Madhur Jaripatke

Roll No. 48

TE A Computer

RMDSSOE, Warje, Pune

5. Data Analytics, II

1. Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset.
2. Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

Import Libraries

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
```

In [2]:
```
df = pd.read_csv('Datasets/Social_Network_Ads.csv')
df
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

Data Preprocessing

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [4]:
```
x = df.drop(['User ID', 'Purchased', 'Gender'], axis=1)
x
```

Out[4]:

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

In [5]:
```
y = df['Purchased']
y
```

Out[5]:
```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

Splitting the dataset into Training set & Test set

In [6]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [7]:
```
x_test
```

Out[7]:
```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
       [ 0.08648817,  0.21503249],
       [-1.79512465,  0.47597078],
       [-0.60673761,  1.37475825],
       [-0.11157634,  0.21503249],
       [-1.89415691,  0.44697764],
       [ 1.67100423,  1.75166912],
       [-0.30964085, -1.37959044],
       [-0.30964085, -0.65476184],
       [ 0.8787462 ,  2.15757314],
       [ 0.28455268, -0.53878926],
       [ 0.8787462 ,  1.02684052],
       [-1.49802789, -1.20563157],
       [ 1.07681071,  2.07059371],
       [-1.00286662,  0.50496393],
       [-0.90383437,  0.30201192],
       [-0.11157634, -0.21986468],
       [-0.60673761,  0.47597078],
       [-1.6960924 ,  0.53395707],
       [-0.11157634,  0.27301877],
       [ 1.86906873, -0.27785096],
       [-0.11157634, -0.48080297],
       [-1.39899564, -0.33583725],
       [-1.99318916, -0.50979612],
       [-1.59706014,  0.33100506],
       [-0.4086731 , -0.77073441],
       [-0.70576986, -1.03167271],
       [ 1.07681071, -0.97368642],
       [-1.10189888,  0.53395707],
       [ 0.28455268, -0.50979612],
       [-1.10189888,  0.41798449],
       [-0.30964085, -1.43757673],
       [ 0.48261718,  1.22979253],
       [-1.10189888, -0.33583725],
       [-0.11157634,  0.30201192],
       [ 1.37390747,  0.59194336],
       [-1.20093113, -1.14764529],
       [ 1.07681071,  0.47597078],
       [ 1.86906873,  1.51972397],
       [-0.4086731 , -1.29261101],
       [-0.30964085, -0.3648304 ],
       [-0.4086731 ,  1.31677196],
       [ 2.06713324,  0.53395707],
       [ 0.68068169, -1.089659  ],
       [-0.90383437,  0.38899135],
       [-1.20093113,  0.30201192],
       [ 1.07681071, -1.20563157],
       [-1.49802789, -1.43757673],
```

```
[-0.60673761, -1.49556302],
[ 2.1661655 , -0.79972756],
[-1.89415691,  0.18603934],
[-0.21060859,  0.85288166],
[-1.89415691, -1.26361786],
[ 2.1661655 ,  0.38899135],
[-1.39899564,  0.56295021],
[-1.10189888, -0.33583725],
[ 0.18552042, -0.65476184],
[ 0.38358493,  0.01208048],
[-0.60673761,  2.331532  ],
[-0.30964085,  0.21503249],
[-1.59706014, -0.19087153],
[ 0.68068169, -1.37959044],
[-1.10189888,  0.56295021],
[-1.99318916,  0.35999821],
[ 0.38358493,  0.27301877],
[ 0.18552042, -0.27785096],
[ 1.47293972, -1.03167271],
[ 0.8787462 ,  1.08482681],
[ 1.96810099,  2.15757314],
[ 2.06713324,  0.38899135],
[-1.39899564, -0.42281668],
[-1.20093113, -1.00267957],
[ 1.96810099, -0.91570013],
[ 0.38358493,  0.30201192],
[ 0.18552042,  0.1570462 ],
[ 2.06713324,  1.75166912],
[ 0.77971394, -0.8287207 ],
[ 0.28455268, -0.27785096],
[ 0.38358493, -0.16187839],
[-0.11157634,  2.21555943],
[-1.49802789, -0.62576869],
[-1.29996338, -1.06066585],
[-1.39899564,  0.41798449],
[-1.10189888,  0.76590222],
[-1.49802789, -0.19087153],
[ 0.97777845, -1.06066585],
[ 0.97777845,  0.59194336],
[ 0.38358493,  0.99784738]])
```

Training the Logistic Regression model on the Training set

In [8]:
```
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

Out[8]:
```
LogisticRegression
LogisticRegression(random_state=0)
```

In [9]:
```
y_train_pred = classifier.predict(x_train)
y_test_pred = classifier.predict(x_test)
```

In [10]:
```
y_train_pred
```

Out[10]:
```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [11]:
```
y_test_pred
```

Out[11]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

Predicting a new result

In [12]:
```
classifier.predict([[19,19000]])
```

Out[12]:
```
array([1], dtype=int64)
```

In [13]:
```
classifier.predict([[-0.79895082, -1.41706417]])
```

Out[13]:
```
array([0], dtype=int64)
```

In [14]:
```
classifier.predict([[-0.215686, 2.146016]])
```

Out[14]:
```
array([1], dtype=int64)
```

Creating the Confusion Matrix

In [15]:
```
matrix = confusion_matrix(y_test, y_test_pred)
matrix
```

Out[15]:
```
array([[65,  3],
       [ 8, 24]], dtype=int64)
```

In [16]:
```
score = accuracy_score(y_test, y_test_pred)
score
```

Out[16]:
```
0.89
```

In [17]:
```
print(classification_report(y_test, y_test_pred))
```

	precision	recall	f1-score	support
0	0.89	0.96	0.92	68
1	0.89	0.75	0.81	32
accuracy			0.89	100
macro avg	0.89	0.85	0.87	100
weighted avg	0.89	0.89	0.89	100

In [18]:
```
print('True Positive:', matrix[0][0])
print('True Negative:', matrix[1][1])
print('False Positive:', matrix[0][1])
print('False Negative:', matrix[1][0])
print('Accuracy:', score)
print('Error Rate:', 1-score)
precision = matrix[0][0]/(matrix[0][0]+matrix[0][1])
print('Precision:', precision)
```

```
True Positive: 65
True Negative: 24
False Positive: 3
False Negative: 8
Accuracy: 0.89
Error Rate: 0.10999999999999999
Precision: 0.9558823529411765
```