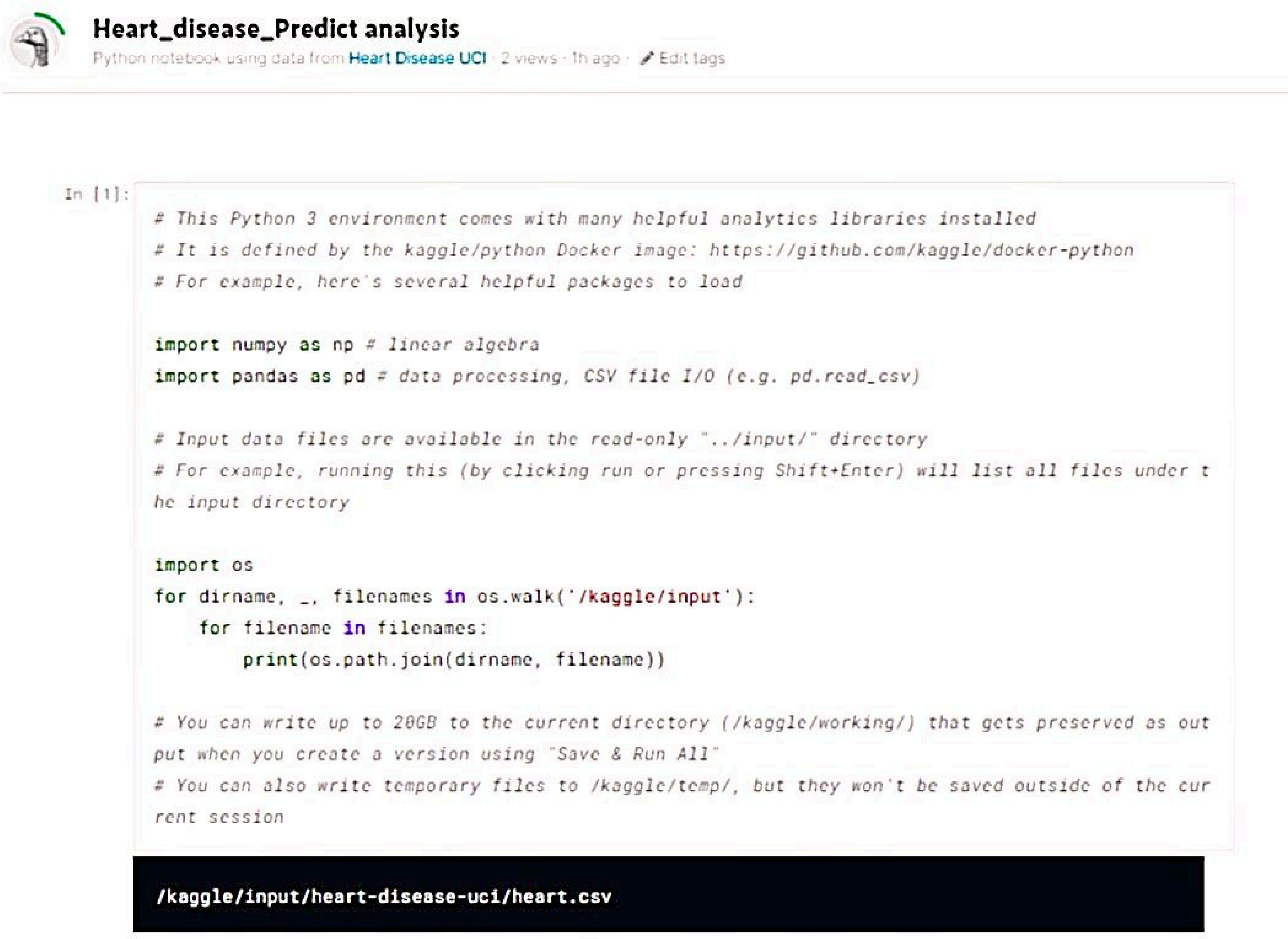
Data:

https://www.kaggle.com/ronitf/heart-disease-uci

TASK:
Make a neural network to predict on the "target" column of the dataset

Steps to complete:

1. Specify Architecture
2. Compile the model
3. Fit the model
4. Predict



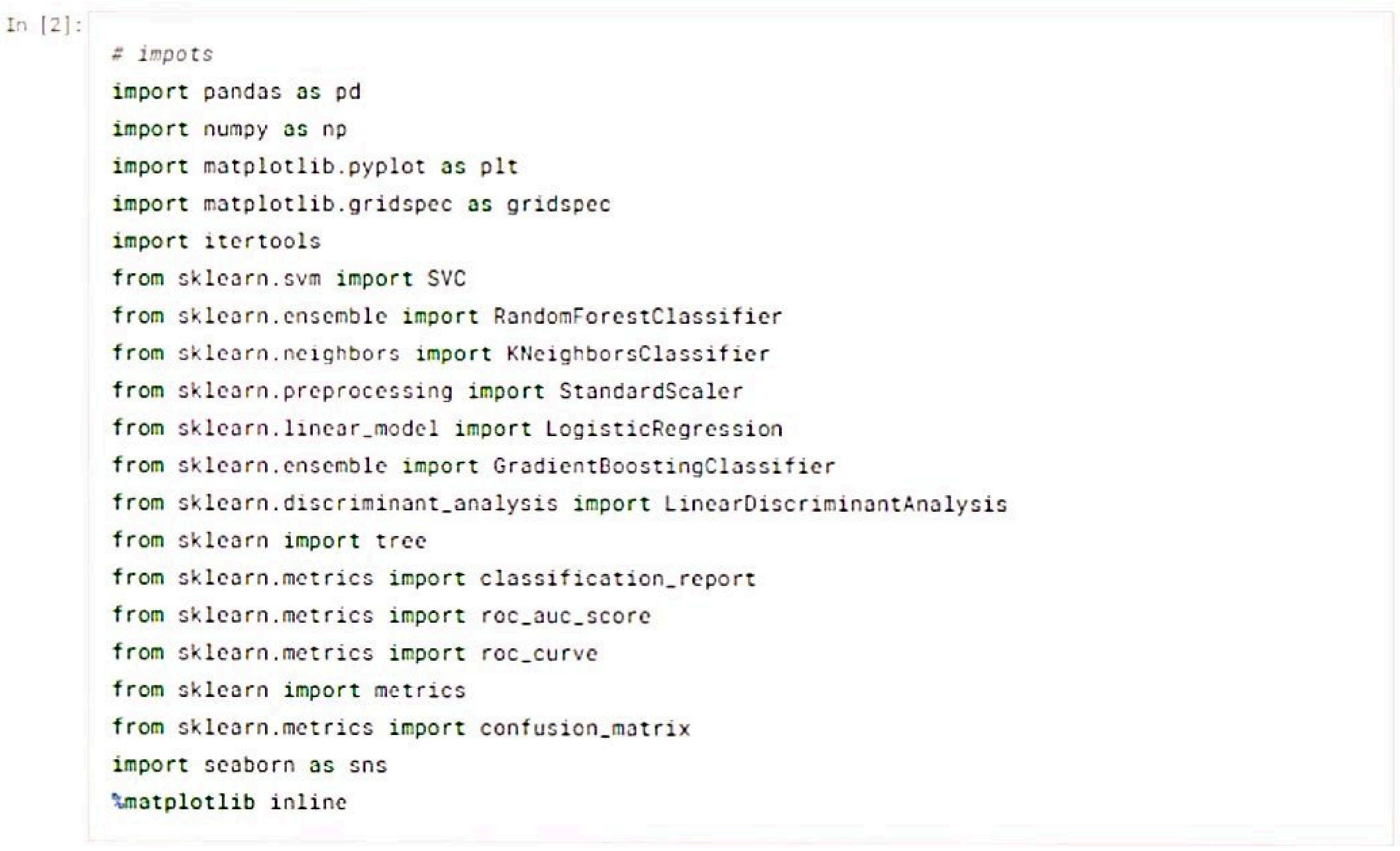
Heart_disease_Predict analysis

Python notebook using data from Heart Disease UCI · 2 views · 1h ago · Edit tags

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under t
he input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as out
put when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the cur
rent session
```

```
/kaggle/input/heart-disease-uci/heart.csv
```



In [2]:

```
# impots
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import itertools
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn import tree
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline
```

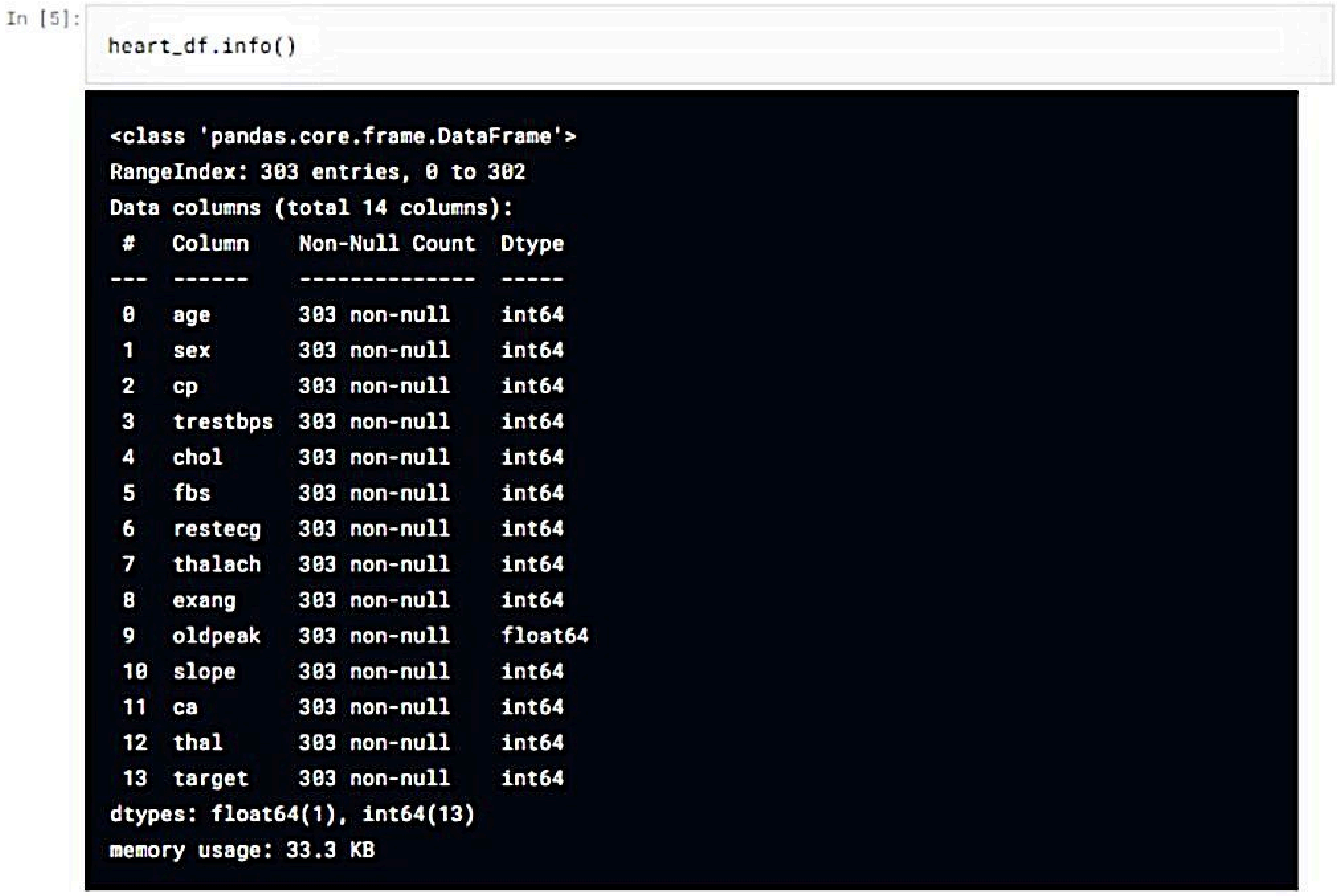
In [3]:

```
import os
print(os.listdir("../input"))
```

```
['heart-disease-uci']
```

In [4]:

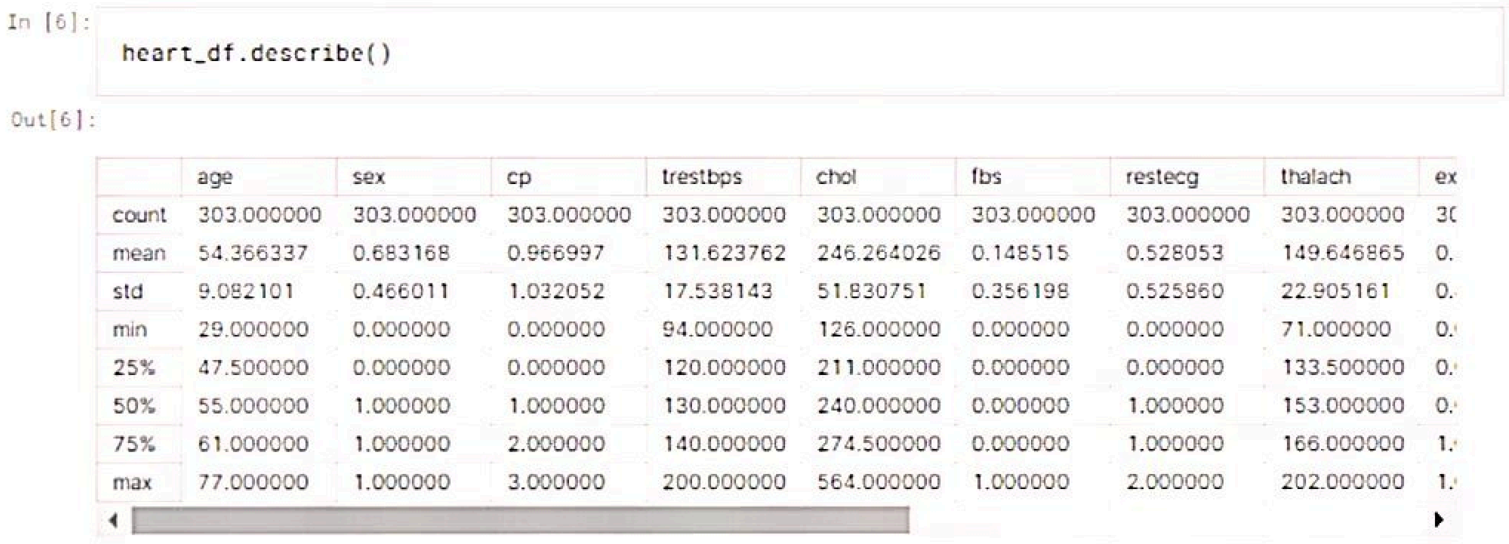
```
heart_df = pd.read_csv('../input/heart-disease-uci/heart.csv')
```



In [5]:

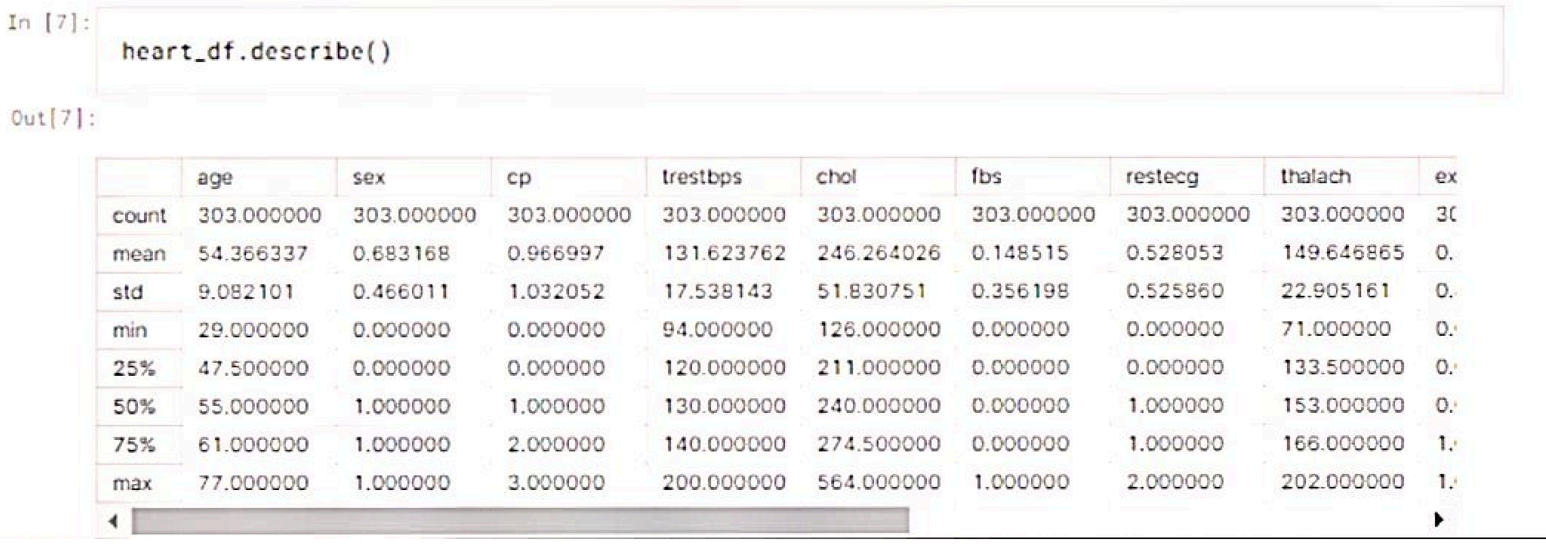
```python
heart_df.info()
```

```
<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    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```



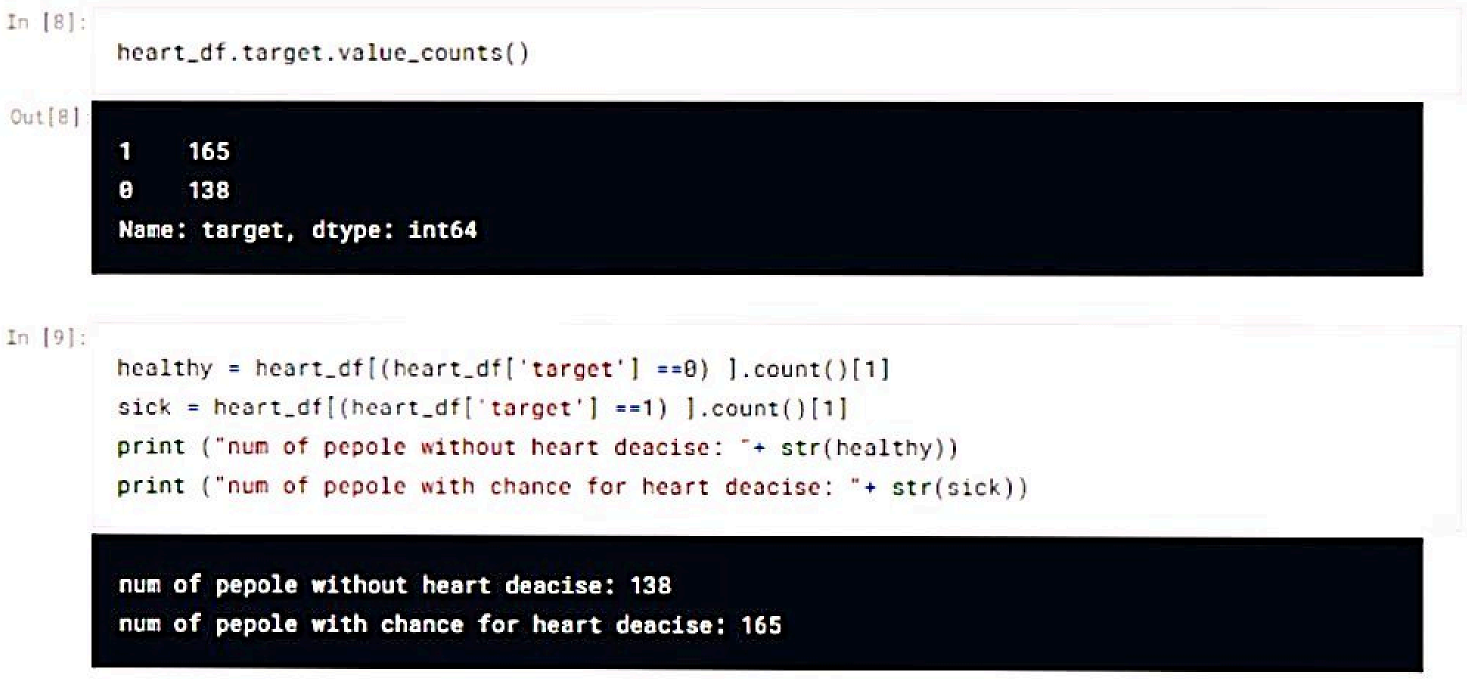
In [6]:
```
heart_df.describe()
```
Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	ex
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.



In [7]:
```
heart_df.describe()
```
Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	ex
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.



In [8]:

```
heart_df.target.value_counts()
```

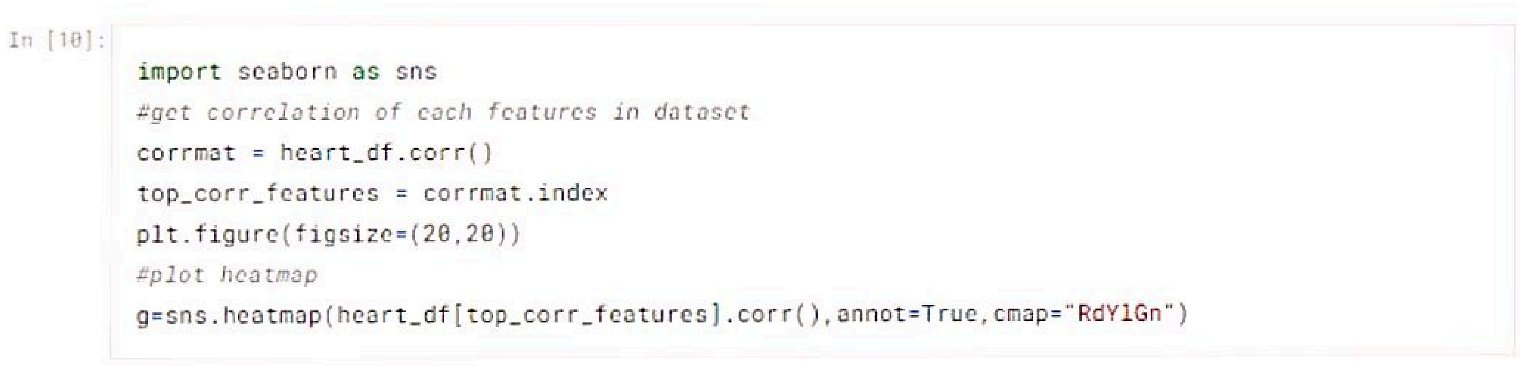
Out[8]:

```
1    165
0    138
Name: target, dtype: int64
```

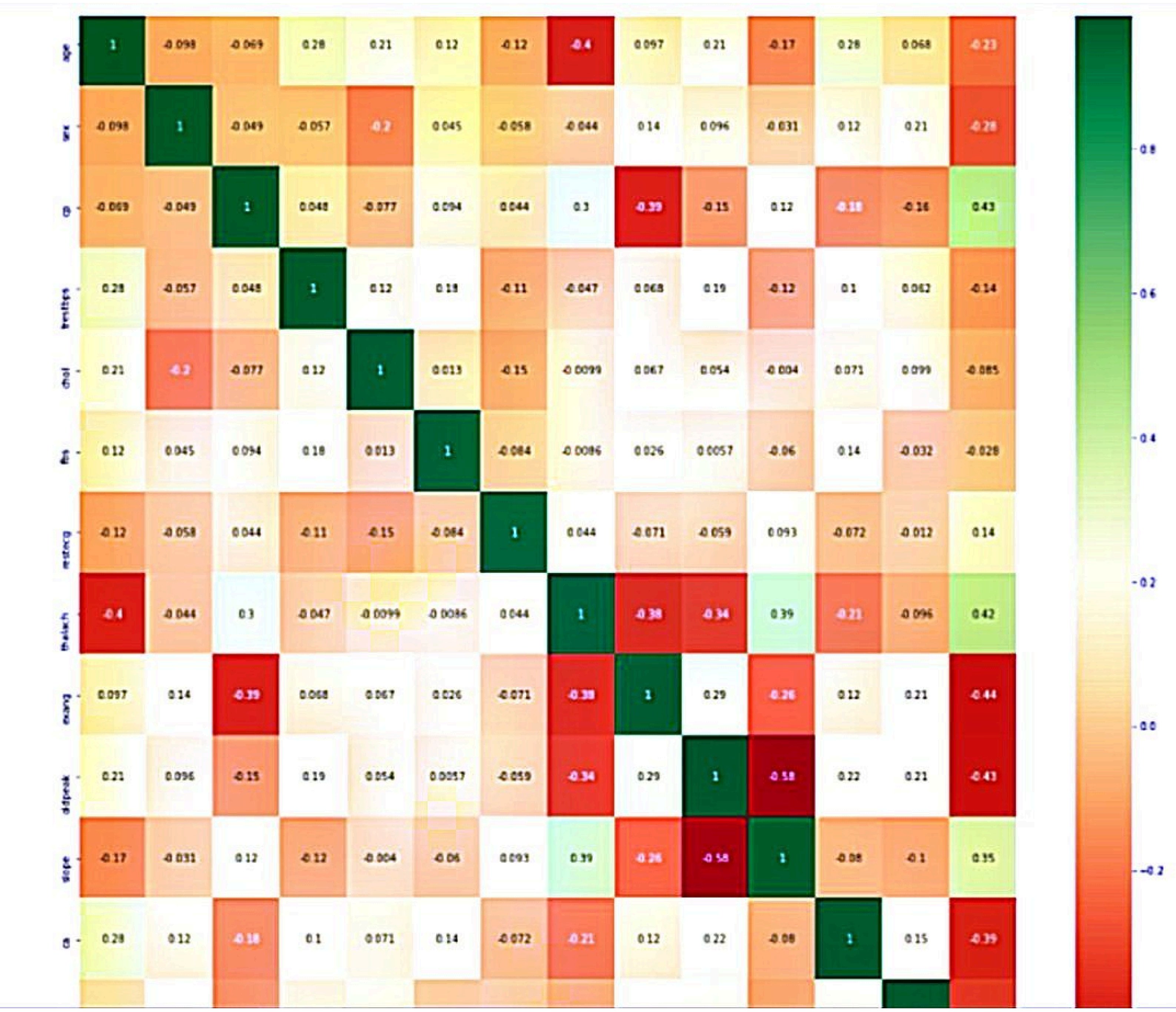
In [9]:

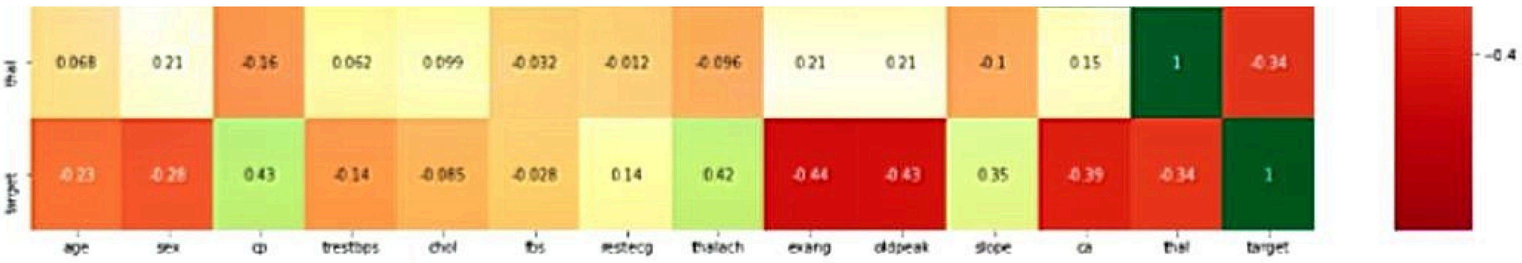
```
healthy = heart_df[(heart_df['target'] ==0) ].count()[1]
sick = heart_df[(heart_df['target'] ==1) ].count()[1]
print ("num of pepole without heart deacise: "+ str(healthy))
print ("num of pepole with chance for heart deacise: "+ str(sick))
```

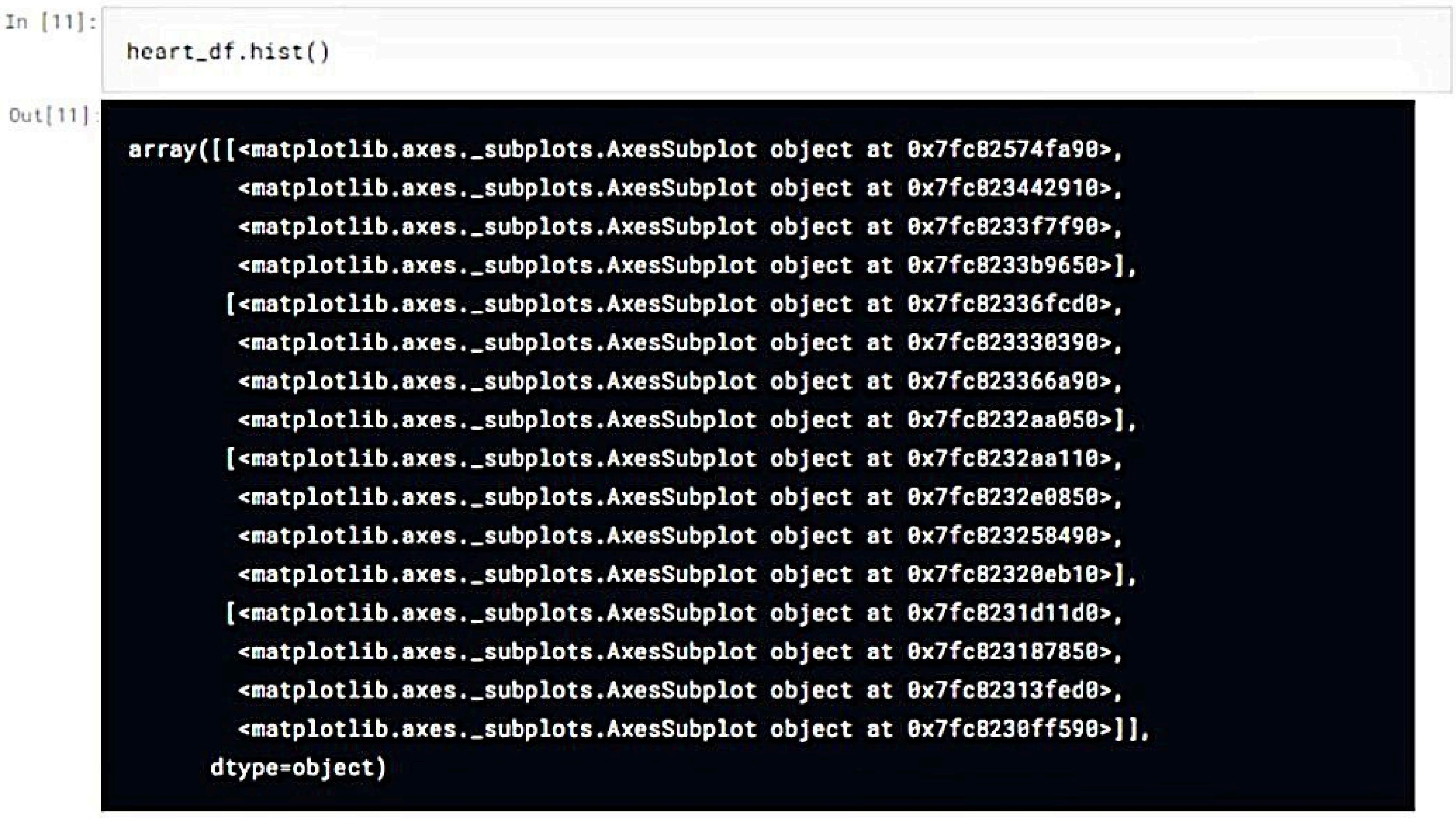
```
num of pepole without heart deacise: 138
num of pepole with chance for heart deacise: 165
```



```
In [10]:
import seaborn as sns
#get correlation of each features in dataset
corrmat = heart_df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heatmap
g=sns.heatmap(heart_df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

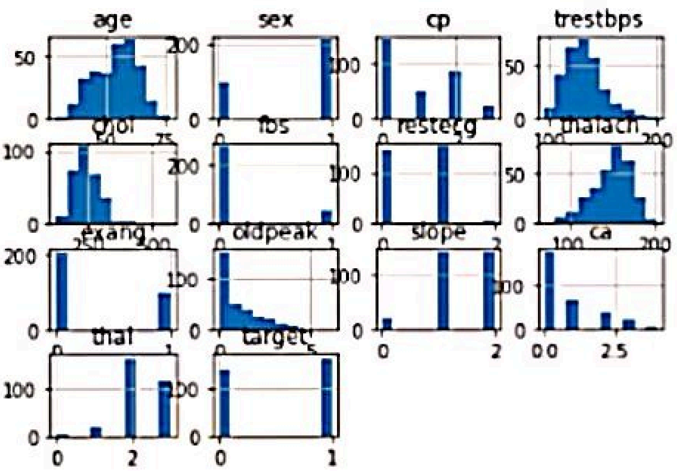

age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
1 -0.098 -0.069 0.28 0.21 0.12 -0.12 -0.4 0.097 0.21 -0.17 0.28 0.068 -0.23
-0.098 1 -0.049 -0.057 -0.2 0.045 -0.058 -0.044 0.14 0.096 -0.031 0.12 0.21 -0.28
-0.069 -0.049 1 0.048 -0.077 0.094 0.044 0.3 -0.39 -0.15 0.12 -0.18 -0.16 0.43
0.28 -0.057 0.048 1 0.12 0.18 -0.11 -0.047 0.068 0.19 -0.12 0.1 0.062 -0.14
0.21 -0.2 -0.077 0.12 1 0.013 -0.15 -0.0099 0.067 0.054 -0.004 0.071 0.099 -0.085
0.12 0.045 0.094 0.18 0.013 1 -0.084 -0.0086 0.026 0.0057 -0.06 0.14 -0.032 -0.028
-0.12 -0.058 0.044 -0.11 -0.15 -0.084 1 0.044 -0.071 -0.059 0.093 -0.072 -0.012 0.14
-0.4 -0.044 0.3 -0.047 -0.0099 -0.0086 0.044 1 -0.38 -0.34 0.39 -0.21 -0.096 0.42
0.097 0.14 -0.39 0.068 0.067 0.026 -0.071 -0.38 1 0.29 -0.26 0.12 0.21 -0.44
0.21 0.096 -0.15 0.19 0.054 0.0057 -0.059 -0.34 0.29 1 -0.58 0.22 0.21 -0.43
-0.17 -0.031 0.12 -0.12 -0.004 -0.06 0.093 0.39 -0.26 -0.58 1 -0.08 -0.1 0.35
0.28 0.12 -0.18 0.1 0.071 0.14 -0.072 -0.21 0.12 0.22 -0.08 1 0.15 -0.39
0.8
0.6
0.4
0.2
0.0
−0.2


thal
target
0.068
0.21
-0.16
0.062
0.099
-0.032
-0.012
-0.096
0.21
0.21
-0.1
0.15
1
-0.34
-0.23
-0.28
0.43
-0.14
-0.085
-0.028
0.14
0.42
-0.44
-0.43
0.35
-0.39
-0.34
1
age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target
-0.4



```
In [11]:  heart_df.hist()
```

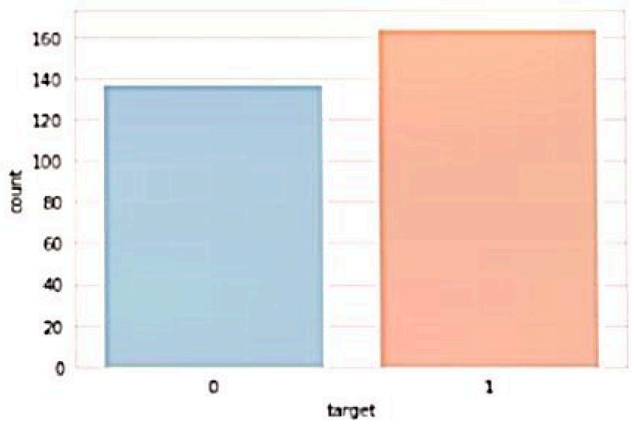
```
Out[11]:
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc82574fa90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc823442910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc8233f7f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc8233b9650>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc82336fcd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc823330390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc823366a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc8232aa050>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc8232aa110>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc8232e0850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc823258490>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc82320eb10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc8231d11d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc823187850>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc82313fed0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc8230ff590>]],
      dtype=object)
```

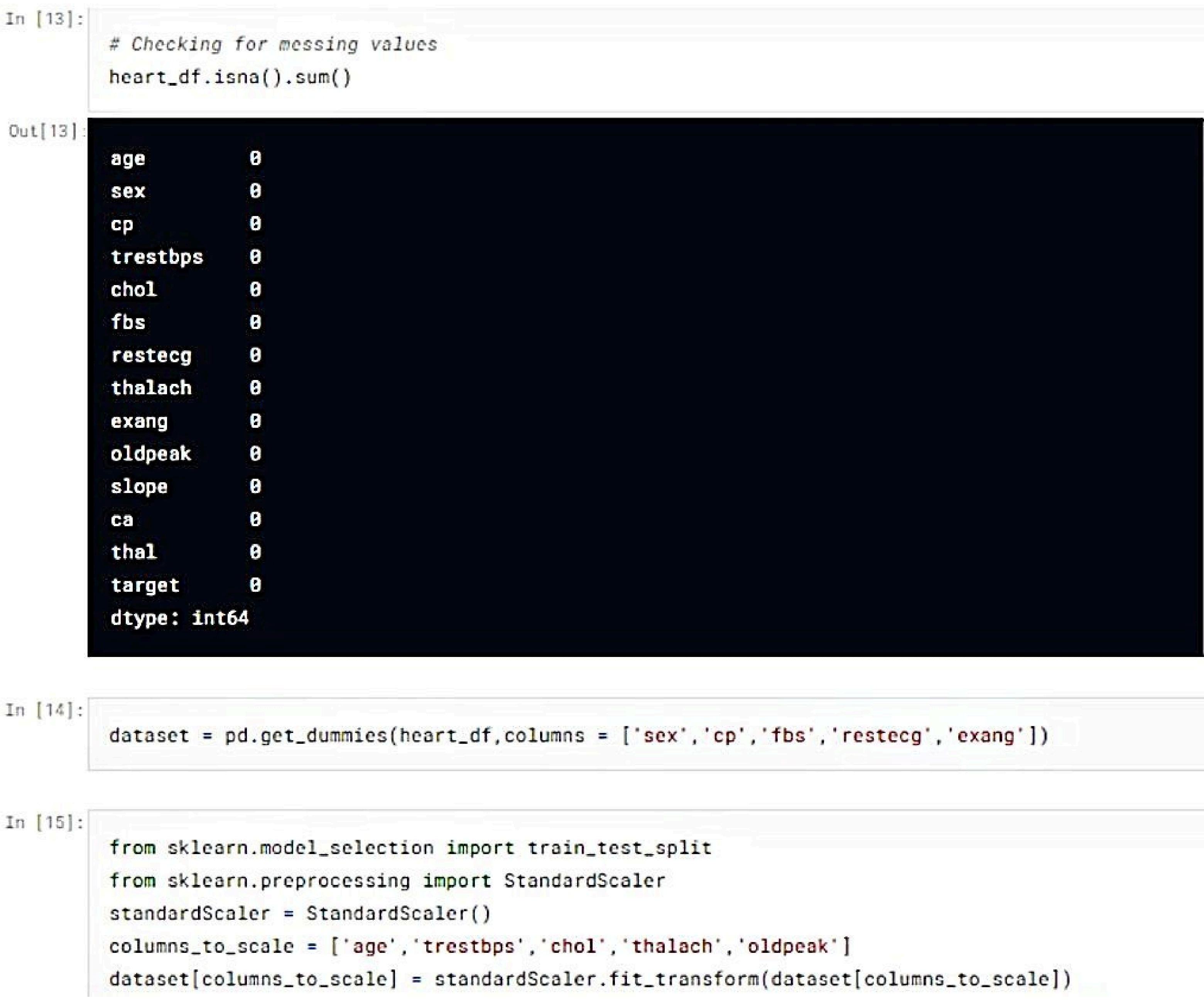

age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
target

In [10]:

```
In [12]:
sns.set_style('whitegrid')
sns.countplot(x='target',data=heart_df,palette='RdBu_r')
```

```
Out[12]:
<matplotlib.axes._subplots.AxesSubplot at 0x7fc822df2950>
```





In [13]:

```
# Checking for messing values
heart_df.isna().sum()
```

Out[13]:

```
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
```

In [14]:

```
dataset = pd.get_dummies(heart_df,columns = ['sex','cp','fbs','restecg','exang'])
```

In [15]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age','trestbps','chol','thalach','oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

In [16]:

```
dataset.head()
```

Out[16]:

	age	trestbps	chol	thalach	oldpeak	slope	ca	thal	target	sex_0	...	cp_1	cp_2	cp_3
0	0.952197	0.763956	-0.256334	0.015443	1.087338	0	0	1	1	0	...	0	0	1
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	0	0	2	1	0	...	0	1	0
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	2	0	2	1	1	...	1	0	0
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	2	0	2	1	0	...	1	0	0
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	2	0	2	1	1	...	0	0	0

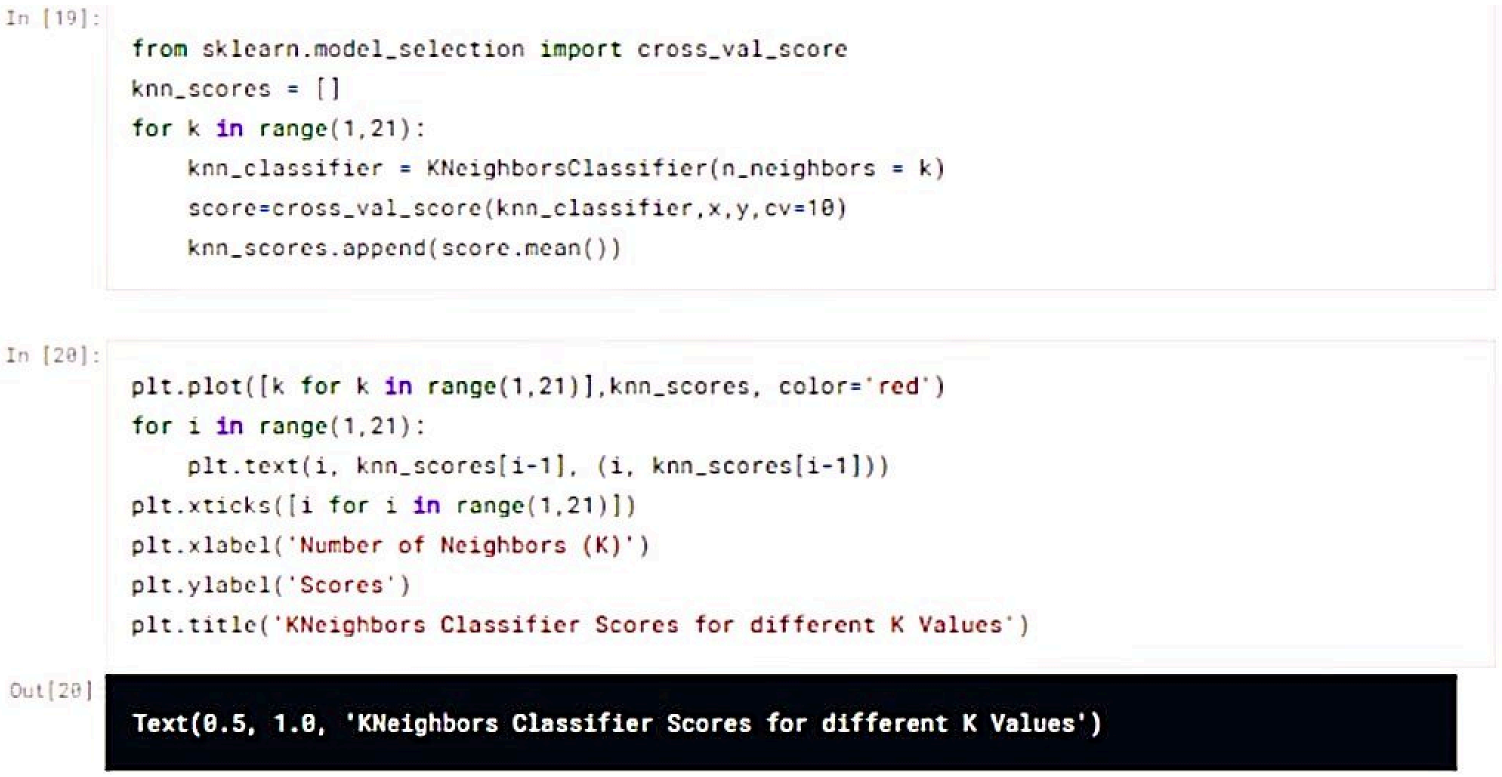
5 rows × 22 columns

In [17]:

```
y = dataset['target']
x = dataset.drop(['target'],axis = 1)
```

In [18]:

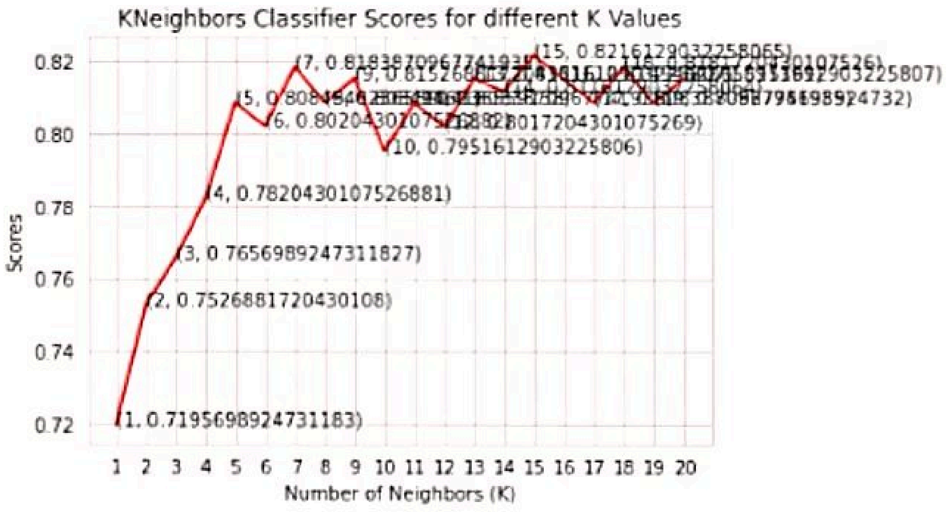
```
from sklearn.model_selection import cross_val_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,x,y,cv=10)
    knn_scores.append(score.mean())
```



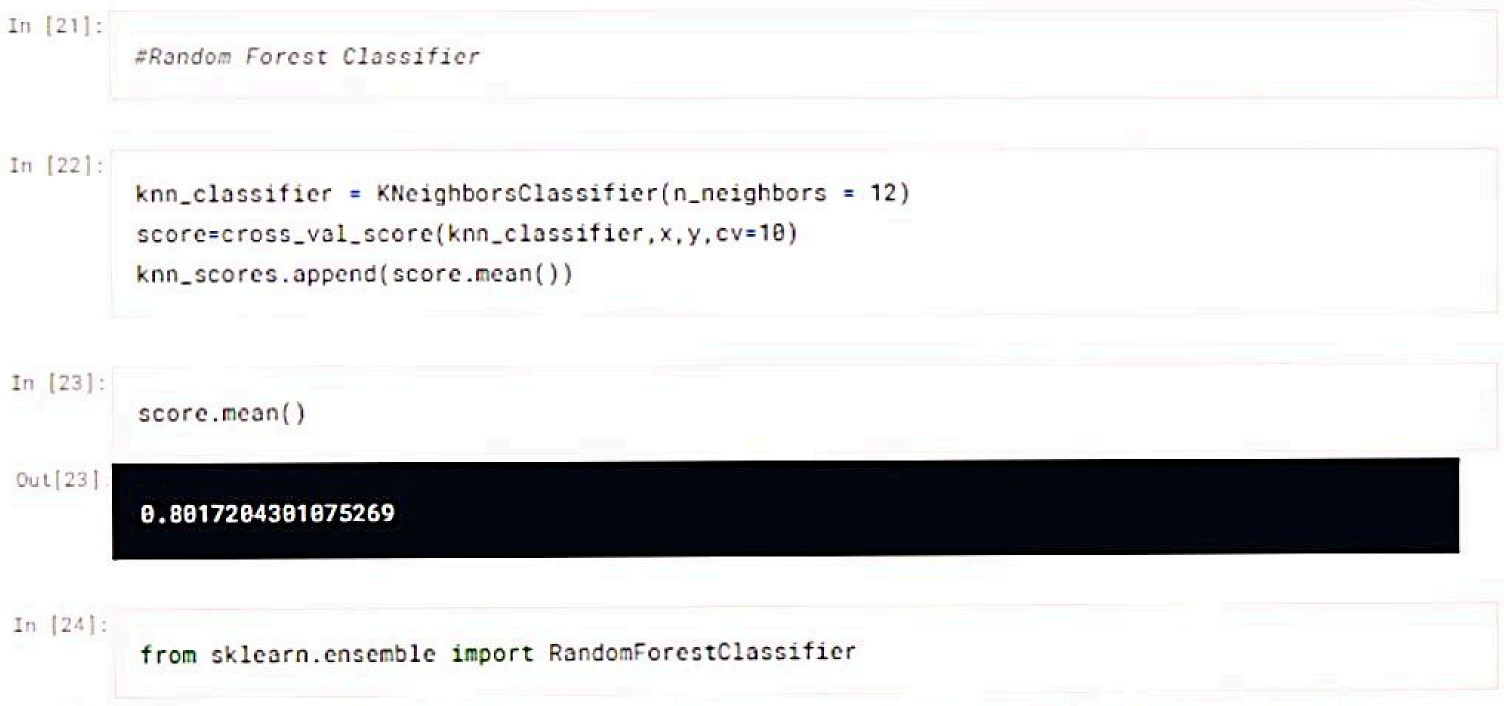
```
In [19]:
from sklearn.model_selection import cross_val_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,x,y,cv=10)
    knn_scores.append(score.mean())
```

```
In [20]:
plt.plot([k for k in range(1,21)],knn_scores, color='red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1,21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('KNeighbors Classifier Scores for different K Values')
```

```
Out[20]
Text(0.5, 1.0, 'KNeighbors Classifier Scores for different K Values')
```



KNeighbors Classifier Scores for different K Values
Scores
Number of Neighbors (K)
(1, 0.7195698924731183)
(2, 0.7526881720430108)
(3, 0.7656989247311827)
(4, 0.7820430107526881)
(5, 0.808
(6, 0.8020430107526882)
(7, 0.8183870967741935)
(9, 0.8152688
(10, 0.7951612903225806)
(15, 0.8216129032258065)
0.8017204301075269)
0.82
0.80
0.78
0.76
0.74
0.72
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20



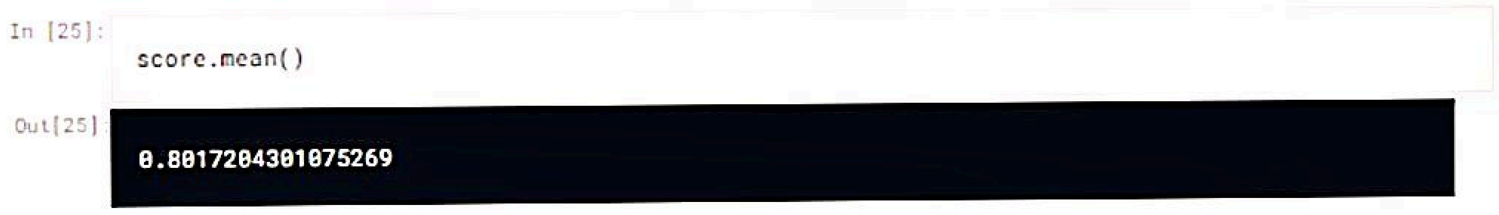
```
In [21]:  #Random Forest Classifier
```

```
In [22]:  knn_classifier = KNeighborsClassifier(n_neighbors = 12)
          score=cross_val_score(knn_classifier,x,y,cv=10)
          knn_scores.append(score.mean())
```

```
In [23]:  score.mean()

Out[23]:  0.8017204301075269
```

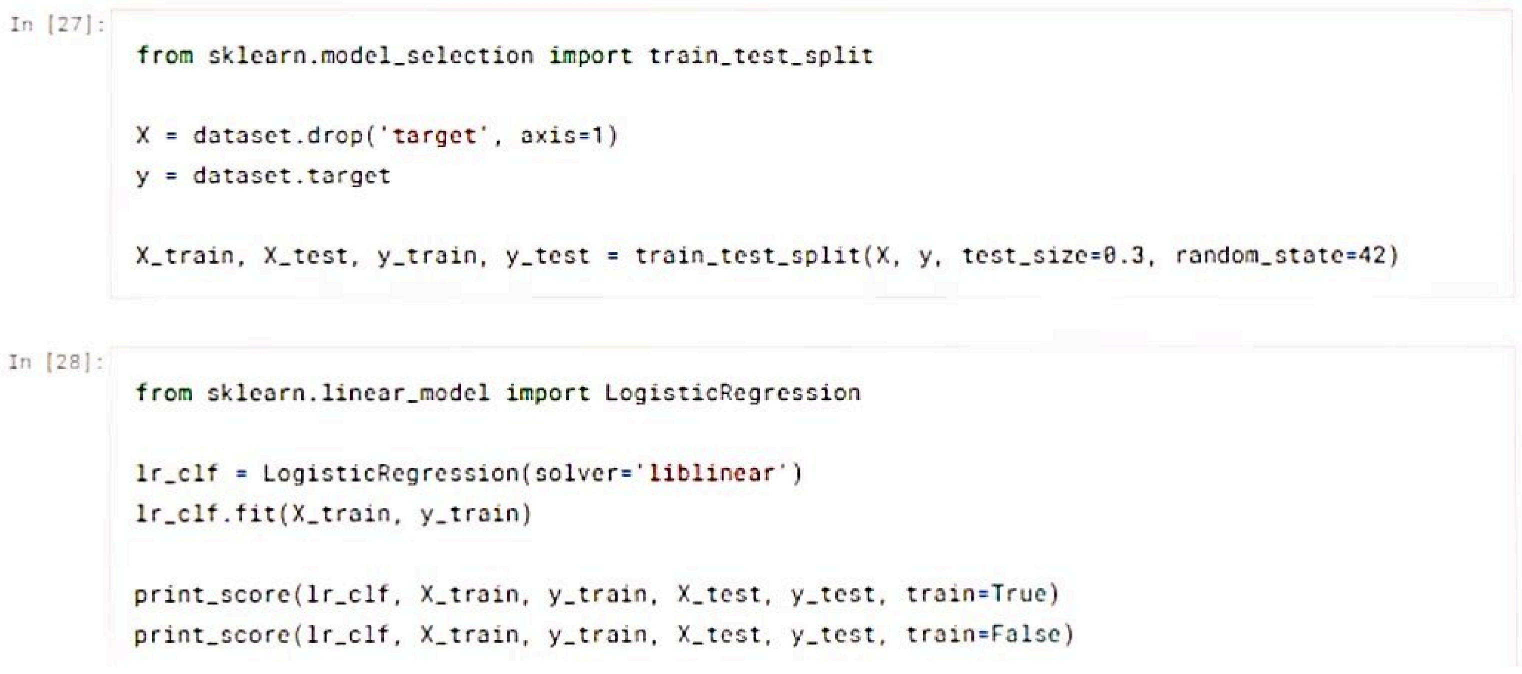
```
In [24]:  from sklearn.ensemble import RandomForestClassifier
```

randomforest_classifier= RandomForestClassifier(n_estimators=10) score=cross_val_score(randomforest_classifier,x,y,cv=10)



```
In [25]:  score.mean()

Out[25]:  0.8017204301075269
```

```
In [26]:
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

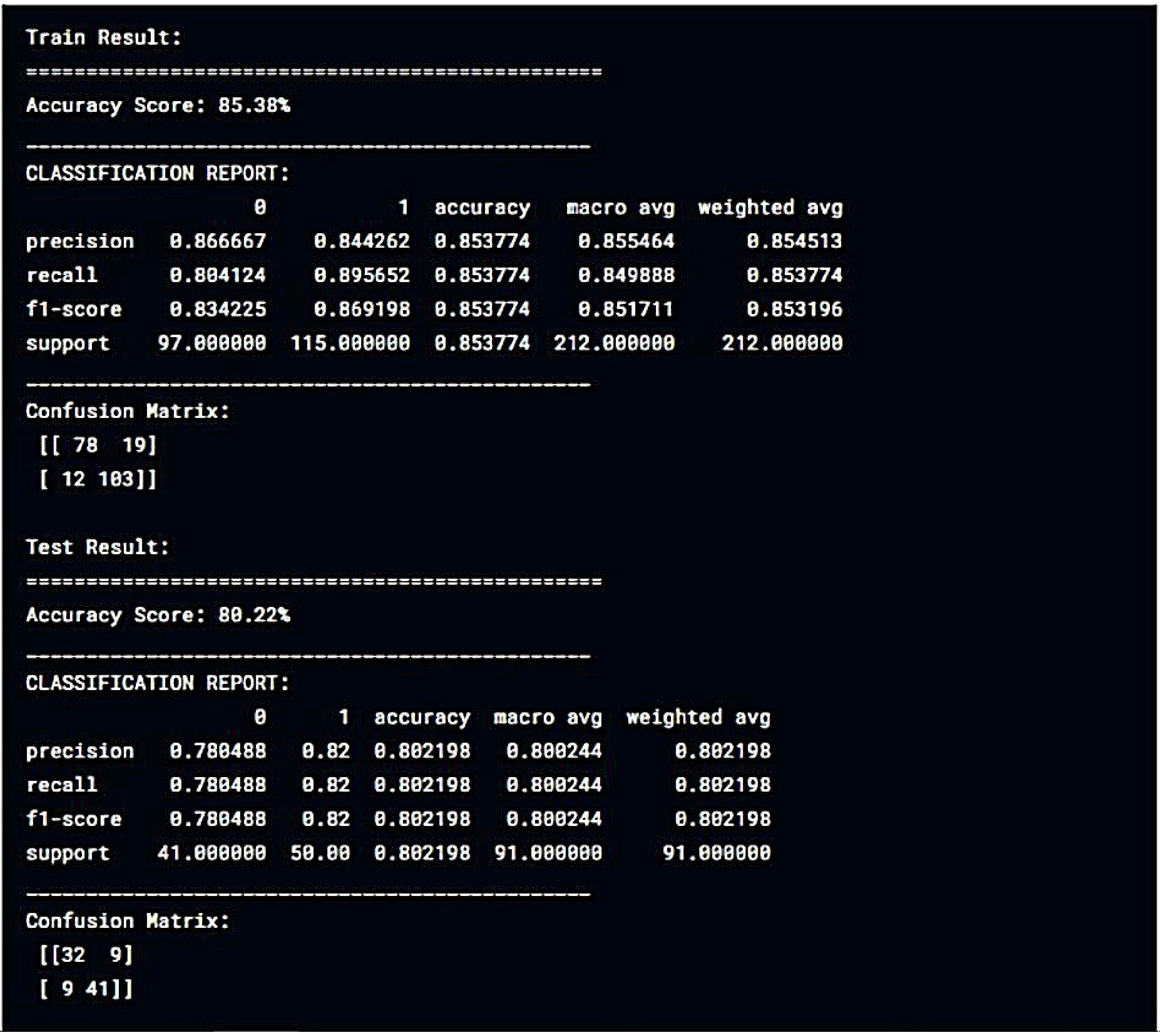


In [27]:

```
from sklearn.model_selection import train_test_split

X = dataset.drop('target', axis=1)
y = dataset.target

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

In [28]:

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(solver='liblinear')
lr_clf.fit(X_train, y_train)

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
```



```
Train Result:
================================================
Accuracy Score: 85.38%
_______________________________________________
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision   0.866667    0.844262  0.853774    0.855464      0.854513
recall      0.804124    0.895652  0.853774    0.849888      0.853774
f1-score    0.834225    0.869198  0.853774    0.851711      0.853196
support    97.000000  115.000000  0.853774  212.000000    212.000000
_______________________________________________
Confusion Matrix:
 [[ 78  19]
 [ 12 103]]

Test Result:
================================================
Accuracy Score: 80.22%
_______________________________________________
CLASSIFICATION REPORT:
                   0      1  accuracy  macro avg  weighted avg
precision   0.780488   0.82  0.802198   0.800244      0.802198
recall      0.780488   0.82  0.802198   0.800244      0.802198
f1-score    0.780488   0.82  0.802198   0.800244      0.802198
support    41.000000  50.00  0.802198  91.000000     91.000000
_______________________________________________
Confusion Matrix:
 [[32  9]
 [ 9 41]]
```

In [29]:

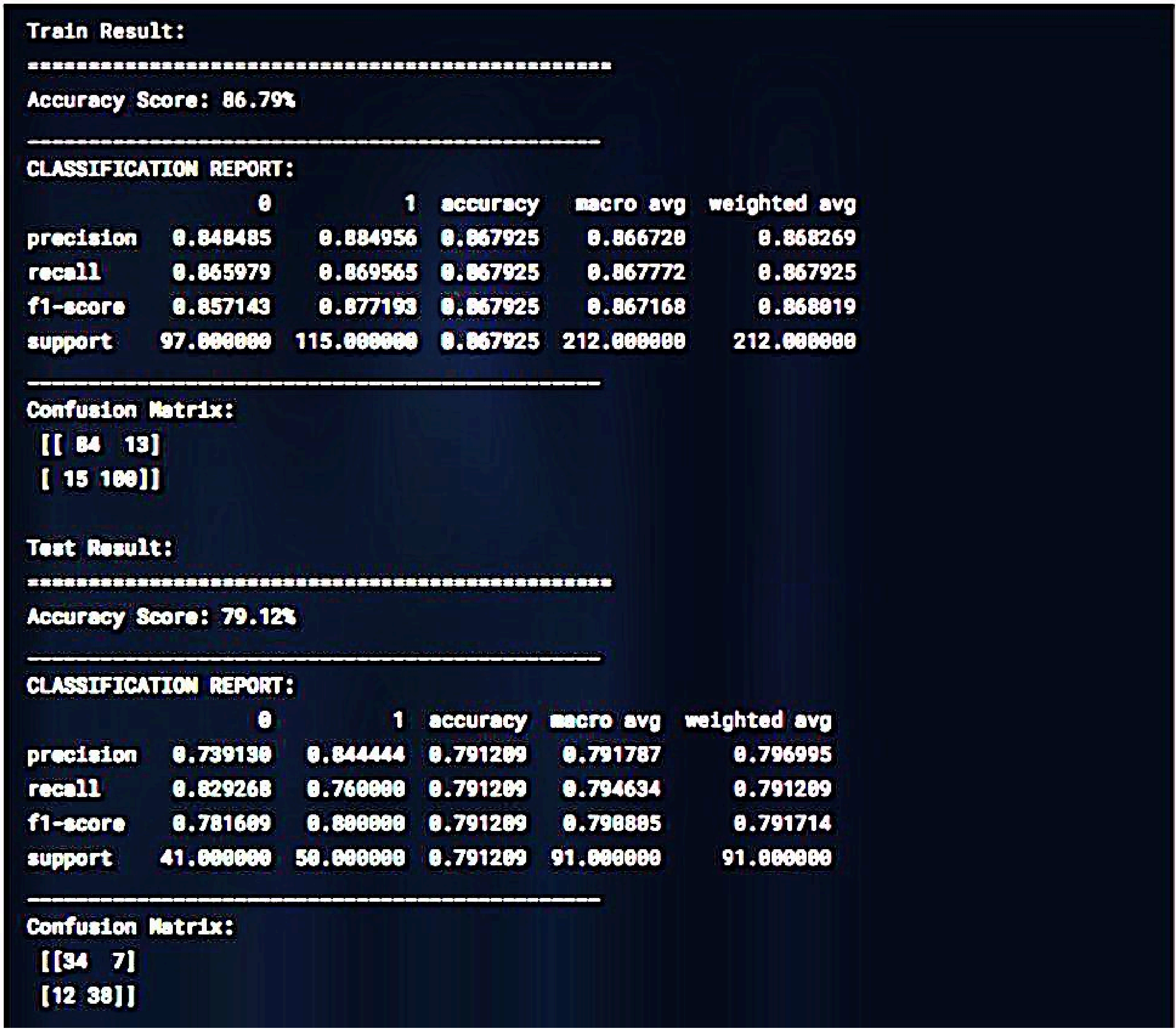
```
test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
```

Out[29]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	85.377358	80.21978

In [30]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)

print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
```



```
Train Result:
================================================
Accuracy Score: 86.79%
_______________________________________________
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision   0.848485    0.884956  0.867925    0.866720      0.868269
recall      0.865979    0.869565  0.867925    0.867772      0.867925
f1-score    0.857143    0.877193  0.867925    0.867168      0.868019
support    97.000000  115.000000  0.867925  212.000000    212.000000
_______________________________________________
Confusion Matrix:
 [[ 84  13]
 [ 15 100]]

Test Result:
================================================
Accuracy Score: 79.12%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.739130   0.844444  0.791209   0.791787      0.796995
recall      0.829268   0.760000  0.791209   0.794634      0.791209
f1-score    0.781609   0.800000  0.791209   0.790805      0.791714
support    41.000000  50.000000  0.791209  91.000000     91.000000
_______________________________________________
Confusion Matrix:
 [[34  7]
 [12 38]]
```

In [31]:

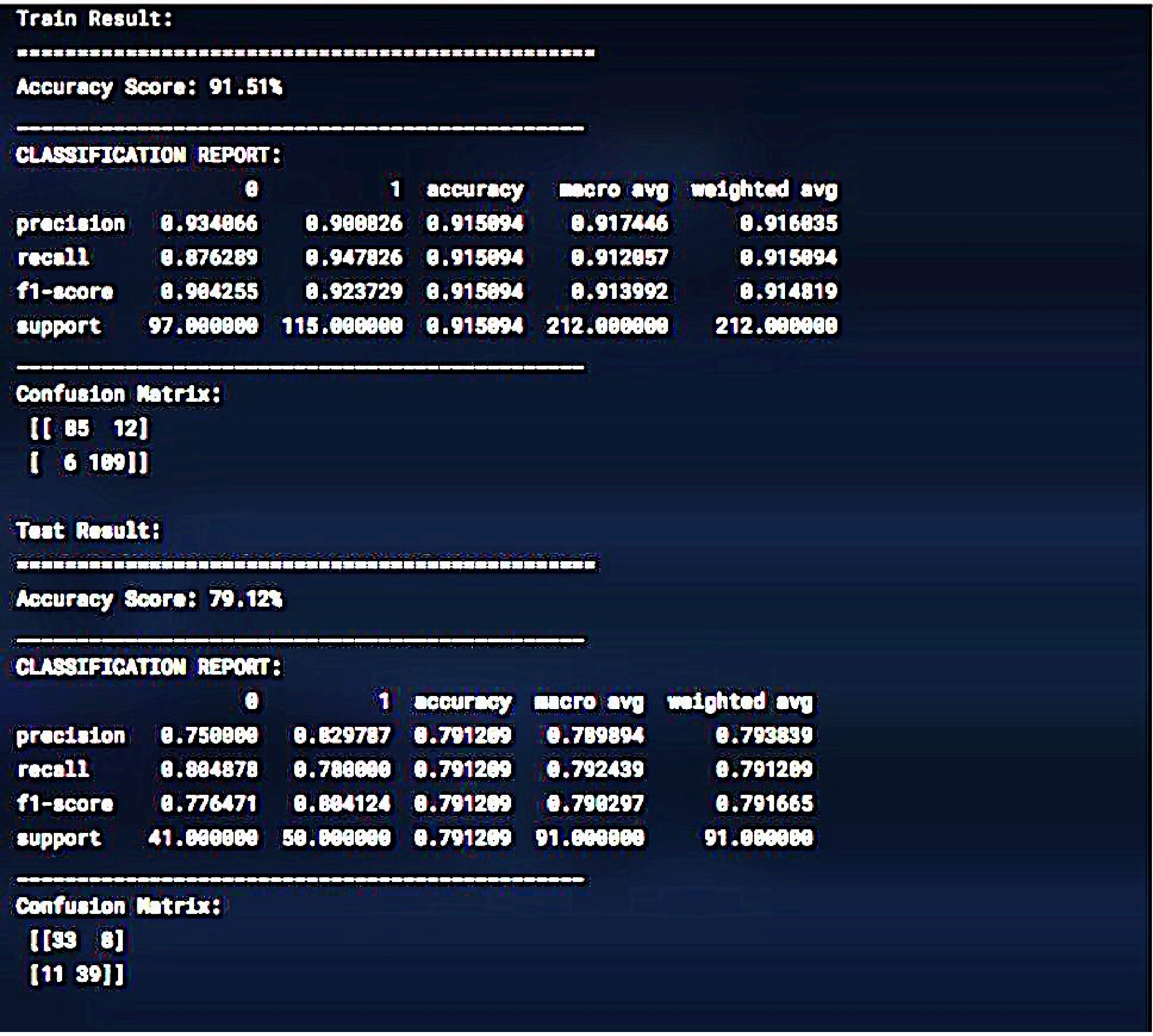
```
test_score = accuracy_score(y_test, knn_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_score]],
                            columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[31]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	85.377358	80.219780
1	K-nearest neighbors	86.792453	79.120879

In [32]:

```
from sklearn.svm import SVC


svm_clf = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)
```



```
Train Result:
=======================================================
Accuracy Score: 91.51%
_______________________________________________
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision   0.934066    0.900826  0.915094    0.917446      0.916835
recall      0.876289    0.947826  0.915094    0.912057      0.915094
f1-score    0.904255    0.923729  0.915094    0.913992      0.914819
support    97.000000  115.000000  0.915094  212.000000    212.000000
_______________________________________________
Confusion Matrix:
 [[ 85  12]
 [  6 109]]

Test Result:
=======================================================
Accuracy Score: 79.12%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.750000   0.829787  0.791209   0.789894      0.793839
recall      0.804878   0.780000  0.791209   0.792439      0.791209
f1-score    0.776471   0.804124  0.791209   0.790297      0.791665
support    41.000000  50.000000  0.791209  91.000000     91.000000
_______________________________________________
Confusion Matrix:
 [[33  8]
 [11 39]]
```

In [33]:

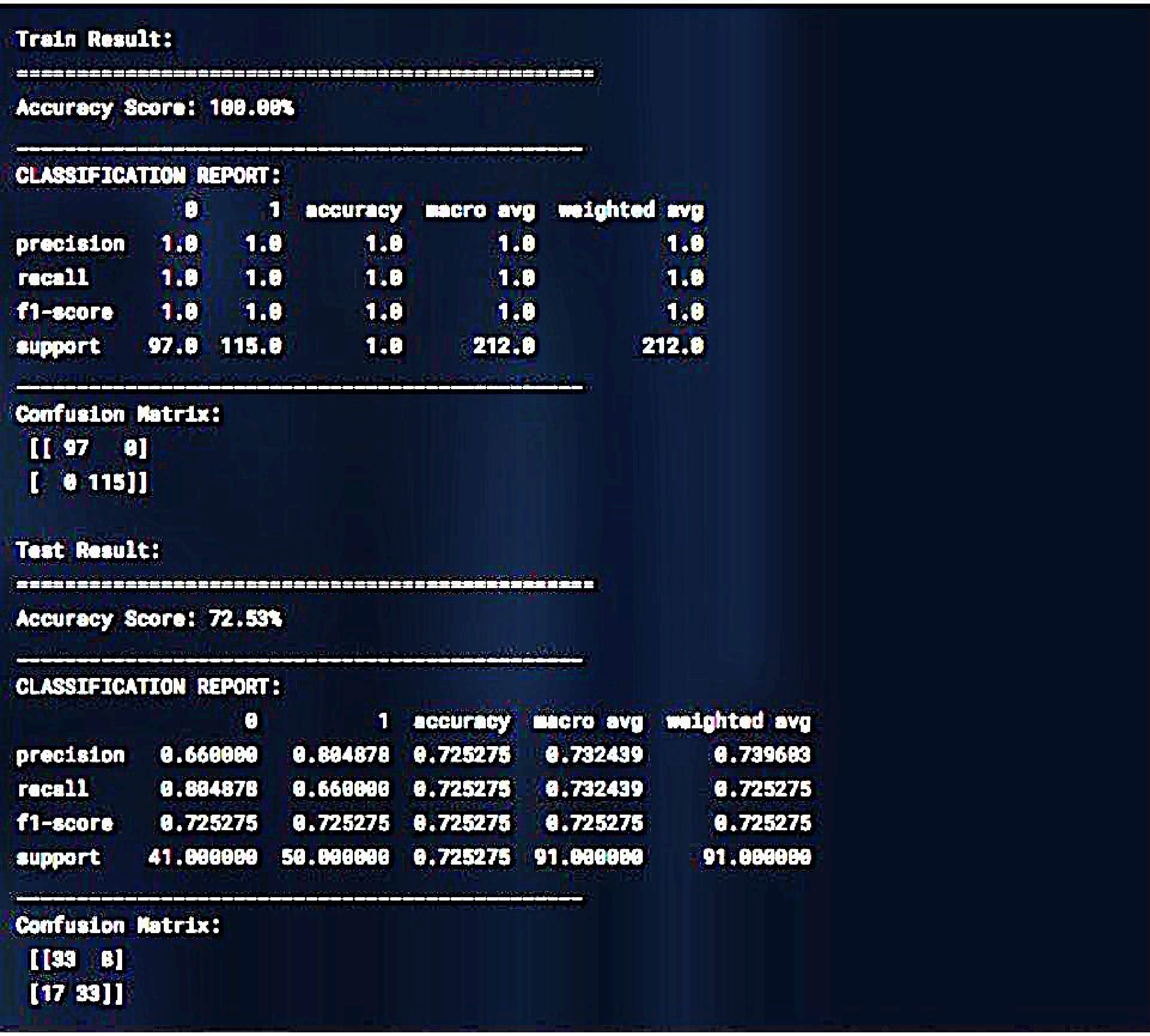
```
test_score = accuracy_score(y_test, svm_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[33]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	85.377358	80.219780
1	K-nearest neighbors	86.792453	79.120879
2	Support Vector Machine	91.509434	79.120879

In [34]:

```
from sklearn.tree import DecisionTreeClassifier


tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)

print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```



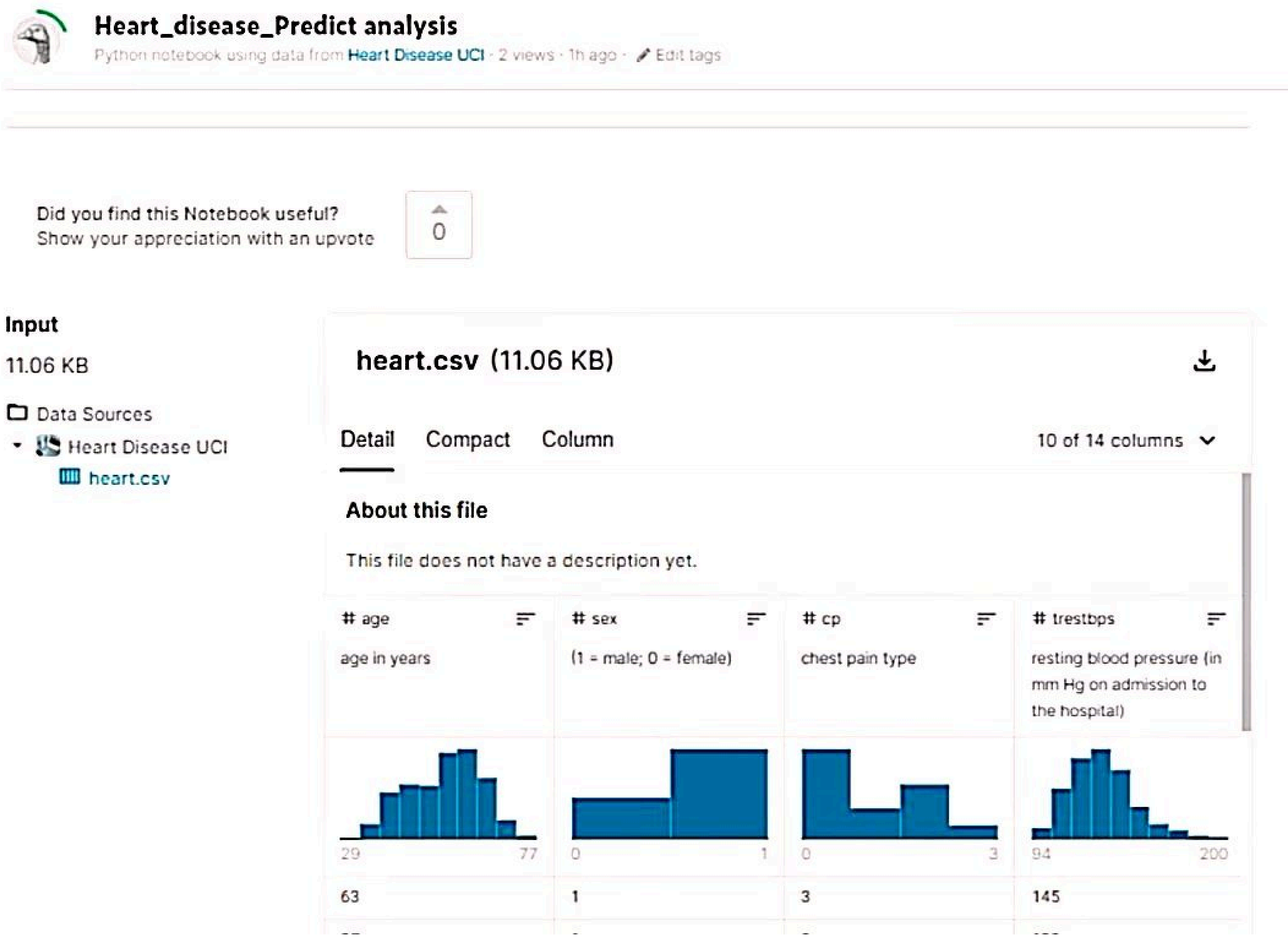
```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
              0      1  accuracy  macro avg  weighted avg
precision   1.0    1.0       1.0        1.0           1.0
recall      1.0    1.0       1.0        1.0           1.0
f1-score    1.0    1.0       1.0        1.0           1.0
support    97.0  115.0       1.0      212.0         212.0
_______________________________________________
Confusion Matrix:
 [[ 97   0]
 [  0 115]]

Test Result:
================================================
Accuracy Score: 72.53%
_______________________________________________
CLASSIFICATION REPORT:
                   0          1  accuracy  macro avg  weighted avg
precision   0.660000   0.804878  0.725275   0.732439      0.739603
recall      0.804878   0.660000  0.725275   0.732439      0.725275
f1-score    0.725275   0.725275  0.725275   0.725275      0.725275
support    41.000000  50.000000  0.725275  91.000000     91.000000
_______________________________________________
Confusion Matrix:
 [[33  8]
 [17 33]]
```

```
In [35]:
test_score = accuracy_score(y_test, tree_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[35]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	85.377358	80.219780
1	K-nearest neighbors	86.792453	79.120879
2	Support Vector Machine	91.509434	79.120879
3	Decision Tree Classifier	100.000000	72.527473

```
In [ ]:
```



Heart_disease_Predict analysis

Python notebook using data from Heart Disease UCI · 2 views · 1h ago · Edit tags

Did you find this Notebook useful?
Show your appreciation with an upvote

0

Input

11.06 KB

Data Sources
- Heart Disease UCI
  - heart.csv

heart.csv (11.06 KB)

Detail Compact Column 10 of 14 columns

About this file

This file does not have a description yet.

# age	# sex	# cp	# trestbps
age in years	(1 = male; 0 = female)	chest pain type	resting blood pressure (in mm Hg on admission to the hospital)
29 – 77	0 – 1	0 – 3	94 – 200
63	1	3	145

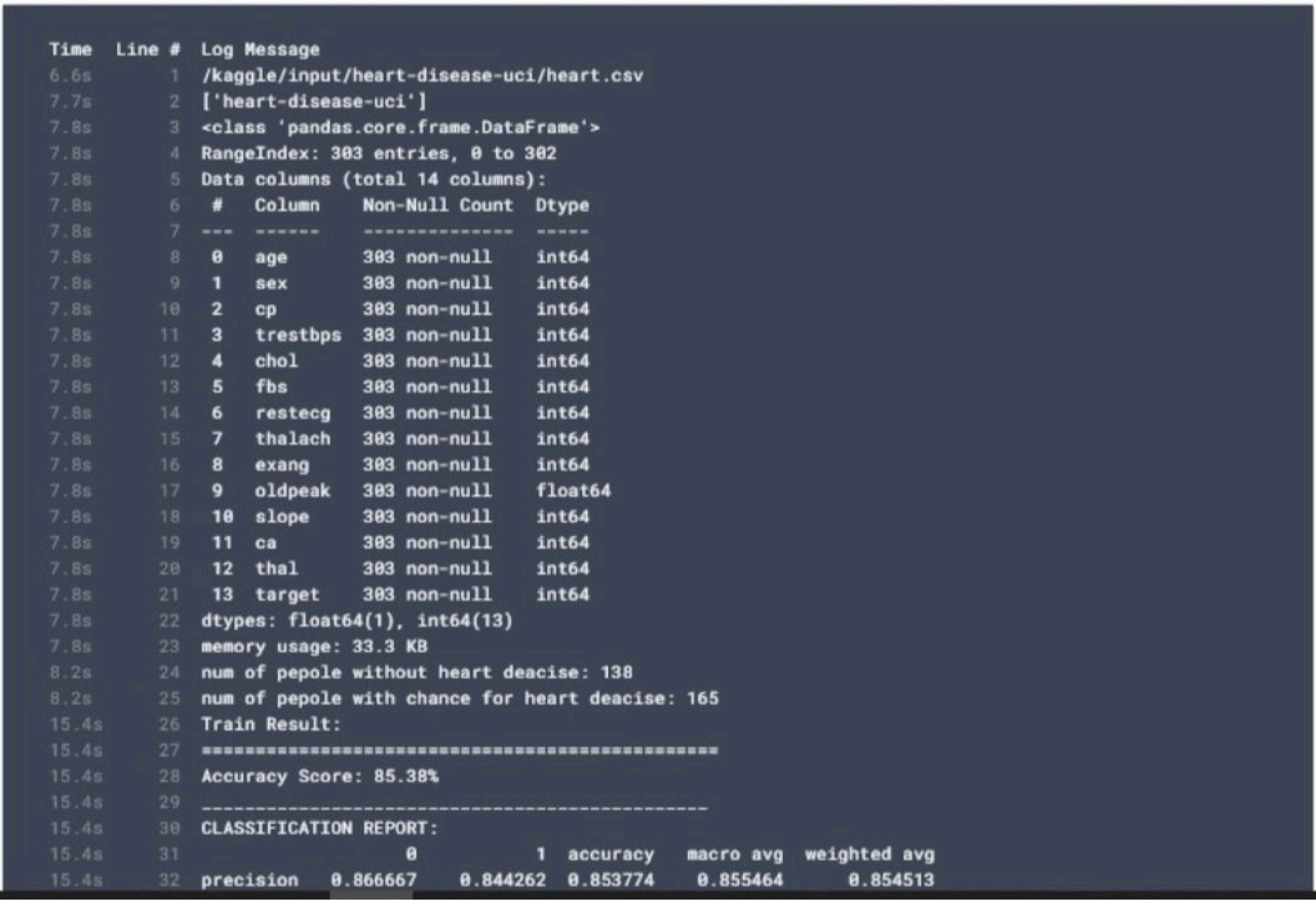
37	1	2	130
41	0	1	130
56	1	1	120
57	0	0	120
57	1	0	140
56	0	1	140
44	1	1	120
52	1	2	172
57	1	2	150

Execution Info

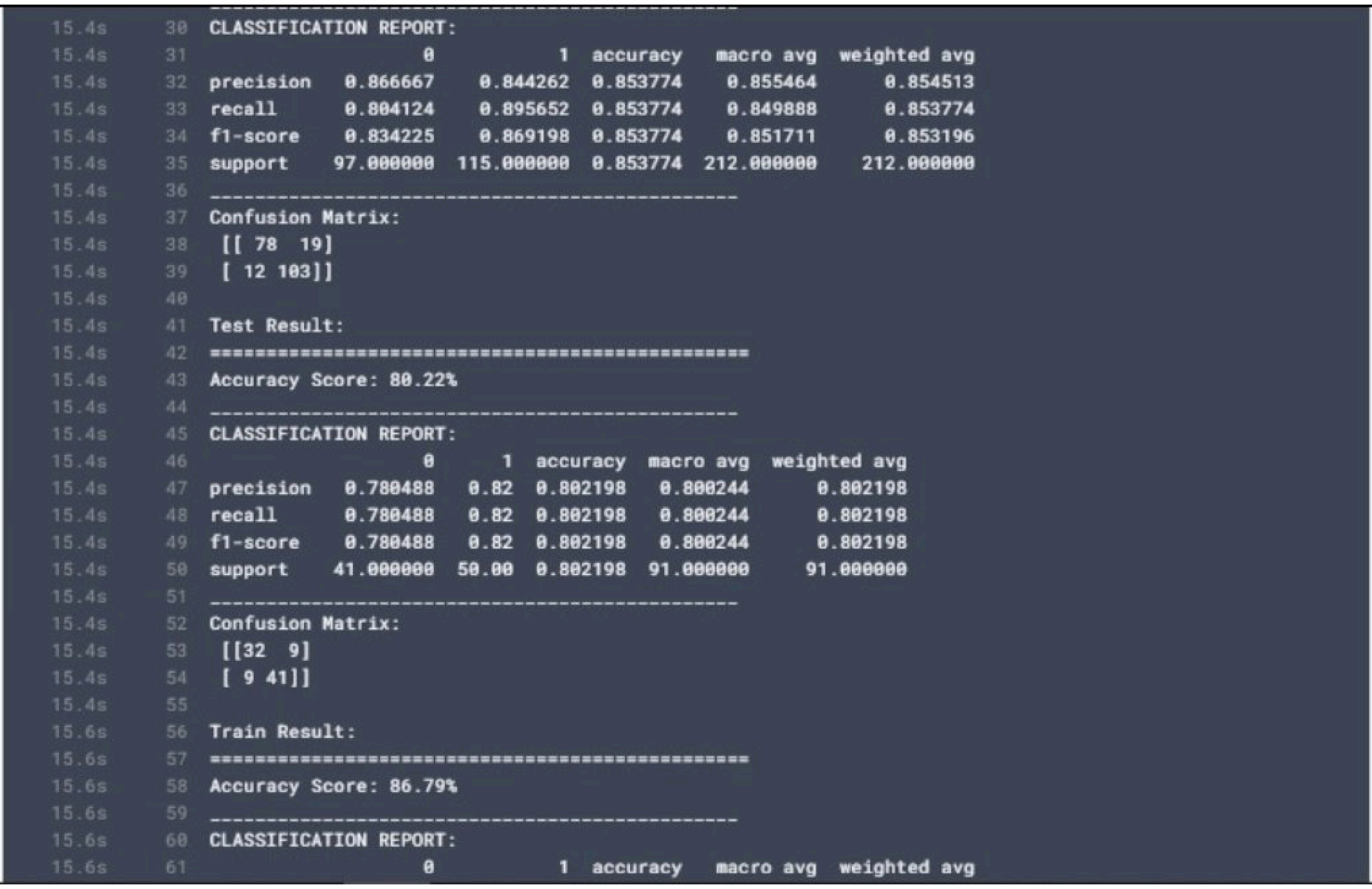
Succeeded	True	Run Time	19.1 seconds
Exit Code	0	Timeout Exceeded	False
Used All Space	False	Output Size	0
Environment	Container Image (Dockerfile)	Accelerator	None

Log

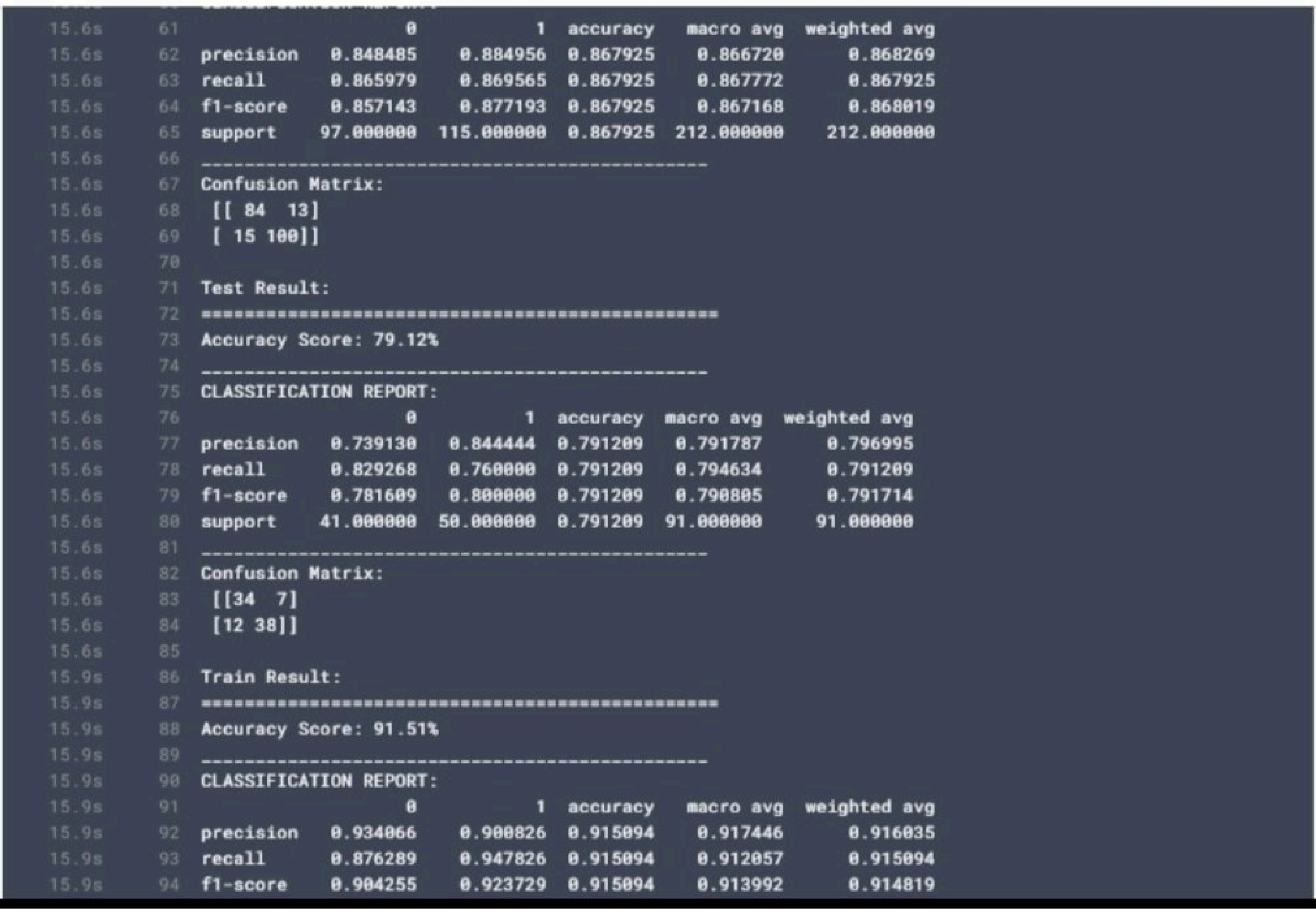
Download Log



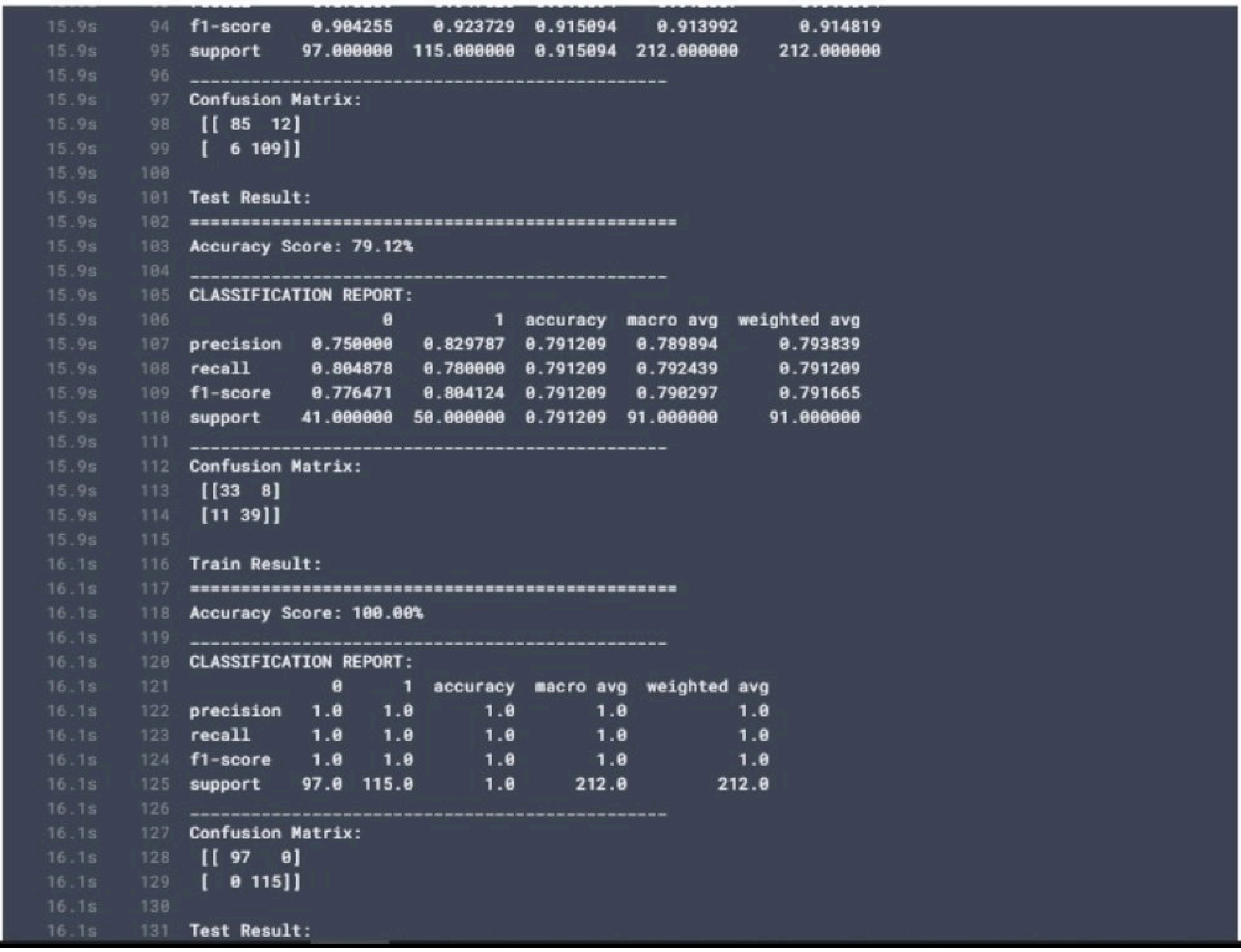
```
Time   Line #  Log Message
6.6s       1   /kaggle/input/heart-disease-uci/heart.csv
7.7s       2   ['heart-disease-uci']
7.8s       3   <class 'pandas.core.frame.DataFrame'>
7.8s       4   RangeIndex: 303 entries, 0 to 302
7.8s       5   Data columns (total 14 columns):
7.8s       6    #   Column    Non-Null Count  Dtype
7.8s       7   ---  ------    --------------  -----
7.8s       8    0   age       303 non-null    int64
7.8s       9    1   sex       303 non-null    int64
7.8s      10    2   cp        303 non-null    int64
7.8s      11    3   trestbps  303 non-null    int64
7.8s      12    4   chol      303 non-null    int64
7.8s      13    5   fbs       303 non-null    int64
7.8s      14    6   restecg   303 non-null    int64
7.8s      15    7   thalach   303 non-null    int64
7.8s      16    8   exang     303 non-null    int64
7.8s      17    9   oldpeak   303 non-null    float64
7.8s      18    10  slope     303 non-null    int64
7.8s      19    11  ca        303 non-null    int64
7.8s      20    12  thal      303 non-null    int64
7.8s      21    13  target    303 non-null    int64
7.8s      22   dtypes: float64(1), int64(13)
7.8s      23   memory usage: 33.3 KB
8.2s      24   num of pepole without heart deacise: 138
8.2s      25   num of pepole with chance for heart deacise: 165
15.4s     26   Train Result:
15.4s     27   ================================================
15.4s     28   Accuracy Score: 85.38%
15.4s     29   _______________________________________________
15.4s     30   CLASSIFICATION REPORT:
15.4s     31                      0          1  accuracy   macro avg  weighted avg
15.4s     32   precision   0.866667   0.844262  0.853774    0.855464      0.854513
```



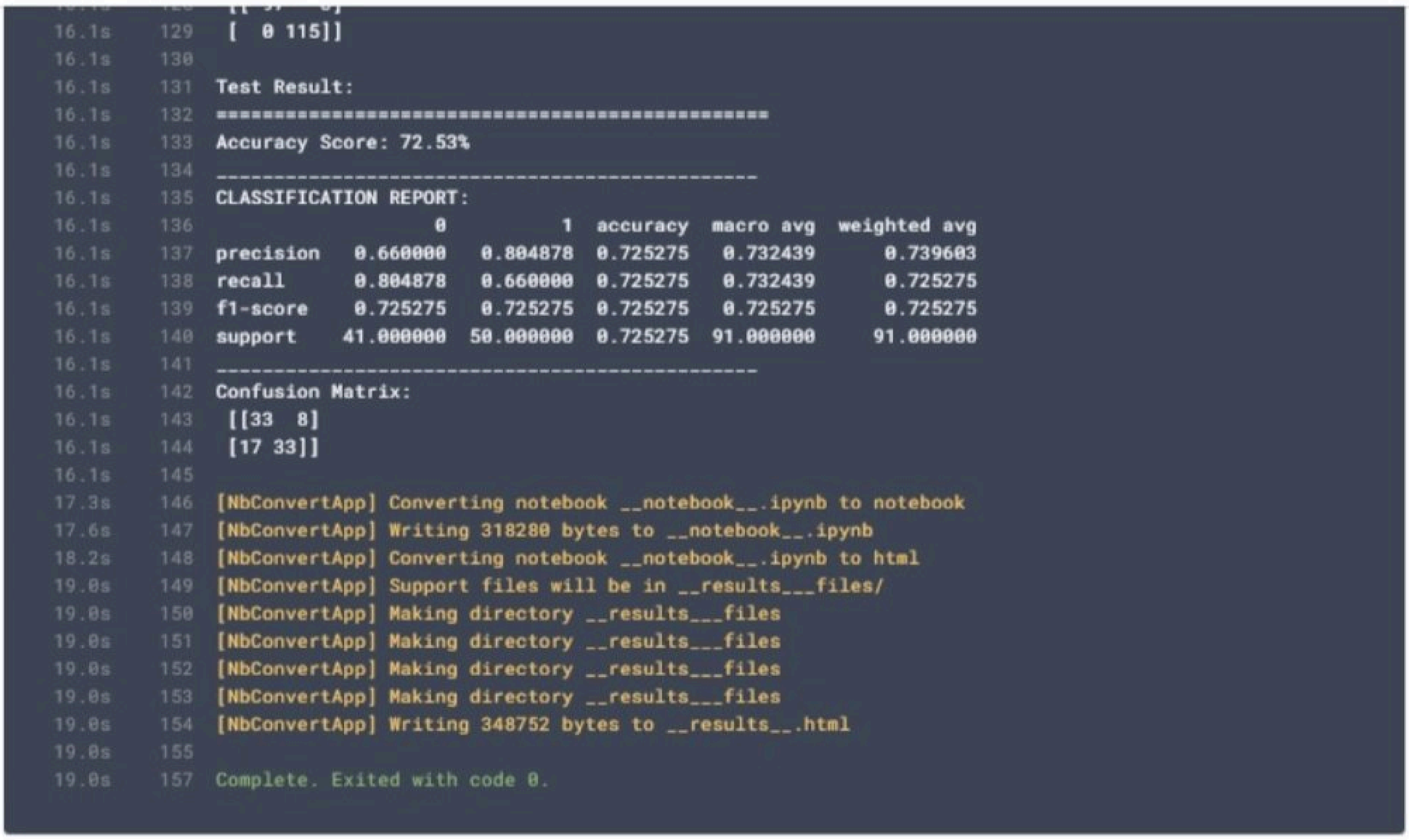
```
15.4s    ------------------------------------------------
15.4s    CLASSIFICATION REPORT:
15.4s                      0           1  accuracy   macro avg  weighted avg
15.4s    precision  0.866667    0.844262  0.853774    0.855464      0.854513
15.4s    recall     0.804124    0.895652  0.853774    0.849888      0.853774
15.4s    f1-score   0.834225    0.869198  0.853774    0.851711      0.853196
15.4s    support   97.000000  115.000000  0.853774  212.000000    212.000000
15.4s    ------------------------------------------------
15.4s    Confusion Matrix:
15.4s     [[ 78  19]
15.4s     [ 12 103]]
15.4s
15.4s    Test Result:
15.4s    ================================================
15.4s    Accuracy Score: 80.22%
15.4s    ------------------------------------------------
15.4s    CLASSIFICATION REPORT:
15.4s                      0     1  accuracy  macro avg  weighted avg
15.4s    precision  0.780488  0.82  0.802198   0.800244      0.802198
15.4s    recall     0.780488  0.82  0.802198   0.800244      0.802198
15.4s    f1-score   0.780488  0.82  0.802198   0.800244      0.802198
15.4s    support   41.000000 50.00  0.802198  91.000000     91.000000
15.4s    ------------------------------------------------
15.4s    Confusion Matrix:
15.4s     [[32  9]
15.4s     [ 9 41]]
15.4s
15.6s    Train Result:
15.6s    ================================================
15.6s    Accuracy Score: 86.79%
15.6s    ------------------------------------------------
15.6s    CLASSIFICATION REPORT:
15.6s                      0           1  accuracy   macro avg  weighted avg
```



```
15.6s                    0          1  accuracy   macro avg  weighted avg
15.6s  precision  0.848485   0.884956  0.867925    0.866720      0.868269
15.6s  recall     0.865979   0.869565  0.867925    0.867772      0.867925
15.6s  f1-score   0.857143   0.877193  0.867925    0.867168      0.868019
15.6s  support   97.000000 115.000000  0.867925  212.000000    212.000000
15.6s  -----------------------------------------------
15.6s  Confusion Matrix:
15.6s   [[ 84  13]
15.6s   [ 15 100]]
15.6s
15.6s  Test Result:
15.6s  ================================================
15.6s  Accuracy Score: 79.12%
15.6s  -----------------------------------------------
15.6s  CLASSIFICATION REPORT:
15.6s                    0          1  accuracy  macro avg  weighted avg
15.6s  precision  0.739130   0.844444  0.791209   0.791787      0.796995
15.6s  recall     0.829268   0.760000  0.791209   0.794634      0.791209
15.6s  f1-score   0.781609   0.800000  0.791209   0.790805      0.791714
15.6s  support   41.000000  50.000000  0.791209  91.000000     91.000000
15.6s  -----------------------------------------------
15.6s  Confusion Matrix:
15.6s   [[34  7]
15.6s   [12 38]]
15.6s
15.9s  Train Result:
15.9s  ================================================
15.9s  Accuracy Score: 91.51%
15.9s  -----------------------------------------------
15.9s  CLASSIFICATION REPORT:
15.9s                    0          1  accuracy   macro avg  weighted avg
15.9s  precision  0.934066   0.900826  0.915094    0.917446      0.916035
15.9s  recall     0.876289   0.947826  0.915094    0.912057      0.915094
15.9s  f1-score   0.904255   0.923729  0.915094    0.913992      0.914819
```



```
15.9s   f1-score    0.904255   0.923729  0.915094   0.913992      0.914819
15.9s   support    97.000000  115.000000  0.915094  212.000000    212.000000
15.9s   -----------------------------------------------
15.9s   Confusion Matrix:
15.9s    [[ 85  12]
15.9s    [  6 109]]
15.9s
15.9s   Test Result:
15.9s   ================================================
15.9s   Accuracy Score: 79.12%
15.9s   -----------------------------------------------
15.9s   CLASSIFICATION REPORT:
15.9s                      0          1  accuracy  macro avg  weighted avg
15.9s   precision   0.750000   0.829787  0.791209   0.789894      0.793839
15.9s   recall      0.804878   0.780000  0.791209   0.792439      0.791209
15.9s   f1-score    0.776471   0.804124  0.791209   0.790297      0.791665
15.9s   support    41.000000  50.000000  0.791209  91.000000     91.000000
15.9s   -----------------------------------------------
15.9s   Confusion Matrix:
15.9s    [[33  8]
15.9s    [11 39]]
15.9s
16.1s   Train Result:
16.1s   ================================================
16.1s   Accuracy Score: 100.00%
16.1s   -----------------------------------------------
16.1s   CLASSIFICATION REPORT:
16.1s                 0      1  accuracy  macro avg  weighted avg
16.1s   precision   1.0    1.0       1.0        1.0           1.0
16.1s   recall      1.0    1.0       1.0        1.0           1.0
16.1s   f1-score    1.0    1.0       1.0        1.0           1.0
16.1s   support    97.0  115.0       1.0      212.0         212.0
16.1s   -----------------------------------------------
16.1s   Confusion Matrix:
16.1s    [[ 97   0]
16.1s    [  0 115]]
16.1s
16.1s   Test Result:
```



```
16.1s   [  0 115]]
16.1s
16.1s   Test Result:
16.1s   ================================================
16.1s   Accuracy Score: 72.53%
16.1s   _______________________________________________
16.1s   CLASSIFICATION REPORT:
16.1s                      0          1  accuracy  macro avg  weighted avg
16.1s   precision   0.660000   0.804878  0.725275   0.732439      0.739603
16.1s   recall      0.804878   0.660000  0.725275   0.732439      0.725275
16.1s   f1-score    0.725275   0.725275  0.725275   0.725275      0.725275
16.1s   support    41.000000  50.000000  0.725275  91.000000     91.000000
16.1s   _______________________________________________
16.1s   Confusion Matrix:
16.1s    [[33  8]
16.1s    [17 33]]
16.1s
17.3s   [NbConvertApp] Converting notebook __notebook__.ipynb to notebook
17.6s   [NbConvertApp] Writing 318280 bytes to __notebook__.ipynb
18.2s   [NbConvertApp] Converting notebook __notebook__.ipynb to html
19.0s   [NbConvertApp] Support files will be in __results___files/
19.0s   [NbConvertApp] Making directory __results___files
19.0s   [NbConvertApp] Making directory __results___files
19.0s   [NbConvertApp] Making directory __results___files
19.0s   [NbConvertApp] Making directory __results___files
19.0s   [NbConvertApp] Writing 348752 bytes to __results__.html
19.0s
19.0s   Complete. Exited with code 0.
```