Logistic_Regression

December 30, 2023

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter("ignore")
```

[3]:
```
df=pd.read_csv("file:///Users/chaitanya_offical/Desktop/SRK-Dataset/diabetes.
↪csv")
```

[4]:
```
df.head()
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
```

```
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[7]:
```
df["Outcome"].unique()
```

[7]:
```
array([1, 0])
```

[8]:
```
df["Outcome"].value_counts()
```

[8]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[10]:
```
continous=["Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction
discrete_count=["Pregnancies"]
discrete_categorical=["Outcome"]
```

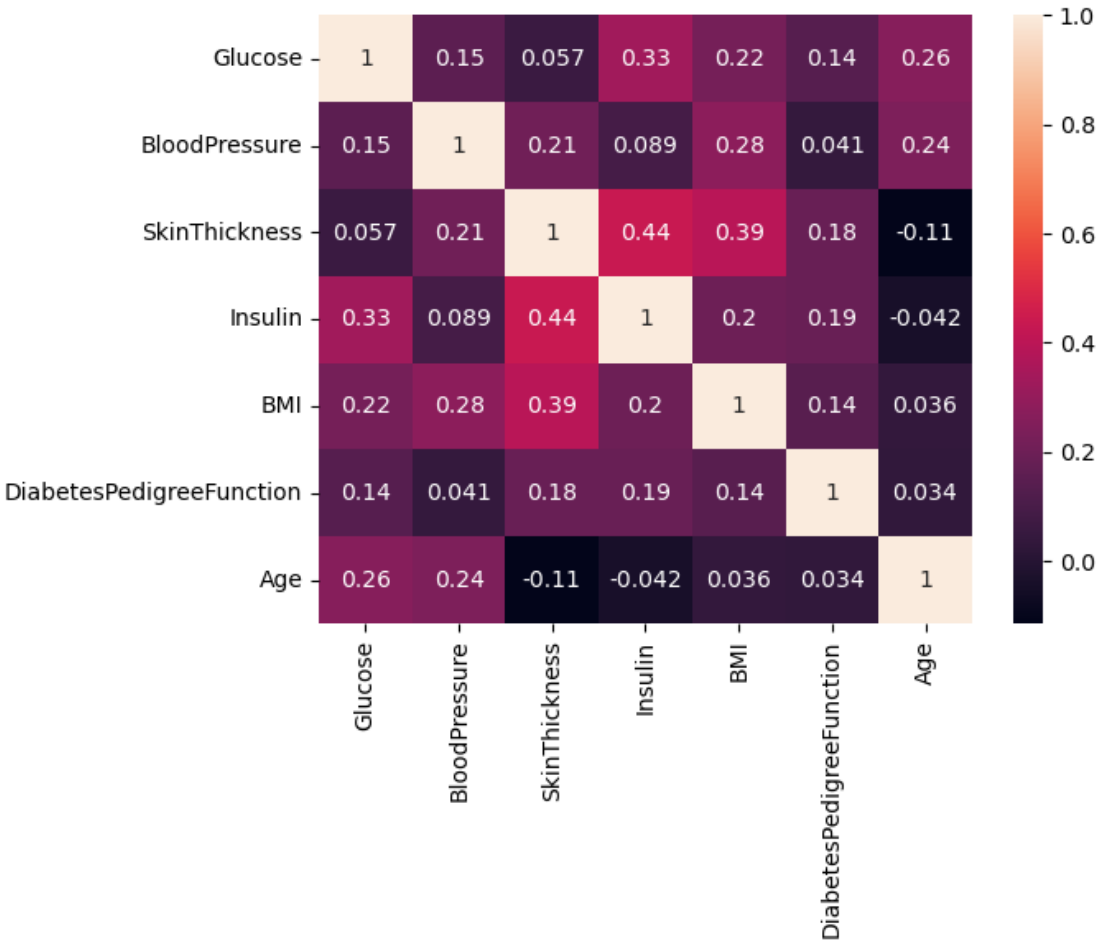
[11]:
```
df[continous].describe()
```

[11]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	120.894531	69.105469	20.536458	79.799479	31.992578	
std	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	140.250000	80.000000	32.000000	127.250000	36.600000	
max	199.000000	122.000000	99.000000	846.000000	67.100000	

	DiabetesPedigreeFunction	Age
count	768.000000	768.000000
mean	0.471876	33.240885
std	0.331329	11.760232
min	0.078000	21.000000
25%	0.243750	24.000000
50%	0.372500	29.000000
75%	0.626250	41.000000
max	2.420000	81.000000

[12]:
```
sns.heatmap(df[continous].corr(),annot=True)
plt.show()
```



[13]:
```
df.isnull().sum()
```

[13]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[14]:
```
X=df.drop('Outcome',axis=1)
y=df['Outcome']
```

[15]:
```
#Train/Test Split

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪2,random_state=16)
```

[18]:
```
from sklearn.preprocessing import StandardScaler

# Create an instance of StandardScaler
sc = StandardScaler()

# Apply StandardScaler to the continuous features in the training set
X_train.iloc[:, 1:8] = sc.fit_transform(X_train.iloc[:, 1:8])

# Apply the same StandardScaler to the continuous features in the test set
X_test.iloc[:, 1:8] = sc.transform(X_test.iloc[:, 1:8])
```

[20]:
```
#Modelling

from sklearn.linear_model import LogisticRegression
log_reg=LogisticRegression()
log_reg.fit(X_train,y_train)
```

[20]:
```
LogisticRegression()
```

[21]:
```
# Assuming you have already trained a logistic regression model (log_reg) on␣
 ↪your training data

# Make predictions on the training set
ypred_train = log_reg.predict(X_train)

# Train accuracy
from sklearn.metrics import accuracy_score  # Corrected the function name
print("Train Accuracy:", accuracy_score(y_train, ypred_train))  # Corrected the␣
 ↪function name

# Cross-validation score
from sklearn.model_selection import cross_val_score
print("CV Score:", cross_val_score(log_reg, X_train, y_train, cv=5,␣
 ↪scoring='accuracy'))  # Specify the scoring metric
```

```
Train Accuracy: 0.7719869706840391
CV Score: [0.73170732 0.67479675 0.79674797 0.78861789 0.7704918 ]
```

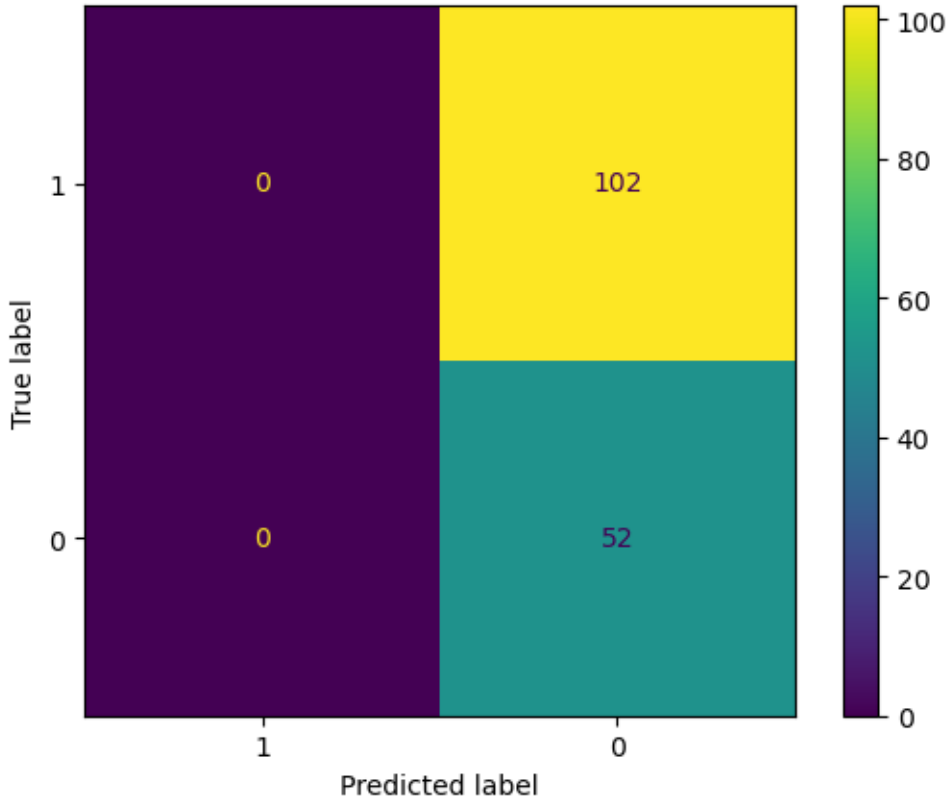
[22]:
```
ypred_test=log_reg.predict(X_test)
print("Test Accuarcy",accuracy_score(y_test,ypred_test))
```

```
Test Accuarcy 0.33766233766233766
```

[24]:
```
#confusion matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,ypred_test)
cm
```

[24]:
```
array([[  0, 102],
       [  0,  52]])
```

[28]:
```
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming you have the confusion matrix (cm) and display labels
display_labels = df["Outcome"].unique()

# Create a ConfusionMatrixDisplay object
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,␣
 ↪display_labels=display_labels)

# Plot the confusion matrix
cm_display.plot()

# Display the plot
plt.show()
```

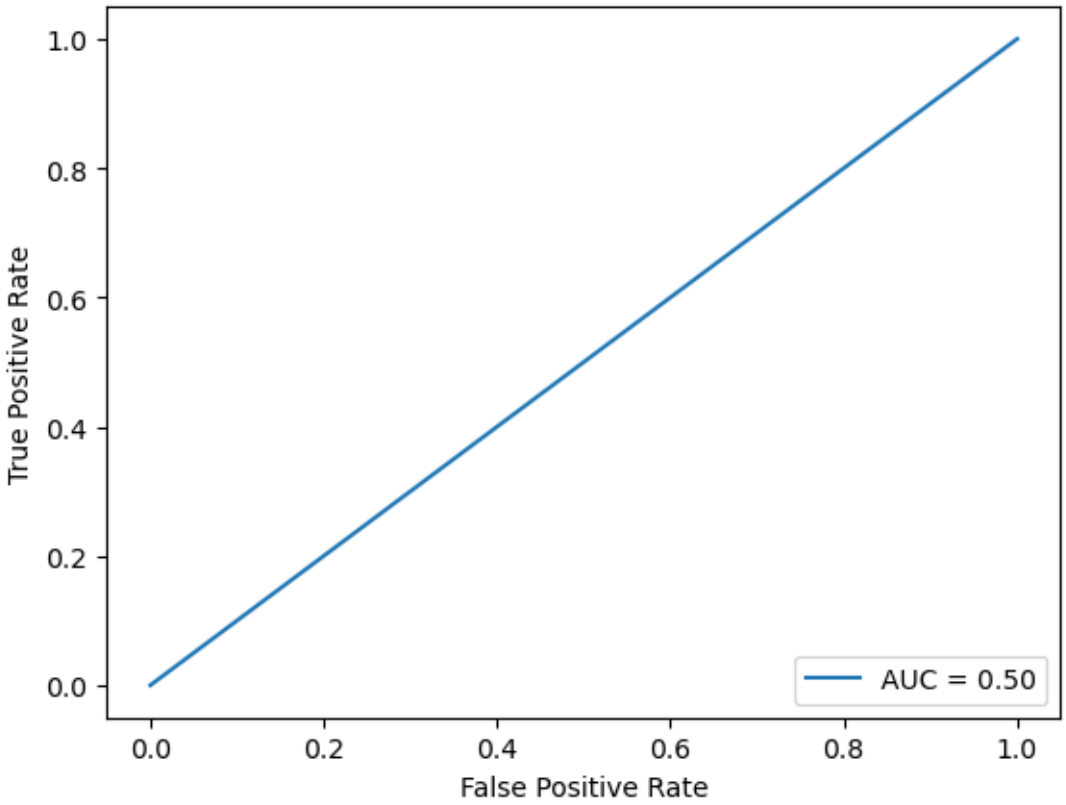


[30]:
```
from sklearn.metrics import classification_report

# Assuming y_test and ypred_test are your true and predicted labels
print(classification_report(y_test, ypred_test))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       102
           1       0.34      1.00      0.50        52

    accuracy                           0.34       154
   macro avg       0.17      0.50      0.25       154
weighted avg       0.11      0.34      0.17       154
```

[31]:
```
from sklearn.metrics import roc_curve,auc,RocCurveDisplay

fpr,tpr,thresholds=roc_curve(y_test,ypred_test)
roc_auc=auc(fpr,tpr)
RocCurveDisplay(fpr=fpr,tpr=tpr,roc_auc=roc_auc).plot()
```

[31]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fb04b00ae20>
```



[32]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test,ypred_test)
```

[32]: 0.5

[ ]: