*IMPORTING ALL LIBRARIES

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt
import seaborn as sns

from mlxtend.plotting import plot_confusion_matrix

from sklearn.metrics import accuracy_score, confusion_matrix ,recall_score, precision_score,f1_score, classification_report
```

PROBLEM STATEMENT To Predict Diabetes using Logistic Regression Classifier and to Plot Logistic Regression Classifier

IMPORTING OF DATA

```
data = pd.read_csv("diabetes.xls")
data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Next steps: Generate code with `data` View recommended plots

```
data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Next steps: Generate code with `data` View recommended plots

```
data.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

*FINDING COLUMNS *

```
data.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
x = data.drop('Outcome', axis=1)
y = data['Outcome']
```

split data to training and testing data

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=43)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((537, 8), (231, 8), (537,), (231,))
```

```
m = LogisticRegression(random_state=42, max_iter=1000)
m.fit(x_train,y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

```
m.intercept_
```

```
array([-7.95981967])
```

```
m.coef_
```

```
array([[ 1.19254495e-01,  3.26733172e-02, -1.62791027e-02,
        -2.20458814e-04, -1.10686670e-03,  8.78589210e-02,
         1.42089430e+00,  1.27159131e-02]])
```

```
m.predict_proba(x_train)[0:5]
```

```
array([[0.65066383, 0.34933617],
       [0.36881305, 0.63118695],
       [0.81425856, 0.18574144],
       [0.88689068, 0.11310932],
       [0.9538669 , 0.0461331 ]])
```

```
m.predict(x_train)[0:5]
```

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

```
y_pred = m.predict(x_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

MODEL EVALUATION

```
confusion_matrix(y_test,y_pred)
```

```
array([[142,  10],
       [ 35,  44]])
```

```
accuracy_score(y_test,y_pred)
```

```
0.8051948051948052
```

Creating model evaluation function

+ Code + Text

```
def model_eval(m, x,y):
    y_pred = m.predict(x)
    con_mat=confusion_matrix(y,y_pred)
    print(f"Confussion Matrix = \n{con_mat}")

    ac =accuracy_score(y,y_pred)
    print(f"Accuracy Score ={ac}")

    pv = precision_val=precision_score(y,y)
    print(f"precision value ={pv}")

    rv = recall_val= recall_score(y,y_pred)
    print(f"Recall Value ={rv}")

    f1_val = f1_score(y,y_pred)
    print(f"F1 Value ={f1_val}")

    cls_repo=classification_report(y,y_pred)
    print(f"Clasification Report = \n{cls_repo}")

    return "Model Evaluation Success "
```

"Model evaluation On Training Data"

```
model_eval(m,x_train,y_train)
```

```
Confussion Matrix =
[[306  42]
 [ 83 106]]
Accuracy Score =0.7672253258845437
precision value =1.0
Recall Value =0.5608465608465608
F1 Value =0.629080118694362
Clasification Report =
              precision    recall  f1-score   support

           0       0.79      0.88      0.83       348
           1       0.72      0.56      0.63       189

    accuracy                           0.77       537
   macro avg       0.75      0.72      0.73       537
weighted avg       0.76      0.77      0.76       537

'Model Evaluation Success '
```

"Model evaluation On Testing Data"

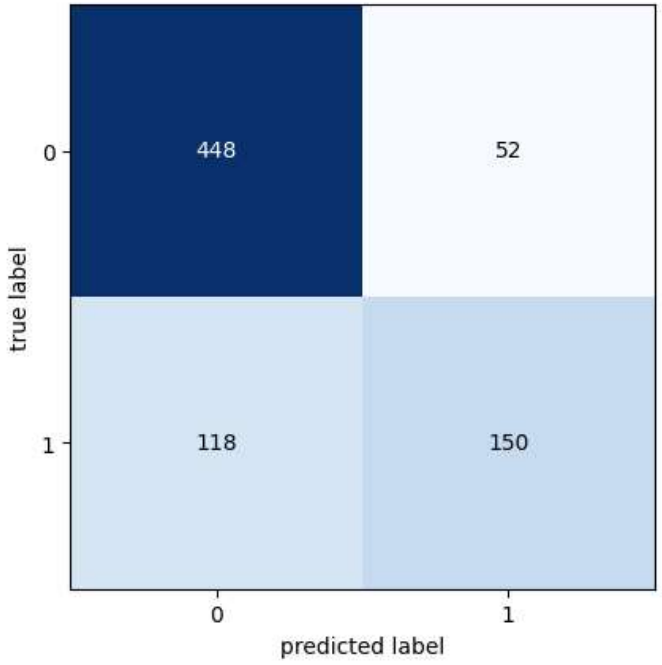
```
model_eval(m,x_test,y_test)
```

```
Confussion Matrix =
[[142  10]
 [ 35  44]]
Accuracy Score =0.8051948051948052
precision value =1.0
Recall Value =0.5569620253164557
F1 Value =0.6616541353383458
Clasification Report =
              precision    recall  f1-score   support

           0       0.80      0.93      0.86       152
           1       0.81      0.56      0.66        79

    accuracy                           0.81       231
   macro avg       0.81      0.75      0.76       231
weighted avg       0.81      0.81      0.79       231

'Model Evaluation Success '
```

PLOTTING OF CONFUSION MATRIX

```
y_pred = m.predict(x)
con_mat=confusion_matrix(y,y_pred)
plot_confusion_matrix(con_mat)
```

```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='predicted label', ylabel='true label'>)
```



PROJECT_02

Importing the libraries

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np
```

PROBLEM STATEMENT

Consider advertising.csv Dataset. Predict sales based on the money spent on TV, Radio, and Newspaper for marketing. In this case, there are three independent variables, i.e., money spent on TV, Radio, and Newspaper for marketing, and one dependent variable, i.e., sales, that is the value to be predicted. Find R squared, Mean Absolute Error, Mean Square Error, Root Mean Square Error

IMPORTING OF DATA

```
data_ad = pd.read_csv("Advertising.csv")
data_ad
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9
...	...	...	...	...	...
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5
199	200	232.1	8.6	8.7	13.4

200 rows × 5 columns

Next steps: Generate code with data_ad View recommended plots

```
data_ad.head()
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

Next steps: Generate code with data_ad View recommended plots

```
data_ad.tail()
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
195	196	38.2	3.7	13.8	7.6
196	197	94.2	4.9	8.1	9.7
197	198	177.0	9.3	6.4	12.8
198	199	283.6	42.0	66.2	25.5
199	200	232.1	8.6	8.7	13.4

Droping of the unnamed column

```
data_ad = data_ad.drop(columns=['Unnamed: 0'])
```

Spliting of the dataset

```
X_ad = data.drop('Outcome', axis=1)
y_ad = data['Outcome']
```

Creating an Linear Regression model

```
linreg = LinearRegression()
```

Fitting the Linear Regression model

```
linreg.fit(X_ad, y_ad)
```

```
▾ LinearRegression
LinearRegression()
```

```
linreg.intercept_
```

```
-0.8538942664855473
```

```
linreg.coef_
```

```
array([ 0.02059187,  0.00592027, -0.00233188,  0.00015452, -0.00018053,
        0.01324403,  0.14723744,  0.00262139])
```

```
linreg.predict(X_ad)[0:5]
```

```
array([ 0.65175729,  0.00573265,  0.73642449, -0.0219232 ,  0.83318937])
```

***Prediction on the same dataset**

```
y_pred_ad = linreg.predict(X_ad)
y_pred_ad[0:10]
```

```
array([ 0.65175729,  0.00573265,  0.73642449, -0.0219232 ,  0.83318937,
        0.21054145,  0.05736238,  0.59612039,  0.66541521,  0.00272879])
```

***Calculation of evaluation metrics**

```
r2 = r2_score(y_ad, y_pred_ad)

mae = mean_absolute_error(y_ad, y_pred_ad)

mse = mean_squared_error(y_ad, y_pred_ad)

rmse = np.sqrt(mse)

print(f'R squared: {r2}')

print(f'Mean Absolute Error: {mae}')

print(f'Mean Square Error: {mse}')

print(f'Root Mean Square Error: {rmse}')
```

```
R squared: 0.303253095650892
Mean Absolute Error: 0.3322003297974105
Mean Square Error: 0.15829143131303658
Root Mean Square Error: 0.39785855691820504
```

Start coding or generate with AI.