In [1]:
```
#Task1. You need to predict the type of program a studen in based on other attributes--> mdata.csv
# prog :    is a categorical variable indicating what type of program a student is in: “General” (1), “Academic” (2), or “Vocational” (3)
# Ses:   is a categorical variable indicating someone’s socioeconomic class: “Low” (1), “Middle” (2), and “High” (3)
# read,write,math,science:  is their scores on different tests
# honors:   Whether they have enrolled or not
```

1. Import Libraries

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,roc_auc_score,classification_report,accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

2. import Datasets

In [3]:
```
#import datasets
data=pd.read_csv("mdata.csv")
```

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

Out[4]:

	Unnamed: 0	id	female	ses	schtyp	prog	read	write	math	science	honors
0	1	45	female	low	public	vocation	34	35	41	29	not enrolled
1	2	108	male	middle	public	general	34	33	41	36	not enrolled
2	3	15	male	high	public	vocation	39	39	44	26	not enrolled
3	4	67	male	low	public	vocation	37	37	42	33	not enrolled
4	5	153	male	middle	public	vocation	39	31	40	39	not enrolled

3.Data Undestading

In [5]:
```
#indicate number of rows and columns
data.shape
```

Out[5]: (200, 11)

In [6]:
```
#Infomation of null entry and memomry usage
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  200 non-null    int64 
 1   id          200 non-null    int64 
 2   female      200 non-null    object
 3   ses         200 non-null    object
 4   schtyp      200 non-null    object
 5   prog        200 non-null    object
 6   read        200 non-null    int64 
 7   write       200 non-null    int64 
 8   math        200 non-null    int64 
 9   science     200 non-null    int64 
 10  honors      200 non-null    object
dtypes: int64(6), object(5)
memory usage: 17.3+ KB
```

In [7]:
```
#Indicate any null values avilible
data.isnull().sum()
```

Out[7]:
```
Unnamed: 0    0
id            0
female        0
ses           0
schtyp        0
prog          0
read          0
write         0
math          0
science       0
honors        0
dtype: int64
```

In [8]:

```
#No dublicate data avilible
data[data.duplicated()]
```

Out[8]:

Unnamed: 0	id	female	ses	schtyp	prog	read	write	math	science	honors

4.Data Preparing

In [9]:

```
#Drop Unnasasary columns
data=data.drop(['Unnamed: 0','id'],axis=1)
```

In [10]:

```
#Apply One hot encoding in input catagorical data
#One-Hot-Encoding has the advantage that the result is binary rather than ordinal
# One-Hot Encoding is the process of creating dummy variables.
# It simply creates additional features based on the number of unique values in the categorical feature
# Every unique value in the category will be added as a feature.
data=pd.get_dummies(data,columns=['schtyp'],drop_first=True)
data=pd.get_dummies(data,columns=['ses'],drop_first=True)
data=pd.get_dummies(data,columns=['female'],drop_first=True)
data=pd.get_dummies(data,columns=['honors'],drop_first=True)
```

In [11]:

```
#Apply label Encoder in output feature
#its always advisable to use label encoder for output feature
le=LabelEncoder()
data['prog']=le.fit_transform(data['prog'])
```

In [12]:
```
data.head()
```

Out[12]:

	prog	read	write	math	science	schtyp_public	ses_low	ses_middle	female_male	honors_not enrolled
0	2	34	35	41	29	1	1	0	0	1
1	1	34	33	41	36	1	0	1	1	1
2	2	39	39	44	26	1	0	0	1	1
3	2	37	37	42	33	1	1	0	1	1
4	2	39	31	40	39	1	0	1	1	1

In [13]:
```
#Devide in input and output
x=data.drop(['prog'],axis=1)
y=data['prog']
```

4.Normlize input features

In [14]:
```
#User define normlize function
def norm_fun(i):
    return (i-i.max())/(i.max()-i.min())
```

In [15]:
```
# Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
# It is helpful to divide the large database table into smaller tables and link them using relationship.
# It avoids duplicate data or no repeating groups into a table.
x=norm_fun(x)
```

5.Model Building

In [16]:
```
#Model Creation
# Logistic Regression is used when the dependent variable(target) is categorical.
#Their value strictly ranges from 0 to 1.
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=10 # set verbose to any positive number for verbosity
                        )
```

In [17]:
```
#Fit The Model
model.fit(x,y)
```

```
convergence after 44 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[17]:
```
LogisticRegression(C=20, class_weight='balanced', max_iter=5000, solver='saga',
                   verbose=10)
```

6.Model Training

In [18]:
```
#output  prediction with model
y_pred=model.predict(x)
y_pred
```

Out[18]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 1, 2, 2, 1, 2, 1, 1, 0, 2, 2, 2, 0, 1, 1, 2, 0, 0, 2,
       1, 1, 2, 1, 2, 0, 2, 2, 1, 0, 1, 1, 0, 1, 0, 2, 2, 2, 0, 2, 2, 2,
       1, 0, 0, 2, 1, 2, 1, 1, 2, 1, 0, 2, 2, 1, 1, 2, 0, 1, 0, 1, 2, 2,
       1, 2, 1, 1, 0, 2, 2, 0, 1, 1, 2, 0, 1, 2, 1, 1, 2, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 2, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 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,
       0, 0])
```

In [19]:
```python
result=pd.DataFrame({"Actual":y,"predicted":y_pred})
result
```

Out[19]:

	Actual	predicted
0	2	2
1	1	2
2	2	2
3	2	2
4	2	2
...	...	...
195	0	0
196	2	0
197	0	0
198	0	0
199	0	0

200 rows × 2 columns

7.Confusion matrix

In [20]:
```python
#Confusion Matrix
confusion_matrix(y,y_pred)
```

Out[20]:
```
array([[73, 21, 11],
       [10, 18, 17],
       [ 8,  9, 33]], dtype=int64)
```

8.Classification Report

In [21]:
```python
#Classification report
print(classification_report(y,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.70      0.74       105
           1       0.38      0.40      0.39        45
           2       0.54      0.66      0.59        50

    accuracy                           0.62       200
   macro avg       0.57      0.59      0.58       200
weighted avg       0.64      0.62      0.63       200
```

9.Roc Auc Score

In [22]:
```python
#Roc auc score
y_pred_prob=model.predict_proba(x)
y_pred_prob
```

Out[22]:
```
array([[0.08163923, 0.24168712, 0.67667365],
       [0.03140983, 0.13395557, 0.83463461],
       [0.41075562, 0.14354823, 0.44569615],
       [0.12009771, 0.33588912, 0.54401316],
       [0.02616027, 0.1365507 , 0.83728902],
       [0.20641639, 0.15445893, 0.63912468],
       [0.04900342, 0.18051674, 0.77047984],
       [0.07067018, 0.12643037, 0.80289945],
       [0.01324678, 0.14671843, 0.84003479],
       [0.06253432, 0.18985376, 0.74761192],
       [0.06204758, 0.39378611, 0.54416631],
       [0.042264  , 0.26105426, 0.69668174],
       [0.14331375, 0.3235326 , 0.53315364],
       [0.02279538, 0.17912872, 0.7980759 ],
       [0.01485305, 0.40147347, 0.58367348],
       [0.17181504, 0.38437877, 0.44380619],
       [0.04747414, 0.4682585 , 0.48426736],
       [0.06455839, 0.17297398, 0.76246764],
       [0.10132832, 0.24643355, 0.65223813],
```

In [23]:
```
#ROC AUC SCORE
area_under_curve = roc_auc_score(y,y_pred_prob,multi_class="ovr")
area_under_curve
```

Out[23]: 0.7923895401586402

10.Model Evalution

In [24]:
```
# 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({'read':34,'write':35,'math':41,'science':29,'schtyp_public':1,'ses_low':1,'ses_middle':0,'female_male':0,'honors_not enrolled'
```

In [25]:
```
#New input data which is unknown for model
#We will give this data to model and find out model predition
new_data
```

Out[25]:

	read	write	math	science	schtyp_public	ses_low	ses_middle	female_male	honors_not enrolled
0	34	35	41	29	1	1	0	0	1

In [26]:
```
#Predict output of our own input
y_pred_new=model.predict(new_data)
```

In [27]:
```
#it predict that student enroll in vocation  programme
# that's True answer
y_pred_new
```

Out[27]: array([0])