Logistic Regression

- is used for classification problem
- it uses sigmoid function to predict the values
- the output of sigmoid function is always between 0 & 1
- mostly preferred for binary classification but it also handle multiclass classification

In [10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [34]:
```
df = pd.read_csv(r"C:\Users\Bhupendra\Desktop\DataCenter\Logistic Regression
df.head()
```

Out[34]:

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

In [36]:
```
df.Outcome.value_counts()
```

Out[36]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [37]:
```
1 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
```

In [39]:
```
1 X = df.drop('Outcome', axis = 1)
2 y = df.Outcome
```

Train Test Split

In [58]:
```
1 from sklearn.model_selection import train_test_split
2 
3 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, ra
```

Model Building

In [62]:
```
1 from sklearn.linear_model import LogisticRegression
2 
3 model = LogisticRegression()
4 model.fit(X_train, y_train)    # model training
```

Out[62]:
```
▾ LogisticRegression

LogisticRegression()
```

Prediction

In [67]:
```
y_pred = model.predict(X_test)
y_pred            # prdicted results
```

Out[67]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
      dtype=int64)
```

In [68]:
```
y_test.values      # actual results
```

Out[68]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
      dtype=int64)
```

Evaluation

Accuracy score of the model

In [69]:
```
model.score(X_test,y_test)
```

Out[69]:
```
0.7857142857142857
```

Using different function to calculate accuracy, precision, recall score

In [70]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

In [71]:
```
# print accuracy score

accuracy_score(y_test, y_pred)
```

Out[71]:
```
0.7857142857142857
```

In [72]:
```
# print precision score

precision_score(y_test, y_pred)
```

Out[72]:
```
0.7837837837837838
```

In [73]:
```
# print recall score

recall_score(y_test, y_pred)
```

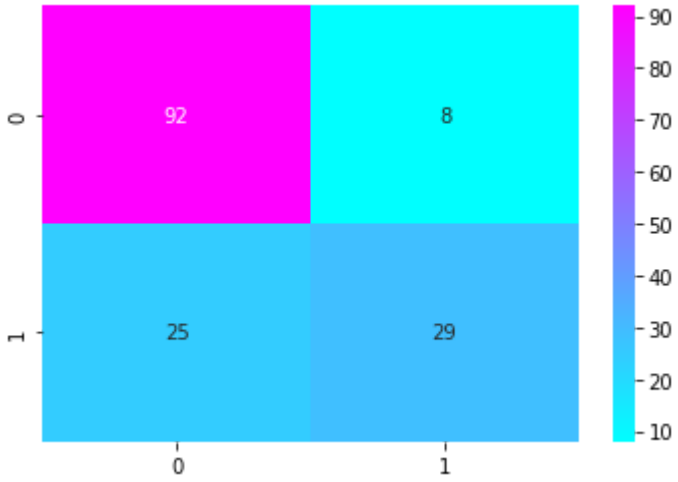
Out[73]: 0.5370370370370371

In [74]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [76]:
```
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[76]:
```
array([[92,  8],
       [25, 29]], dtype=int64)
```

In [82]:
```
# visualize the confusion matrix

sns.heatmap(cm, annot = True, cmap = 'cool');
```



In [85]:
```
cr = classification_report(y_test, y_pred)

print(cr)
```

```
              precision    recall  f1-score   support

           0       0.79      0.92      0.85       100
           1       0.78      0.54      0.64        54

    accuracy                           0.79       154
   macro avg       0.79      0.73      0.74       154
weighted avg       0.79      0.79      0.77       154
```

read more about micro, macro and weighted everage from here

https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f (https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-

clearly-explained-b603420b292f)

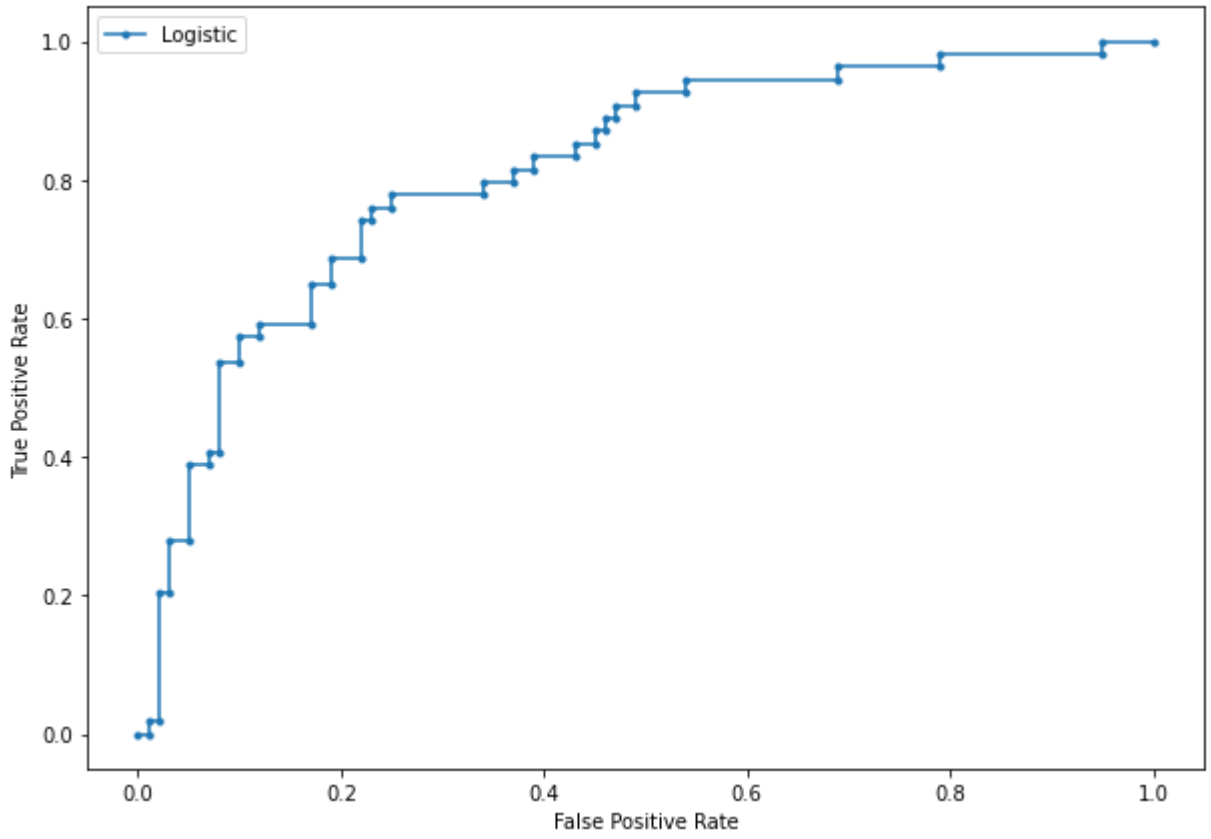
ROC curve

- which stands for receiver operating characteristics shows the tradeoff between **True Positive Rate** vs **False Positive Rate**

In [102]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [111]:
```
plt.figure(figsize = (10,7))
lr_probs = model.predict_proba(X_test)

# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]

# calculate scores
lr_auc = roc_auc_score(y_test, lr_probs)

# summarize scores
print('Logistic: ROC AUC=%.3f' % (lr_auc))


# calculate roc curves
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

# plot the roc curve for the model
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')


# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

```
Logistic: ROC AUC=0.815
```



In [ ]:
```
1
```