Analysis of Student Academic Performance

by

Dhairya Chheda

Department of Information Technology

Shah & Anchor Kutchhi Engineering College, Mumbai

I. Problem Statement

Educational data mining focuses on developing different methods for solving educational problems which are hidden in an education field. The major problem which is faced in an education field is ‘Student Dropouts or Failure’. There are many factors which are influencing the student dropouts. Many Data mining methods are used for identifying and predicting student’s failure.

A student failure is a major social problem where educational professionals need to understand the causes, why many students fail in completing their education. It is a difficult task as there are many factors that cause student failure.

II. Dataset

This is an educational data set which consists of 305 males and 175 females. The datset is actually divided into three features namely Academic Feature (includes SectionID, GradeID, Topic, Semester), Behavioral features (includes RaisedHands, VisitedResources, AnnouncementView, Discussion) and Demographic features (includes Relation, ParentAnsweringSurvey, ParentschoolSatisfaction).

The students come from different origins such as 179 students are from Kuwait, 172 students are from Jordan, 28 students from Palestine, 22 students are from Iraq, 17 students from Lebanon, 12 students from Tunis, 11 students from Saudi Arabia, 9 students from Egypt, 7 students from Syria, 6 students from USA, Iran and Libya, 4 students from Morocco and one student from Venezuela.

The dataset is collected through two educational semesters: 245 student records are collected during the first semester and 235 student records are collected during the second semester.

The data set includes also the school attendance feature such as the students are classified into two categories based on their absence days: 191 students exceed 7 absence days and 289 students their absence days under 7.

This dataset includes also a new category of features; this feature is parent parturition in the educational process. Parent participation features have two sub features: Parent Answering Survey and Parent School Satisfaction. There are 270 of the parents answered surveys and 210 are not, 292 of the parents are satisfied from the school and 188 are not.

III. Algorithms Used

I have used three algorithms namely:

a) Logistic Regression.
b) Decision Tree.
c) Random Forest.
d) Extreme Gradient Boosting.

IV. Visualisation

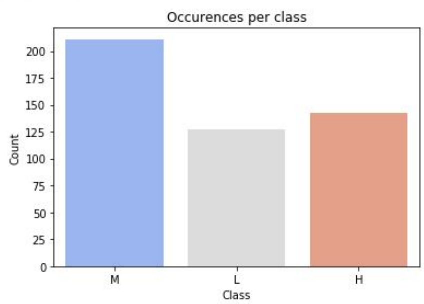
In [4]:
```
# Here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
#% matplotlib inline
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import cross_val_score

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

#from subprocess import check_output
#print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

In [15]:
```
# Import data, start exploratory data analysis
edm = pd.read_csv('student_performance.csv')
edm.head()
```

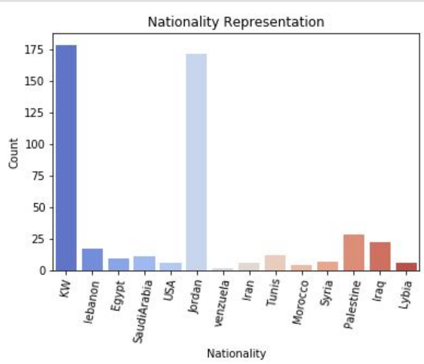
In [16]:
```
edm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 17 columns):
Gender                      480 non-null object
Nationality                 480 non-null object
PlaceofBirth                480 non-null object
StageID                     480 non-null object
GradeID                     480 non-null object
SectionID                   480 non-null object
Topic                       480 non-null object
Semester                    480 non-null object
Relation                    480 non-null object
RaisedHands                 480 non-null int64
VisitedResources            480 non-null int64
AnnouncementsView           480 non-null int64
Discussion                  480 non-null int64
ParentAnsweringSurvey       480 non-null object
ParentschoolSatisfaction    480 non-null object
StudentAbsenceDays          480 non-null object
Class                       480 non-null object
dtypes: int64(4), object(13)
memory usage: 63.9+ KB
```

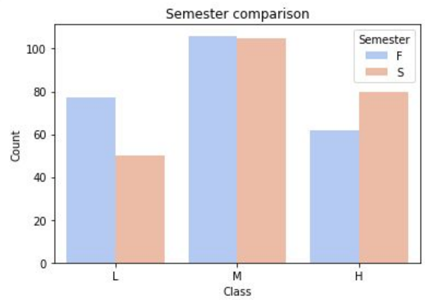
In [17]:
```
# Counts per class --> Is the dataset unbalanced?
counts = sns.countplot(x='Class', data=edm, palette='coolwarm')
counts.set(xlabel='Class', ylabel='Count', title='Occurences per class')
plt.show()
```



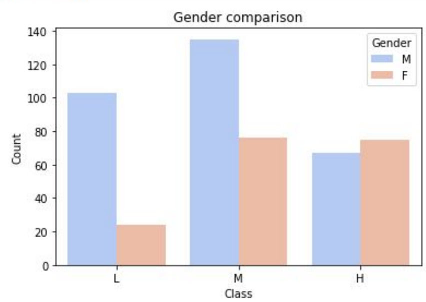
In [18]:
```
# Exploring nationalities
nat = sns.countplot(x='Nationality', data=edm, palette='coolwarm')
nat.set(xlabel='Nationality', ylabel='Count', title='Nationality Representation')
plt.setp(nat.get_xticklabels(), rotation=80)
plt.show()
```



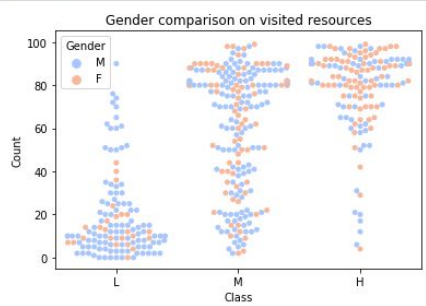
In [19]:
```
# Semester comparison
sem = sns.countplot(x='Class', hue='Semester', order=['L', 'M', 'H'], data=edm, palette='coolwarm')
sem.set(xlabel='Class', ylabel='Count', title='Semester comparison')
plt.show()
```



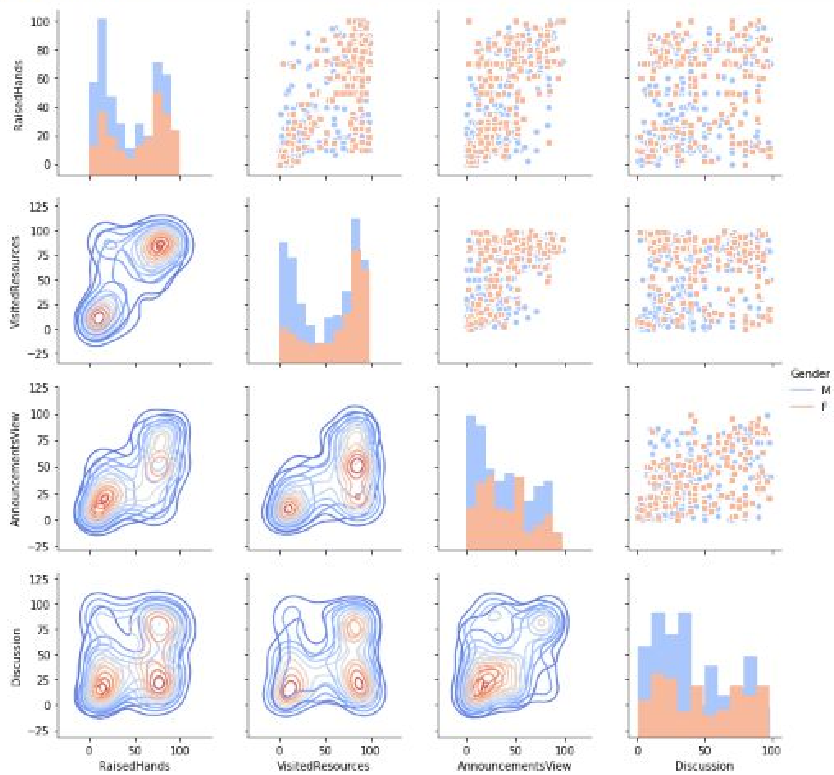
In [20]:
```
# gender comparison
plot = sns.countplot(x='Class', hue='Gender', data=edm, order=['L', 'M', 'H'], palette='coolwarm')
plot.set(xlabel='Class', ylabel='Count', title='Gender comparison')
plt.show()
```



In [21]:
```
plot = sns.swarmplot(x='Class', y='VisitedResources', hue='Gender', order=['L', 'M', 'H'],
              data=edm, palette='coolwarm')
plot.set(xlabel='Class', ylabel='Count', title='Gender comparison on visited resources')
plt.rcParams['figure.figsize']=(10,5)
plt.show()
```



In [23]:
```python
# Pairgrid, exploring our numerical variables and keeping the gender comparison
g = sns.PairGrid(edm, hue='Gender', palette='coolwarm', hue_kws={'marker': ['o', 's']})
g = g.map_diag(plt.hist)
g = g.map_upper(plt.scatter, linewidths=1, edgecolor='w', s=40)
g = g.map_lower(sns.kdeplot, lw=3, legend=False, cmap='coolwarm')
g = g.add_legend()
```



In [11]:
```python
X = edm.drop('Class', axis=1)
y = edm['Class']

# Encoding our categorical columns in X
labelEncoder = LabelEncoder()
cat_columns = X.dtypes.pipe(lambda x: x[x == 'object']).index
for col in cat_columns:
    X[col] = labelEncoder.fit_transform(X[col])

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=52)
```

In [25]:
```python
# Logistic Regression as baseline, then exploring tree-based methods

keys = []
scores = []
models = {'Logistic Regression': LogisticRegression(), 'Decision Tree': DecisionTreeClassifier(),
          'Random Forest': RandomForestClassifier(n_estimators=300, random_state=52)}

for k,v in models.items():
    mod = v
    mod.fit(X_train, y_train)
    pred = mod.predict(X_test)
    print('Results for: ' + str(k) + '\n')
    print(confusion_matrix(y_test, pred))
    print(classification_report(y_test, pred))
    acc = accuracy_score(y_test, pred)
    print("accuracy is "+ str(acc))
    print('\n' + '\n')
    keys.append(k)
    scores.append(acc)
    table = pd.DataFrame({'model':keys, 'accuracy score':scores})

print(table)
```

```
Results for: Logistic Regression

[[37  1 14]
 [ 0 32  4]
 [ 8  7 41]]
              precision    recall  f1-score   support

           H       0.82      0.71      0.76        52
           L       0.80      0.89      0.84        36
           M       0.69      0.73      0.71        56

    accuracy                           0.76       144
   macro avg       0.77      0.78      0.77       144
weighted avg       0.77      0.76      0.76       144

accuracy is 0.7638888888888888
```

```
Results for: Decision Tree

[[32  0 20]
 [ 1 30  5]
 [15  5 36]]
              precision    recall  f1-score   support

           H       0.67      0.62      0.64        52
           L       0.86      0.83      0.85        36
           M       0.59      0.64      0.62        56

    accuracy                           0.68       144
   macro avg       0.70      0.70      0.70       144
weighted avg       0.68      0.68      0.68       144

accuracy is 0.6805555555555556
```

```
Results for: Random Forest

[[36  0 16]
 [ 0 31  5]
 [ 2  4 50]]
              precision    recall  f1-score   support

           H       0.95      0.69      0.80        52
           L       0.89      0.86      0.87        36
           M       0.70      0.89      0.79        56

    accuracy                           0.81       144
   macro avg       0.85      0.82      0.82       144
weighted avg       0.84      0.81      0.81       144

accuracy is 0.8125
```

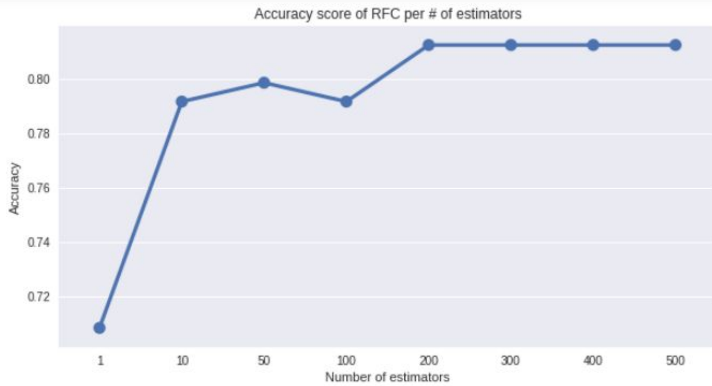
	model	accuracy score
0	Logistic Regression	0.763889
1	Decision Tree	0.680556
2	Random Forest	0.812500

Random Forest

The Random Forest Classifier performed best. Let's explore the number of estimators in the forest further. A general rule is that the RFC performs better when the amount of estimators increases.

In [13]:

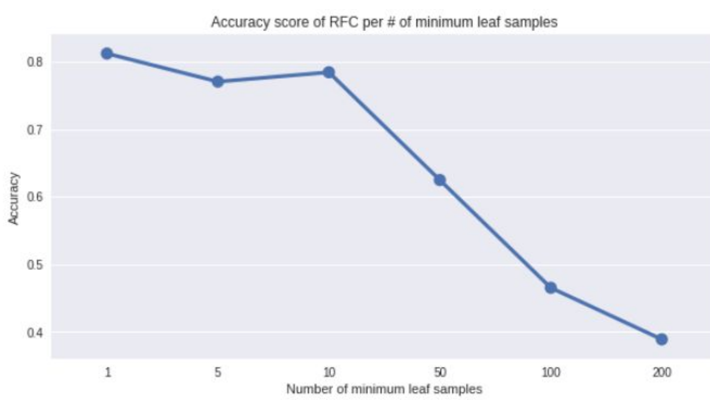
```
# Exploring the number of estimators in the random forest
score = []
est = []
estimators = [1, 10, 50, 100, 200, 300, 400, 500]
for e in estimators:
    rfc1 = RandomForestClassifier(n_estimators=e, random_state=52)
    pred1 = rfc1.fit(X_train, y_train).predict(X_test)
    accuracy = accuracy_score(y_test, pred1)
    score.append(accuracy)
    est.append(e)
plot = sns.pointplot(x=est, y=score)
plot.set(xlabel='Number of estimators', ylabel='Accuracy',
         title='Accuracy score of RFC per # of estimators')
plt.show()
```



And indeed, the RFC performs better when the number of estimators increases. However, it plateaus at 200 estimators. In the for loop before I used 300 estimators which is a general number I like to start trying it out with. Apparently 200 estimators is enough for this dataset. If you start experimenting on a very large dataset, having less estimators will save you a lot of running time.

We can also explore another variable like the minimum number of samples required to be at a leaf node.

In [14]:
```
# Exploring minimum leaf samples
score = []
leaf = []
leaf_options = [1, 5, 10, 50, 100, 200]
for l in leaf_options:
    rfc2 = RandomForestClassifier(n_estimators=200, random_state=52, min_samples_leaf=l)
    pred2 = rfc2.fit(X_train, y_train).predict(X_test)
    accuracy = accuracy_score(y_test, pred2)
    score.append(accuracy)
    leaf.append(l)
plot = sns.pointplot(x=leaf, y=score)
plot.set(xlabel='Number of minimum leaf samples', ylabel='Accuracy',
         title='Accuracy score of RFC per # of minimum leaf samples')
plt.show()
```



In this case we see that the accuracy score simply decreases as the minimum leaf samples increase. Therefore, it is best to keep this value at the default of 1.

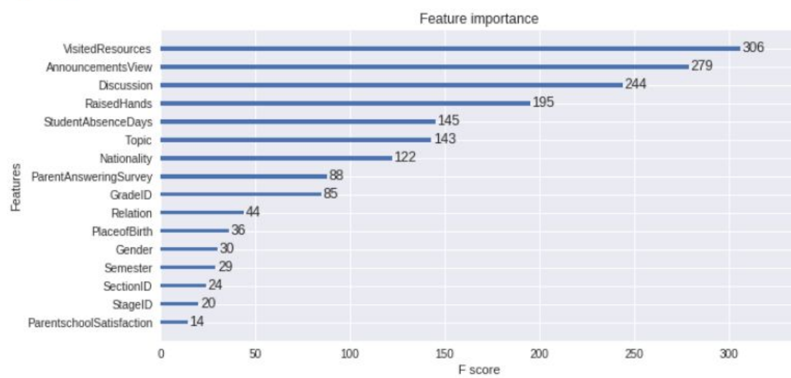
Extreme Gradient Boosting

Many Kaggle competitions have been won by using Extreme Gradient Boosting. I have never used it so let's give it a try. If you have any tips please share them in the comments.

In [15]:
```
xgb = XGBClassifier(seed=52)
pred = xgb.fit(X_train, y_train).predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("accuracy is "+ str(accuracy_score(y_test, pred)))
```

```
[[38  0 14]
 [ 0 30  6]
 [ 4  5 47]]
             precision    recall  f1-score   support

          H       0.90      0.73      0.81        52
          L       0.86      0.83      0.85        36
          M       0.70      0.84      0.76        56

avg / total       0.81      0.80      0.80       144

accuracy is 0.798611111111
```

In [16]:
```
plot_importance(xgb)
plt.rcParams['figure.figsize']=(10,5)
plt.show()
```



In [17]:
```
# Let's try to improve the accuracy of the XGClassifier with a grid search approach.

d_values = []
l_values = []
n_values = []
acc_values = []
depth = [2, 3, 4]
learning_Rate = [0.01, 0.1, 1]
n_estimators = [50, 100, 150, 200]
for d in depth:
    for l in learning_Rate:
        for n in n_estimators:
            xgb = XGBClassifier(max_depth=d, learning_rate=l, n_estimators=n, seed=52)
            pred = xgb.fit(X_train, y_train).predict(X_test)
            acc = accuracy_score(y_test, pred)
            d_values.append(d)
            l_values.append(l)
            n_values.append(n)
            acc_values.append(acc)

dict = {'max_depth':d_values, 'learning_rate':l_values, 'n_estimators':n_values,
       'accuracy':acc_values}

output = pd.DataFrame.from_dict(data=dict)
print(output.sort_values(by='accuracy', ascending=False))
```

	accuracy	learning_rate	max_depth	n_estimators
29	0.819444	0.10	4	100
31	0.812500	0.10	4	200
16	0.805556	0.10	3	50
20	0.805556	1.00	3	50
27	0.805556	0.01	4	200
28	0.798611	0.10	4	50
19	0.798611	0.10	3	200
4	0.798611	0.10	2	50
23	0.798611	1.00	3	200
17	0.798611	0.10	3	100
30	0.798611	0.10	4	150
22	0.798611	1.00	3	150
14	0.798611	0.01	3	150
5	0.791667	0.10	2	100
13	0.791667	0.01	3	100
15	0.791667	0.01	3	200
21	0.784722	1.00	3	100
7	0.784722	0.10	2	200
32	0.784722	1.00	4	50
33	0.777778	1.00	4	100
34	0.777778	1.00	4	150
18	0.777778	0.10	3	150
35	0.777778	1.00	4	200
8	0.777778	1.00	2	50
6	0.777778	0.10	2	150
26	0.770833	0.01	4	150
25	0.763889	0.01	4	100
12	0.763889	0.01	3	50
9	0.763889	1.00	2	100
3	0.763889	0.01	2	200
11	0.756944	1.00	2	200
10	0.756944	1.00	2	150
2	0.743056	0.01	2	150
24	0.736111	0.01	4	50
1	0.715278	0.01	2	100
0	0.645833	0.01	2	50

Accuracy improved :)

We can see that using a learning_rate of 0.1, a max_depth of 4 and 100 estimators in our XGB classifier provides an accuracy of 0.8194.

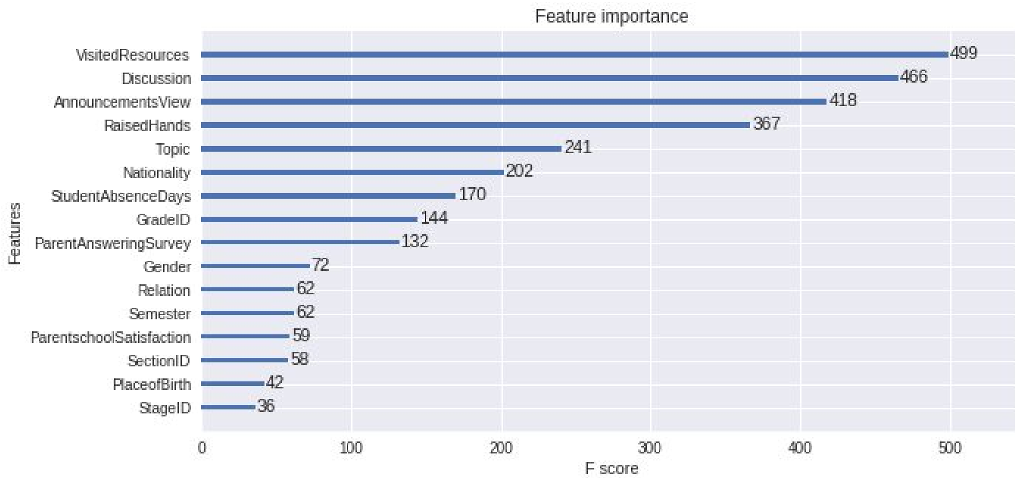
This is a nice improvement over our previous score of 0.7986

The XGB model now also performs better than the random forest classifier which capped at 0.8125

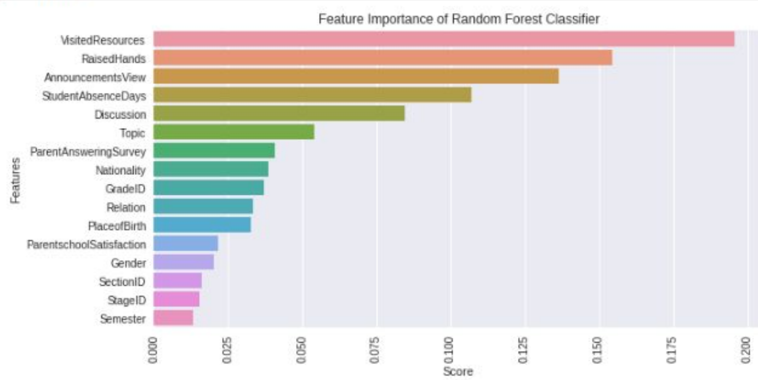
Let's explore the important features in this 'best' model.

In [18]:
```
# Building the best XGB and looking at feature importances

xgb2 = XGBClassifier(max_depth=4, learning_rate=0.1, n_estimators=100, seed=52)
pred = xgb2.fit(X_train, y_train).predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
print("accuracy is "+ str(accuracy_score(y_test, pred)))

plot_importance(xgb2)
plt.rcParams['figure.figsize']=(10,5)
plt.show()
```

```
[[39  0 13]
 [ 0 32  4]
 [ 5  4 47]]
             precision    recall  f1-score   support

          H       0.89      0.75      0.81        52
          L       0.89      0.89      0.89        36
          M       0.73      0.84      0.78        56

avg / total       0.83      0.82      0.82       144

accuracy is 0.819444444444
```




```
In [19]: rfc = RandomForestClassifier(n_estimators=200, random_state=52)
pred = rfc.fit(X_train, y_train).predict(X_test)
dn = {'features':X.columns, 'score':rfc.feature_importances_}
df = pd.DataFrame.from_dict(data=dn).sort_values(by='score', ascending=False)
plot = sns.barplot(x='score', y='features', data=df, orient='h')
plot.set(xlabel='Score', ylabel='Features',
         title='Feature Importance of Random Forest Classifier')
plt.setp(plot.get_xticklabels(), rotation=90)
plt.show()
```



V. BI Decision

Assigning research-based topics to students help them explore the online resources for their survey. Moreover, regular class test results will provide how attentive and interactive the students are in the class while lecture delivery.

Furthermore, an eye should be kept on how frequently the students check the announcements made by the professor and be strictly followed. Adding on, group discussions should be organized in order to share the knowledge of any trending topic related to studies.

These prove to be the promising factors to decrease the Student's Dropout or Failure rate.