In [1]:
```
# 2. Here,you need to predict the apply variable,which is an ordered categorical variable
# with responses to a survey about whether a student feels
#    they are “Unlikely” (0), “Somewhat likely” (1), or “Very likely” (2) to apply to graduate school -->odata.csv
```

1.import Nessasary Libraries

In [2]:
```
#Import Libraries
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.preprocessing import LabelEncoder,StandardScaler,MinMaxScaler
from sklearn.metrics import confusion_matrix,roc_curve,roc_auc_score,classification_report,accuracy_score

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

2.Import Dataset

In [3]:
```
#Import Datasets
data=pd.read_csv('odata.csv')
```

In [4]:
```
#Read Top 5 Data
data.head()
```

Out[4]:

	Unnamed: 0	apply	pared	public	gpa
0	1	very likely	0	0	3.26
1	2	somewhat likely	1	0	3.21
2	3	unlikely	1	1	3.94
3	4	somewhat likely	0	0	2.81
4	5	somewhat likely	0	0	2.53

3.Data Undestanding

In [5]:
```
#Data Type of columns contains
data.dtypes
```

Out[5]:
```
Unnamed: 0      int64
apply          object
pared           int64
public          int64
gpa           float64
dtype: object
```

In [6]:
```
#Indicate any null values avilible
data.isnull().sum()
```

Out[6]:
```
Unnamed: 0    0
apply         0
pared         0
public        0
gpa           0
dtype: int64
```

In [7]:
```python
#indicate number of rows and columns
data.shape
```

Out[7]: (400, 5)

In [8]:
```python
#Infomation of null entry and memomry usage
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  400 non-null    int64  
 1   apply       400 non-null    object 
 2   pared       400 non-null    int64  
 3   public      400 non-null    int64  
 4   gpa         400 non-null    float64
dtypes: float64(1), int64(3), object(1)
memory usage: 15.8+ KB
```

In [9]:
```python
data['apply'].value_counts()
```

Out[9]:
```
unlikely           220
somewhat likely    140
very likely         40
Name: apply, dtype: int64
```

In [10]:
```python
data['pared'].value_counts()
```

Out[10]:
```
0    337
1     63
Name: pared, dtype: int64
```

In [11]:
```
data['public'].value_counts()
```

Out[11]:
```
0    343
1     57
Name: public, dtype: int64
```

4.Data Preparing

In [12]:
```
#Remove Unnessary columns
data=data.drop(['Unnamed: 0'],axis=1)
```

In [13]:
```
#Convert in catagorical data in numeric representation for further process.
#One-Hot-Encoding has the advantage that the result is binary rather than ordinal
# Every unique value in the category will be added as a feature.
data=pd.get_dummies(data,columns=['public'],drop_first=True)
data=pd.get_dummies(data,columns=['pared'],drop_first=True)
```

In [14]:
```
#labling output data in numeric data that's machine can Undestand
#Its always advisable to use label encoder for output feature
le=LabelEncoder()
data['apply']=le.fit_transform(data['apply'])
```

In [15]:
```
#We have to devide dataset in input and output
x=data.drop(['apply'],axis=1)
y=data['apply']
```

In [17]:
```
#User define normlize function
# Normalization by adding and/or multiplying by constants so values fall between -1 and 1.
def norm_fun(i):
    return (i-i.max())/(i.max()-i.min())
```

In [18]:
```
#Apply normlize funcation
x['gpa']=norm_fun(x['gpa'])
```

In [19]:
```
#Data Trasform and value fall between -1 to 1.
x.head()
```

Out[19]:

	gpa	public_1	pared_1
0	-0.352381	0	0
1	-0.376190	0	1
2	-0.028571	1	1
3	-0.566667	0	0
4	-0.700000	0	0

In [20]:
```
#Devide data in training and testing part.20%value fall in training dataset and 80% value fall in training dataset
x_train,x_test,y_train,y_test=train_test_split(x,y,shuffle=True,stratify=y,random_state=12,test_size=0.2)
```

5.Model Building

In [21]:
```
model=LogisticRegression(C= 20,      #Inverse of regularization strength
                         max_iter= 5000,#Maximum number of iterations taken for the solvers to converge.
                         solver="saga" ,         #Algorithm to use in the optimization problem
                          class_weight="balanced", #Weights associated with classes
                          verbose=5,
                          tol=0.01) # set verbose to any positive number for verbosity                        )
```

In [22]:
```
model.fit(x_train,y_train)
```

```
convergence after 24 epochs took 0 seconds

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

Out[22]:
```
LogisticRegression(C=20, class_weight='balanced', max_iter=5000, solver='saga',
                   tol=0.01, verbose=5)
```

6.Model Testing

In [23]:
```
y_pred_test=model.predict(x_test)
y_pred_train=model.predict(x_train)
y_pred_test
```

Out[23]:
```
array([0, 1, 2, 1, 0, 1, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 2, 2, 0, 1, 1, 1, 1, 2, 2, 2, 2, 1,
       1, 1, 1, 1, 1, 1, 2, 1, 0, 2, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 2, 1,
       1, 2, 2, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1])
```

In [24]:
```
result=pd.DataFrame({"Actual":y_test,"predicted":y_pred_test})
result.tail()
```

Out[24]:

	Actual	predicted
324	1	1
321	1	1
332	0	1
34	1	1
313	2	1

7.Accuracy Score

In [25]:
```
accuracy_score(y_test,y_pred_test)
```

Out[25]:
```
0.6
```

8.Confusion Matrix

In [26]:
```
confusion_matrix(y_test,y_pred_test)
```

Out[26]:
```
array([[ 9, 10,  9],
       [ 3, 36,  5],
       [ 0,  5,  3]], dtype=int64)
```

9.Classification Report

In [27]:
```
#Classification report
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.32      0.45        28
           1       0.71      0.82      0.76        44
           2       0.18      0.38      0.24         8

    accuracy                           0.60        80
   macro avg       0.54      0.50      0.48        80
weighted avg       0.67      0.60      0.60        80
```

10 Roc Auc Score

In [28]:
```
#Roc auc score
y_pred_prob=model.predict_proba(x)
y_pred_prob
```

Out[28]:
```
array([[0.34973856, 0.35600751, 0.29425393],
       [0.38471083, 0.1581867 , 0.45710247],
       [0.17646824, 0.11397439, 0.70955736],
       ...,
       [0.36451187, 0.43357594, 0.20191219],
       [0.34973856, 0.35600751, 0.29425393],
       [0.34307028, 0.33549574, 0.32143398]])
```

In [29]:
```python
#ROC AUC SCORE
area_under_curve = roc_auc_score(y,y_pred_prob,multi_class="ovr")
area_under_curve
```

Out[29]:
```
0.6015522787397788
```

11.Model Evalution

In [30]:
```python
# Model Evaluation is the process through which we quantify the quality of a system's predictions.
# To do this, we measure the newly trained model performance on a new and independent dataset.
# This model will compare labeled data with it's own predictions
#For Model we prepare some data
new_data=pd.DataFrame({'pared':0,"public":0,"gpa":3.26},index=[0])
```

In [31]:
```python
#New input data which is unknown for model
#We will give this data to model and find out model predition
new_data
```

Out[31]:

	pared	public	gpa
0	0	0	3.26

In [32]:
```python
#Predict output of our own input
y_pred=model.predict(new_data)
```

In [33]:
```python
#it indicate student very likely to apply to graduate school
# that's True answer
y_pred
```

Out[33]:
```
array([2])
```