)
```

```
#getting some info about the data heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
 RangeIndex: 1025 entries, 0 to 1024 Data
 columns (total 14 columns):
  #   Column    Non-Null Count  Dtype
 ---  ------    --------------  -----
 0    age       1025 non-null   int64
 1    sex       1025 non-null   int64
 2    cp        1025 non-null   int64
 3    trestbps  1025 non-null   int64
 4    chol      1025 non-null   int64
 5    fbs       1025 non-null   int64
 6    restecg   1025 non-null   int64
 7    thalach   1025 non-null   int64
 8    exang     1025 non-null   int64
 9    oldpeak   1025 non-null   float64
 10   slope     1025 non-null   int64
 11   ca        1025 non-null   int64
 12   thal       1025 non-null    int64    13   target     1025 non-null    int64
 dtypes: float64(1), int64(13) memory usage: 112.2 KB
```

```
#checking          for         missing          values
heart_data.isnull().sum()
```

	0
age	0
sex	0
cp	0

trestbps	0
chol	0
fbs	0
restecg	0
thalach	0
exang	0
oldpeak	0
slope	0
ca	0
thal	0
target	0

dtype: int64

```
#statistical measures about the data heart_data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	0.336585	1.071512
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	0.472772	1.175053
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	0.000000	0.000000
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	0.000000	0.800000
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	1.000000	1.800000
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	1.000000	6.200000

```
#checking       the       distribution       of       the       target       variables
heart_data['target'].value_counts()
```

	count
target	
1	526
0	499

dtype: int64

1 -----> Defective Heart

0 -----> Healthly Heart

Splitting the Features and Target

```
X  =  heart_data.drop(columns='target',  axis=1)
Y = heart_data['target']
```

```
print(X)
```

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

       slope   ca   thal   0
  2   2     3
  1         0   0     3
  2         0   0     3
  3         2   1     3
  4         1   3     2  ...     ...  ..   ...
  1020      2   0     2
```

```
1021      1   1     3
1022      1   1     2
1023      2   0     2
1024      1   1     3

1025      rows x 13 columns]
```

```
print(Y)
```

```
0       0
 1       0
 2       0
 3       0
 4       0        ..
 1020    1
 1021    0
 1022    0
 1023    1
 1024    0
 Name: target, Length: 1025, dtype: int64
```

Splitting the data into Training data & Test data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
#
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(1025, 13) (820, 13) (205, 13)
```

Model Training

Logist ic Regression

```
model = LogisticRegression()
```

```
#training the LogisticRegression model with Training data model.fit(X_train,
Y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=
 STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

 Increase the number of iterations (max_iter) or scale the data as shown in:
 https://scikit-learn.org/stable/modules/preprocessing.html Please also refer to
 the documentation for alternative solver options:         https://scikit-
 learn.org/stable/modules/linear_model.html#logistic-regression      n_iter_i   =
 _check_optimize_result( ▾ LogisticRegression i ?

 LogisticRegression()
```

Model Evaluation

Accuracy Score

```
#accuracy on training data
X_train_prediction = model.predict(X_train) training_data_accuracy =
accuracy_score(X_train_prediction, Y_train)
```

```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.8524390243902439
```

```
#accuracy on test data
X_test_prediction = model.predict(X_test) test_data_accuracy =
accuracy_score(X_test_prediction, Y_test)
```

```
print('Accuracy on Test data : ', test_data_accuracy)
```

```
Accuracy on Test data :  0.8048780487804879 Building
```

a Predictive System

```
input_data = (71,0,0,112,149,0,1,125,0,1.6,1,0,2)

#change the input data to a numpy array input_data_as_numpy_array
= np.asarray(input_data)

#reshape   the   numpy   array   as   we   are   predicting   for   only   on   instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction          =          model.predict(input_data_reshaped)
print(prediction)

if(prediction[0]==0):
  print('The Person does not have a Heart Disease') else:
  print('The Person has Heart Disease')
```

```
[1]
    The Person has Heart Disease
```