1xp9qwl1r

February 20, 2025

[3]:
```
# Import all the Dependencies

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

[4]:
```
# Import the heart_csv dataset
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving heart.csv to heart.csv
```

[6]:
```
# Load the dataset into a dataframe
heart_data = pd.read_csv('heart.csv')
```

[7]:
```
# Print the first 5 rows of the dataset
heart_data.head()
```

[7]:

	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

[8]:
```
# Print the last 5 rows of the dataset
heart_data.tail()
```

[8]:
```
      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
```

[9]:
```
# Number of rows & columns in the dataset
heart_data.shape
```

[9]:
```
(1025, 14)
```

[10]:
```
# Getting information 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
```

[11]:
```
# Checking for missing values
heart_data.isnull().sum()
```

[11]:
```
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
```

[12]:
```
# Get statistical measure of the data
heart_data.describe()
```

[12]:

	age	sex	cp	…	ca	thal	target
count	1025.000000	1025.000000	1025.000000	…	1025.000000	1025.000000	1025.000000
mean	54.434146	0.695610	0.942439	…	0.754146	2.323902	0.513171
std	9.072290	0.460373	1.029641	…	1.030798	0.620660	0.500070
min	29.000000	0.000000	0.000000	…	0.000000	0.000000	0.000000
25%	48.000000	0.000000	0.000000	…	0.000000	2.000000	0.000000
50%	56.000000	1.000000	1.000000	…	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	…	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	…	4.000000	3.000000	1.000000

[8 rows x 14 columns]

[13]:
```
# How is the target value distributed
heart_data['target'].value_counts()
```

[13]:
```
target
1    526
0    499
Name: count, dtype: int64
```

[14]:
```
# Splitting the features and target
X = heart_data.drop(columns='target', axis=1)
Y = heart_data['target']
```

[15]:
```
print(X)
print(Y)
```

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

[1025 rows x 13 columns]
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
```

[16]:
```
# Split the data into training and testing data
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size=0.2,␣
↪stratify=Y, random_state=2)
```

[17]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

[18]:
```
# Model training
# Logistic Regression
model = LogisticRegression()
```

[19]:
```
# Training the Logistic Regression moidel 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=1):
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(
```

[19]:
```
LogisticRegression()
```

[20]:
```
# Model Evaluation
# Accuracy_score as evaluation metrics
# Accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

[21]:
```
# Finding accuracy_score of taining data
print('Accuracy on training data:',training_data_accuracy)
```

```
Accuracy on training data: 0.8524390243902439
```

[22]:
```
# Accuracy score on the testing data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

[23]:
```
# Finding the accuracy score of test data
print('Accuracy on test data:', test_data_accuracy)
```

```
Accuracy on test data: 0.8048780487804879
```

[28]:
```
# Building a predictive system
input_data = (61,1,1,0,149,203,0,1,161,0,0,2,1)

# Change the input data to np.array
input_data_as_numpy_array = np.asarray(input_data)

#Reshape the numpy array as we are predicting for only one instance
input_data_reshape = np.reshape(input_data_as_numpy_array, (1,-1))

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

if prediction ==[1]:
  print('The person does not have a heart disease')
else:
  print('The person has heart disease')
```

```
[0]
The person has heart disease

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LogisticRegression was
fitted with feature names
  warnings.warn(
```