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.

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler,LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report

le = LabelEncoder()
scaler = MinMaxScaler()
lgr = LogisticRegression()
```

In [2]:

```
df = pd.read_csv('Social_Network_Ads.csv')
if df.empty == False:
    print("loaded!")
```

```
loaded!
```

In [3]:

```
df.head(5)
```

Out[3]:

	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

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [5]:

```
df.describe()
```

Out[5]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500

std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [7]:

```
df['Gender'] = le.fit_transform(df.Gender)
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    int32
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int32(1), int64(4)
memory usage: 14.2 KB
```

In [12]:

```
df_new = scaler.fit_transform(df)
```

In [14]:

```
df1 = pd.DataFrame(df_new)
df1.head()
```

Out[14]:

	0	1	2	3	4
0	0.232636	1.0	0.023810	0.029630	0.0
1	0.982732	1.0	0.404762	0.037037	0.0
2	0.409926	0.0	0.190476	0.207407	0.0
3	0.147083	0.0	0.214286	0.311111	0.0
4	0.954801	1.0	0.023810	0.451852	0.0

In [14]:

```
df = pd.DataFrame(df_new)
X=df.iloc[:,:-1] # takes all independent columns(except last one)
y=df.iloc[:,-1] # takes last dependent column
```

In [15]:

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.33,random_st
```

In [16]:

```
fitted = lgr.fit(x_train,y_train)
```

In [17]:
```
print("r2 score = {}".format(fitted.score(x_test,y_test)))
```

```
r2 score = 0.8333333333333334
```

Evaluation Matrix

In [18]:
```
predictions = fitted.predict(x_test)
```

In [19]:
```
predictions
```

Out[19]:
```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 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., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0.,
       0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 1.,
       1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1.])
```

In [20]:
```
report = classification_report(y_test,predictions,output_dict=True)
```

In [21]:
```
report = pd.DataFrame(report).transpose()
report
```

Out[21]:

	precision	recall	f1-score	support
0.0	0.816327	0.952381	0.879121	84.000000
1.0	0.882353	0.625000	0.731707	48.000000
accuracy	0.833333	0.833333	0.833333	0.833333
macro avg	0.849340	0.788690	0.805414	132.000000
weighted avg	0.840336	0.833333	0.825516	132.000000

In [22]:
```
cm = confusion_matrix(y_test,predictions)
cm
```

Out[22]:
```
array([[80,  4],
       [18, 30]], dtype=int64)
```

```
negetive positive
```

true false

In [23]:
```
tn ,fp, fn,tp  = cm.ravel()
res = '''True negetive  = {} \n
False negetive  = {}\n
True positive  = {}\n
False positive  = {}
'''
print(res.format(tn ,fp, fn,tp))
```

```
True negetive  = 80

False negetive  = 4

True positive  = 18
```

```
True positive  = 18

False positive  = 30
```

In [24]:

```
error_rate = (fp+fn)/(fp+fn+tp+tn)
print("Error rate = {}".format(error_rate))
```