Data Analysis and Machine-Learning

Chapter 10.6.

ML Modelling Applications (5)

*Parameter Tuning for Various Types of ML/Deep-Learning Models:*

*SVM, Random Forest, XGB, LightGBM, CatBoost,*

*Neural Network, Keras(CNN),*

*and Stacking Model*



*Data is the new science.*

*Big Data holds the answers.*

*- Pat Gelsinger*

1. Introduction

Continuing from the last chapter, this subchapter will demonstrate parameter tuning techniques for various types of machine-learning and deep-learning models, including (1) Generalized Linear Models, Support Vector Machine, (2) LightGBM, Catboost (boosting models), (3) Neural Network and CNN (Deep-learning models), and (4) Stacking Models. Along with the model tuning techniques, better visualization techniques using *ax patches* and *pandas crosstab* are to be introduced during the EDA process, together with *encoding* techniques for target variable with classification data. Students’ academic performance xAPI dataset is used for demonstrations (for specific information and citations regarding the dataset, consider following papers: (1) Amrieh, E. A., Hamtini, T., & Aljarah, I. (2016). Mining Educational Data to Predict Student’s academic Performance using Ensemble Methods. International Journal of Database Theory and Application, 9(8), 119-136., (2) Amrieh, E. A., Hamtini, T., & Aljarah, I. (2015, November). Preprocessing and analyzing educational data set using X-API for improving student's performance. In Applied Electrical Engineering and Computing Technologies (AEECT), 2015 IEEE Jordan Conference on (pp. 1-5). IEEE.

2. EDA (Exploratory Data Analysis) and Visualizations of Variables

import numpy as np

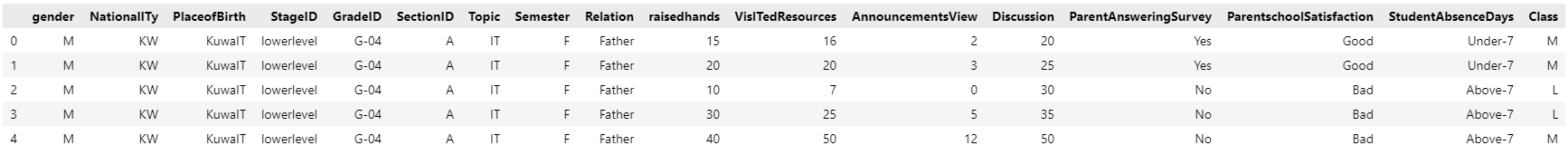
import pandas as pd

import matplotlib.pyplot as plt

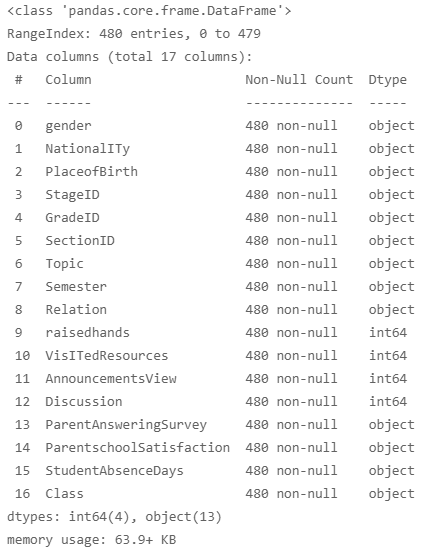
import seaborn as sns

rawData = pd.read\_csv('Your File Path\\xAPI-Edu-Data.csv')

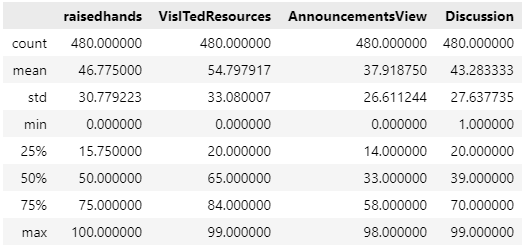
rawData.head()



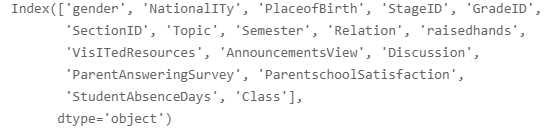
rawData.info()



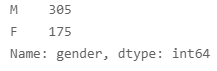
rawData.describe()



rawData.columns

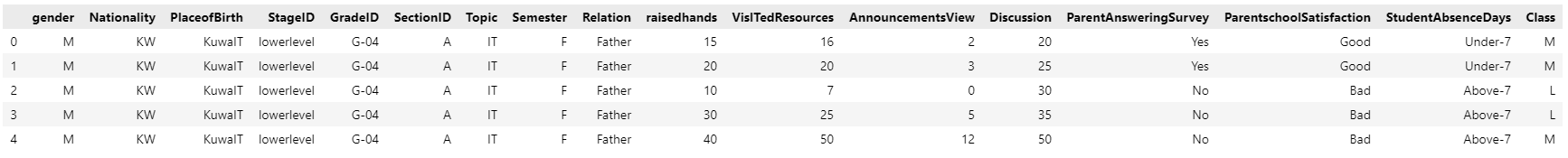


rawData['gender'].value\_counts()

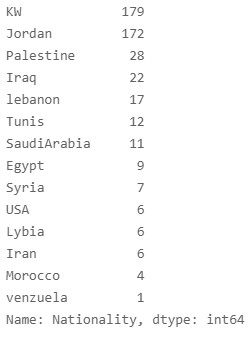


rawData.rename(columns = {'NationalITy': 'Nationality'}, inplace=True)

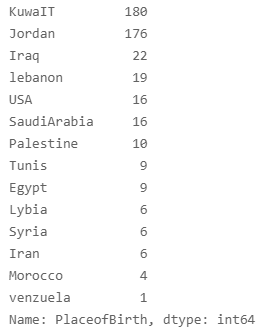
rawData.head()



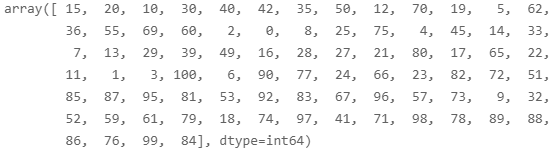
rawData['Nationality'].value\_counts()



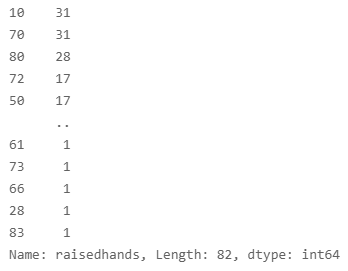
rawData['PlaceofBirth'].value\_counts()



rawData['raisedhands'].unique()

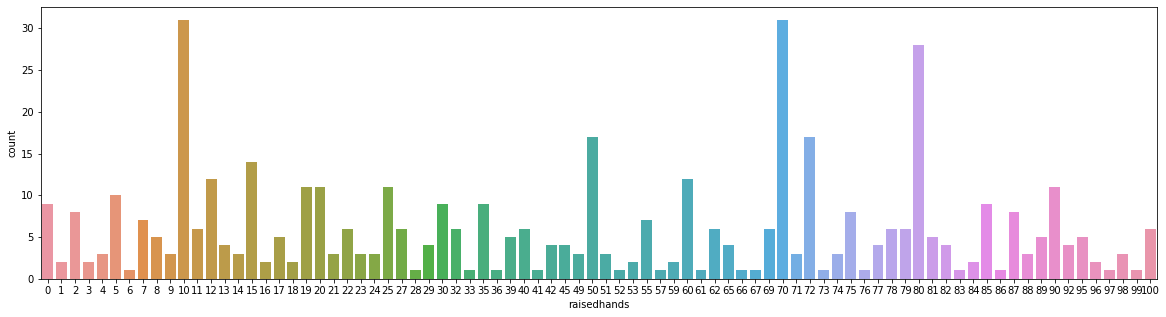


rawData['raisedhands'].value\_counts()

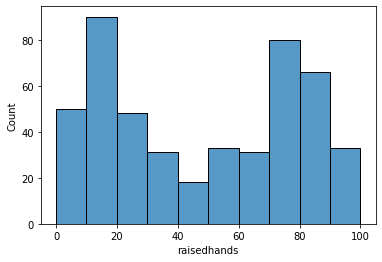


plt.figure(figsize=(20,5))

sns.countplot(data=rawData, x='raisedhands')

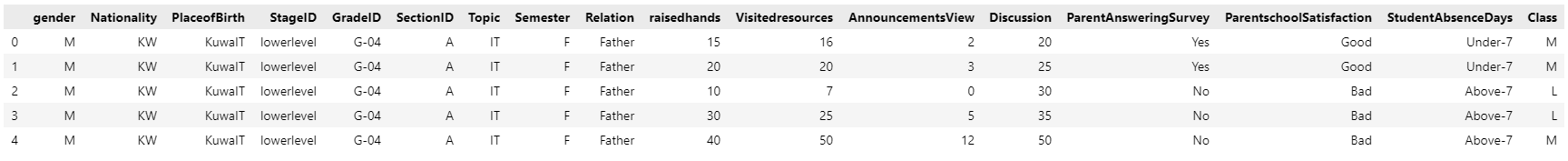


sns.histplot(data=rawData, x='raisedhands')

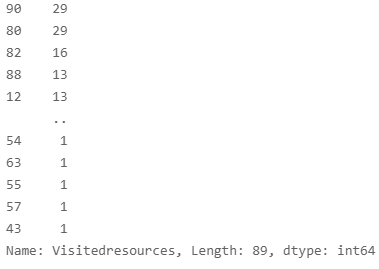


rawData.rename(columns={'VisITedResources': 'Visitedresources'}, inplace=True)

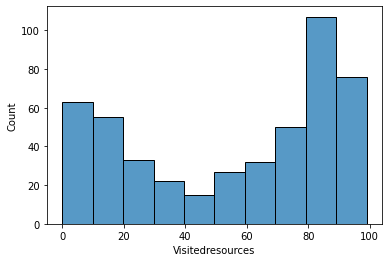
rawData.head()



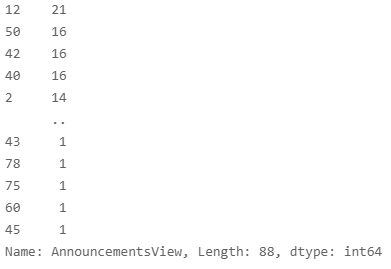
rawData['Visitedresources'].value\_counts()



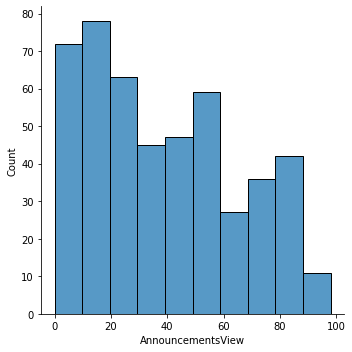
sns.histplot(data=rawData, x='Visitedresources')



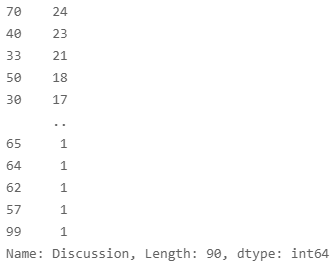
rawData.AnnouncementsView.value\_counts()



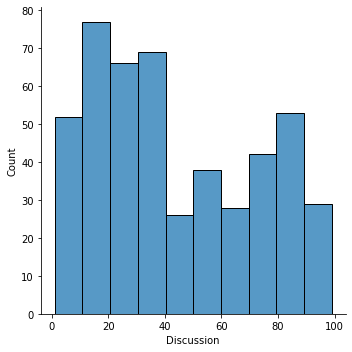
sns.displot(data=rawData, x='AnnouncementsView')



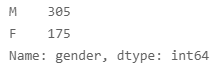
rawData.Discussion.value\_counts()



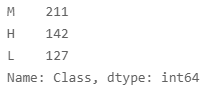
sns.displot(data=rawData, x='Discussion')



rawData['gender'].value\_counts()

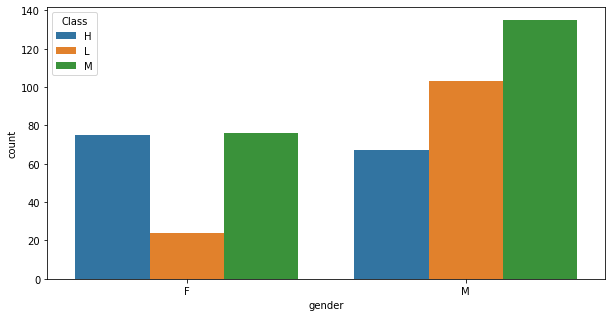


rawData['Class'].value\_counts()

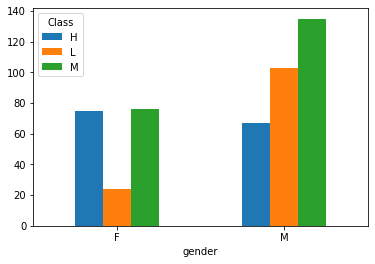


plt.figure(figsize=(10,5))

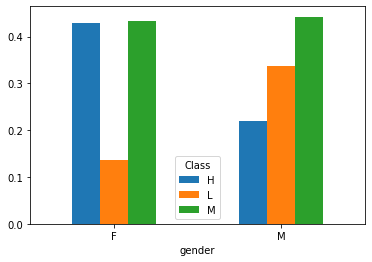
sns.countplot(data=rawData, x='gender', hue='Class', hue\_order=['H', 'L', 'M'], order=['F', 'M'])



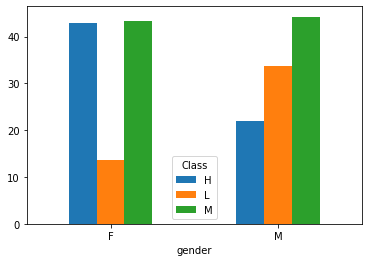
pd.crosstab(rawData['gender'],rawData['Class']).plot.bar(rot=0)



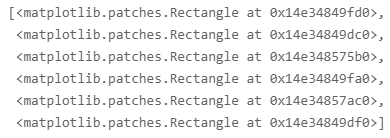
pd.crosstab(rawData['gender'],rawData['Class'], normalize='index').plot.bar(rot=0)



ax = pd.crosstab(rawData['gender'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0)



ax.patches



p=ax.patches[0]

p.get\_bbox()



p.get\_bbox().bounds



ax = pd.crosstab(rawData['gender'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(10,4))

for p in ax.patches:

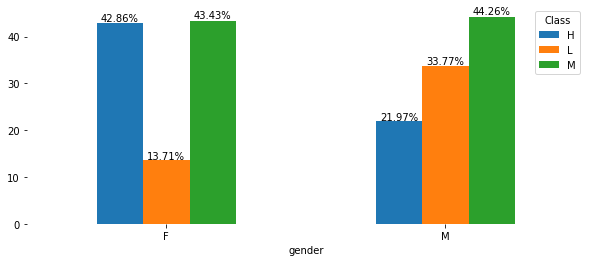
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x+width/2, height\*1.01), ha='center')

plt.sca(ax)

plt.box(False)

plt.show()



ax = pd.crosstab(rawData['gender'], rawData['Class'], normalize='index').mul(100).plot(kind='barh', rot=0, figsize=(18,4))

for p in ax.patches:

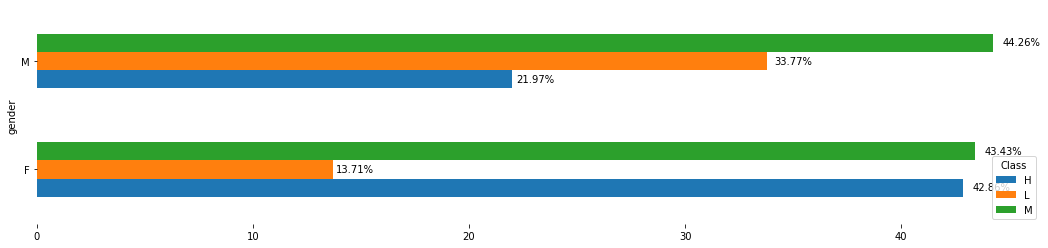
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(width), (width\*1.01, y+height/2), va='center')

plt.sca(ax)

plt.box(False)

plt.show()



ax = pd.crosstab(rawData['gender'], rawData['Class'], normalize='index').mul(100).plot(kind='barh', rot=0, figsize=(15,4))

for p in ax.patches:

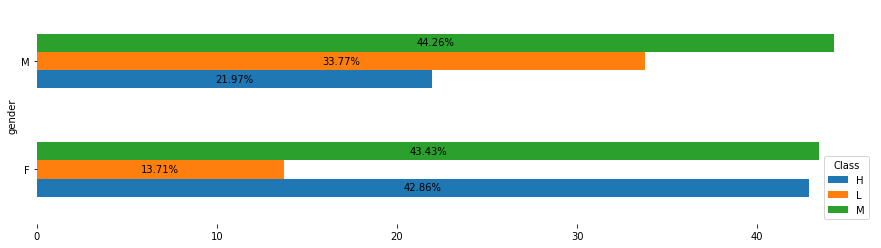
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(width), (x+width/2, y+height/2), ha='center', va='center')

plt.sca(ax)

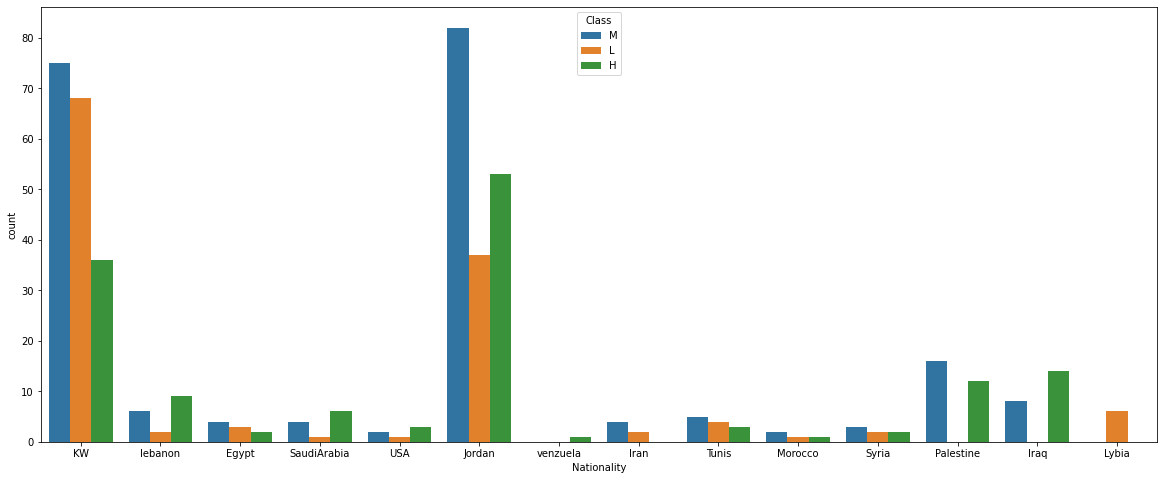
plt.box(False)

plt.show()



plt.figure(figsize=(20,8))

sns.countplot(data=rawData, x='Nationality', hue='Class')



ax = pd.crosstab(rawData['Nationality'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,8))

for p in ax.patches:

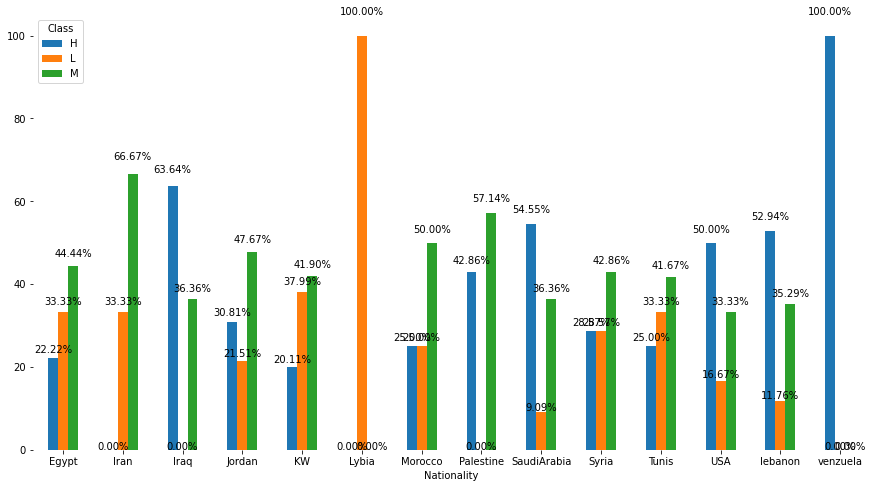
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x+width/2, height\*1.05), ha='center')

plt.sca(ax)

plt.box(False)

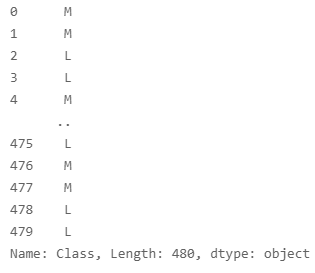
plt.show()



rawData['Relation'].unique()



rawData['Class']



ax = pd.crosstab(rawData['Relation'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,4))

for p in ax.patches:

    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x+width/2, height\*1.03), ha='center')

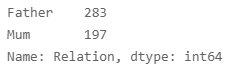
plt.sca(ax)

plt.box(False)

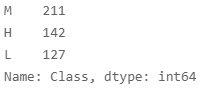
plt.show()



rawData['Relation'].value\_counts()



rawData['Class'].value\_counts()



ax = pd.crosstab(rawData['Relation'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,5))

for p in ax.patches:

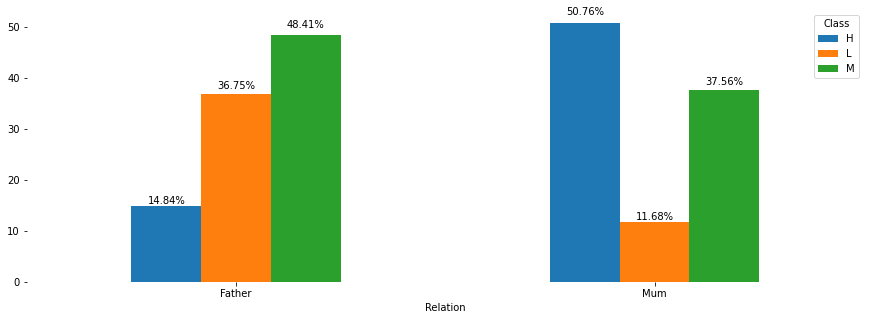
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x+width/2, height\*1.03), ha='center')

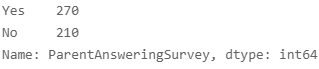
plt.sca(ax)

plt.box(False)

plt.show()



rawData['ParentAnsweringSurvey'].value\_counts()



ax = pd.crosstab( rawData['ParentAnsweringSurvey'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,5))

for p in ax.patches:

    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x + width/2, height \* 1.02), ha='center', )

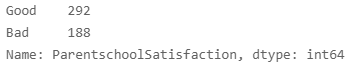
plt.sca(ax)

plt.box(False)

plt.show()



rawData['ParentschoolSatisfaction'].value\_counts()



ax = pd.crosstab( rawData['ParentschoolSatisfaction'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,5), )

for p in ax.patches:

    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x + width/2, height \* 1.02), ha='center', )

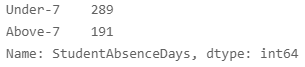
plt.sca(ax)

plt.box(False)

plt.show()



rawData['StudentAbsenceDays'].value\_counts()



ax = pd.crosstab( rawData['StudentAbsenceDays'], rawData['Class'], normalize='index').mul(100).plot(kind='bar', rot=0, figsize=(15,5), )

for p in ax.patches:

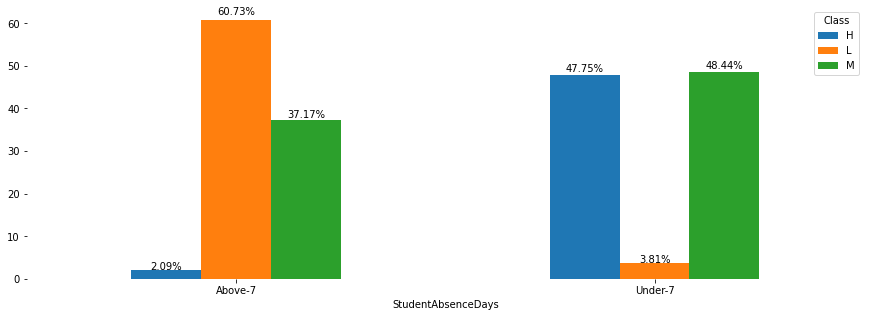
    x, y, width, height = p.get\_bbox().bounds

    ax.annotate('{:.02f}%'.format(height), (x + width/2, height \* 1.02), ha='center', )

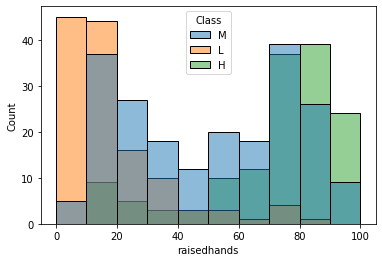
plt.sca(ax)

plt.box(False)

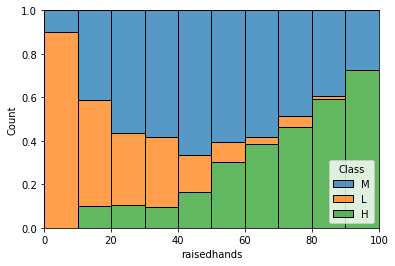
plt.show()



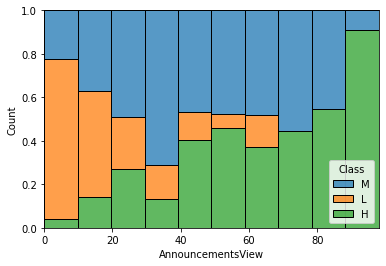
sns.histplot(data=rawData, x='raisedhands', hue='Class')



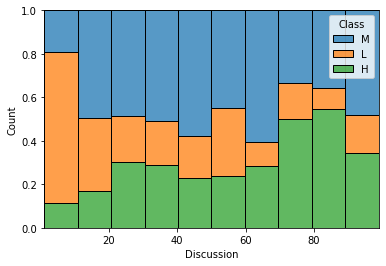
sns.histplot(data=rawData, x='raisedhands', hue='Class', multiple='fill')



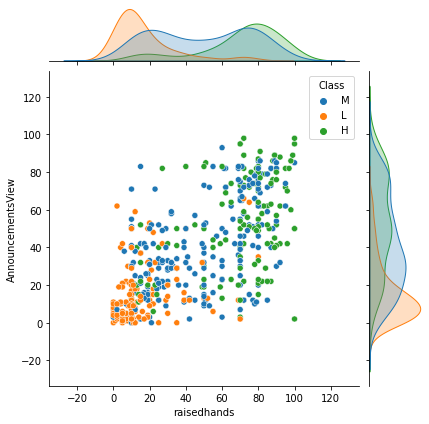
sns.histplot(data=rawData, x='AnnouncementsView', hue='Class', multiple='fill')



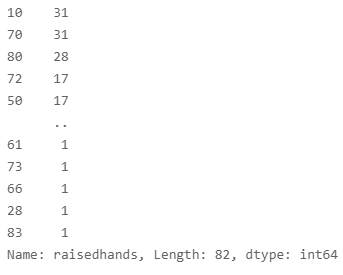
sns.histplot(data=rawData, x='Discussion', hue='Class', multiple='fill')



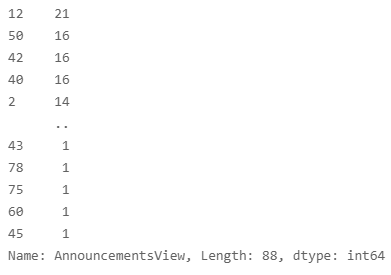
sns.jointplot(data=rawData, x='raisedhands', y='AnnouncementsView', hue='Class')



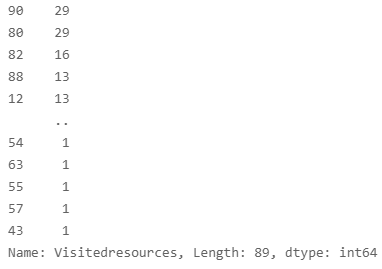
rawData['raisedhands'].value\_counts()



rawData['AnnouncementsView'].value\_counts()

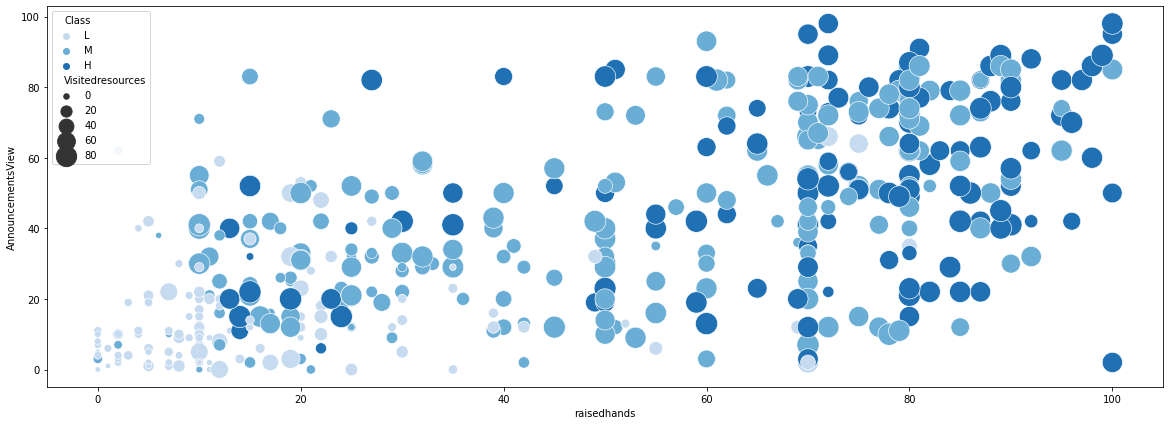


rawData['Visitedresources'].value\_counts()



plt.figure(figsize=(20,7))

sns.scatterplot(data=rawData, x='raisedhands', y='AnnouncementsView', size='Visitedresources', hue='Class', hue\_order=['L', 'M', 'H'], sizes=(30,500), palette = "Blues")

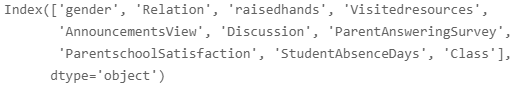


3. Simple Preprocessing

drop\_columns = ['Nationality', 'PlaceofBirth', 'StageID', 'GradeID', 'SectionID', 'Topic', 'Semester']

x = rawData.drop(columns=drop\_columns)

x.columns



cat\_columns = ['gender', 'Relation', 'ParentAnsweringSurvey', 'ParentschoolSatisfaction', 'StudentAbsenceDays']

x = pd.get\_dummies(data=x, columns=cat\_columns)

y = x['Class']

x = x.drop(columns='Class')

4. Modelling and Parameter Tuning Demonstrations

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, GridSearchCV

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3)

4.1. Generalized Linear Model

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

folds = StratifiedKFold(n\_splits=5, random\_state=True, shuffle=True)

model = LogisticRegression(solver='saga', penalty='elasticnet')

params = dict(

    C=[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000],

    class\_weight = ['balanced', None],

    multi\_class = ['auto', 'ovr', 'multinomial'],

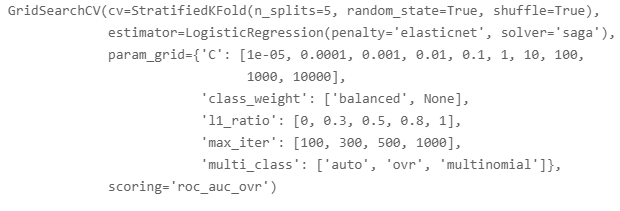
    max\_iter = [100, 300, 500, 1000],

    l1\_ratio = [0, 0.3, 0.5, 0.8, 1]

)

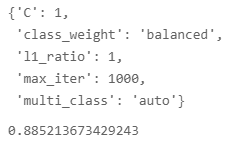
search = GridSearchCV(model, cv=folds, param\_grid=params, scoring='roc\_auc\_ovr')

search.fit(x\_train, y\_train)



display(search.best\_params\_)

display(search.best\_score\_)

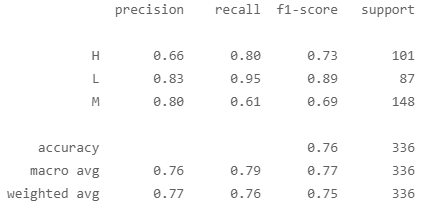


bestModel = search.best\_estimator\_

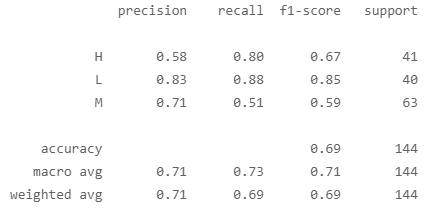
yhat\_train = bestModel.predict(x\_train)

yhat\_test = bestModel.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



plt.figure(figsize=(15,5))

plt.xticks(rotation=80)

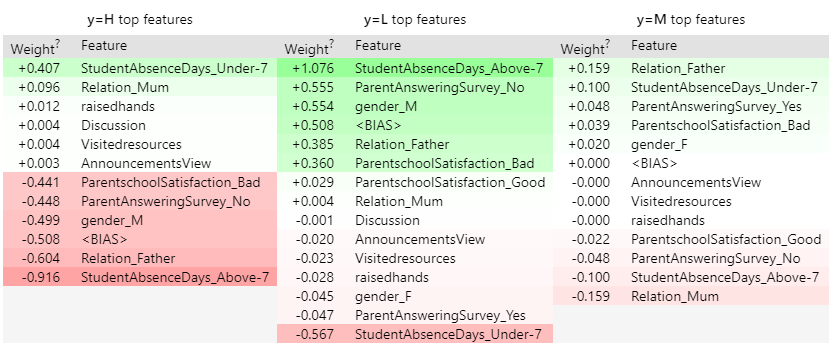
plt.bar(x.columns, bestModel.coef\_[0])



import eli5

from eli5.sklearn import PermutationImportance

eli5.show\_weights(bestModel, feature\_names = x.columns.tolist())



4.2. Support Vector Machine

from sklearn.svm import SVC

model = SVC(probability=True)

params = dict(

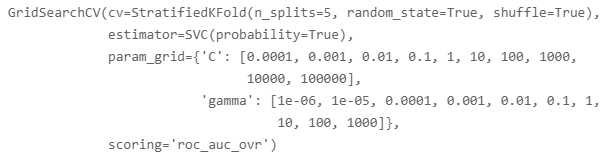
    C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000],

    gamma = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

)

search = GridSearchCV(model, param\_grid=params, cv=folds, scoring='roc\_auc\_ovr')

search.fit(x\_train, y\_train)



display(search.best\_params\_)

display(search.best\_score\_)



bestModel = search.best\_estimator\_

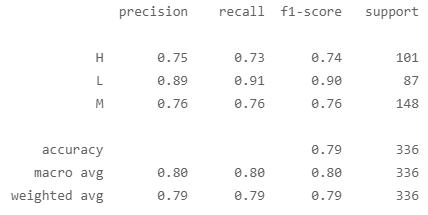
bestModel



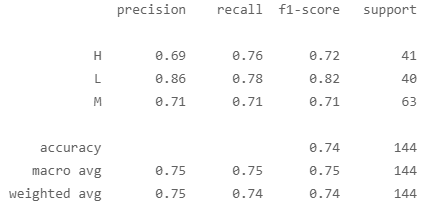
yhat\_train = bestModel.predict(x\_train)

yhat\_test = bestModel.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))

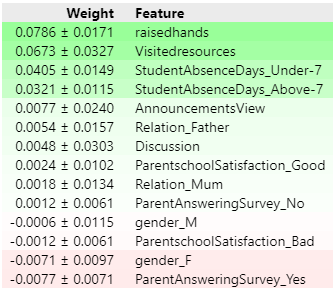


print(classification\_report(y\_test, yhat\_test))



perm = PermutationImportance(bestModel, random\_state=123).fit(x\_train, y\_train)

eli5.show\_weights(perm, feature\_names = x\_train.columns.tolist())



4.3. Random Forest

from sklearn.ensemble import RandomForestClassifier

from sklearn import preprocessing

encoder = preprocessing.LabelEncoder()

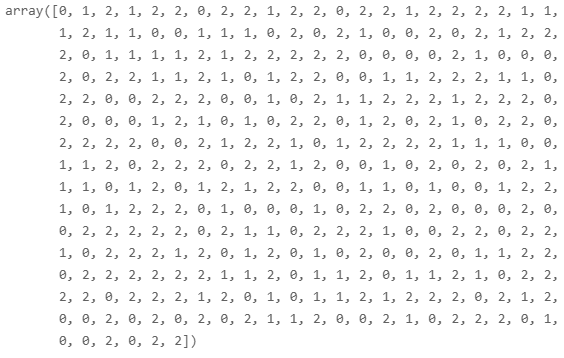
y\_train\_label\_encode = encoder.fit\_transform(y\_train)

y\_test\_label\_encode = encoder.fit\_transform(y\_test)

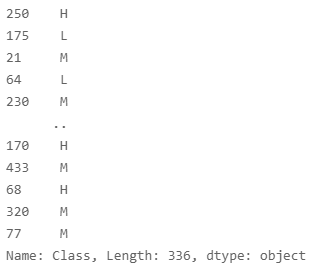
encoder.classes\_



y\_train\_label\_encode



y\_train



model = RandomForestClassifier()

params = dict(

    n\_estimators = [100, 300],

    criterion = ['gini', 'entropy'],

    max\_depth = [None, 3, 5, 7, 11, 17, 19, 21],

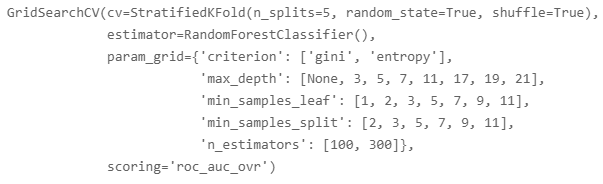
    min\_samples\_split = [2,3,5,7,9,11],

    min\_samples\_leaf = [1,2,3,5,7,9,11]

)

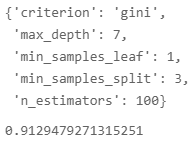
search = GridSearchCV(estimator = model, param\_grid=params, cv=folds, scoring='roc\_auc\_ovr')

search.fit(x\_train, y\_train)



display(search.best\_params\_)

display(search.best\_score\_)

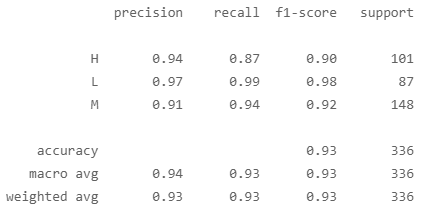


bestModel = search.best\_estimator\_

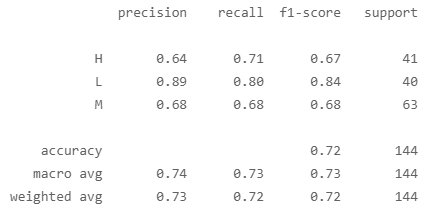
yhat\_train = bestModel.predict(x\_train)

yhat\_test = bestModel.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))

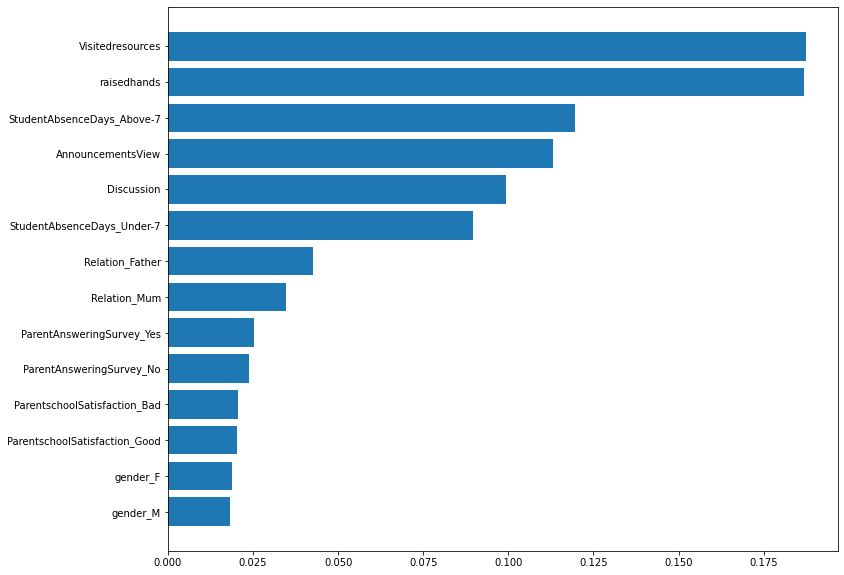


importance = bestModel.feature\_importances\_

indices = np.argsort(importance)

plt.figure(figsize=(12,10))

plt.barh(x\_train.columns[indices], importance[indices], align='center')



4.4. XGBoost

import xgboost as xgb

model = xgb.XGBClassifier()

params = dict(

    n\_estimators = [100, 300, 500, 1000],

    max\_depth = [None, 3, 5, 7, 11, 17, 19, 21],

    colsample\_bytree = [0.2, 0.5, 0.7, 1],

    reg\_lambda = [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]

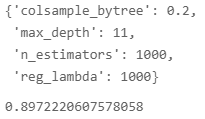
)

search = GridSearchCV(estimator= model, param\_grid= params, cv=folds, scoring = 'roc\_auc\_ovr')

search.fit(x\_train, y\_train)

display(search.best\_params\_)

display(search.best\_score\_)

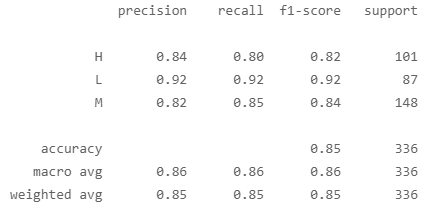


bestModel = search.best\_estimator\_

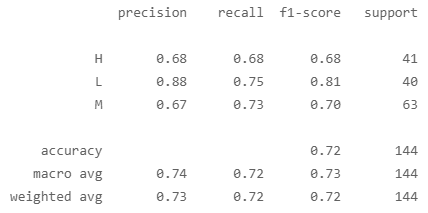
yhat\_train = bestModel.predict(x\_train)

yhat\_test = bestModel.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))

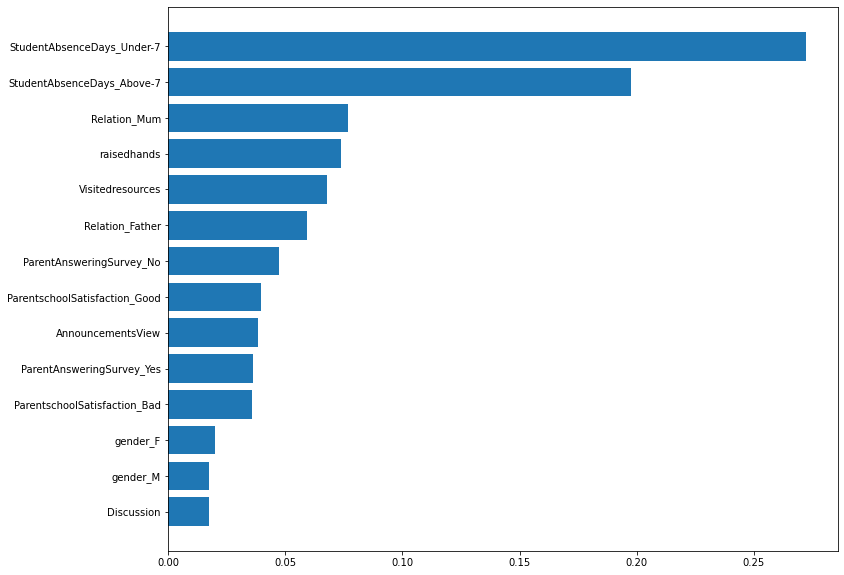


importance = bestModel.feature\_importances\_

indices = np.argsort(importance)

plt.figure(figsize=(12,10))

plt.barh(x\_train.columns[indices], importance[indices], align='center')



4.4. LightGBM

import lightgbm as lgb

lgbModel = lgb.LGBMClassifier()

params = {

    'n\_estimators': [50, 100, 300],

    'num\_leaves': [32, 64, 128],

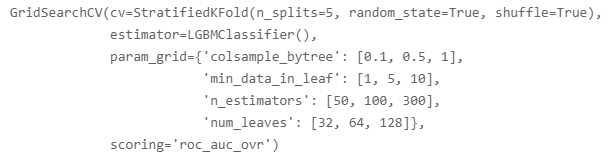
    'colsample\_bytree': [0.1, 0.5, 1],

    'min\_data\_in\_leaf': [1,5,10]

}

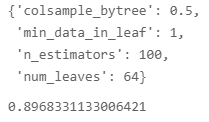
search = GridSearchCV(lgbModel, param\_grid=params, cv=folds, scoring='roc\_auc\_ovr')

search.fit(x\_train, y\_train)



display(search.best\_params\_)

display(search.best\_score\_)

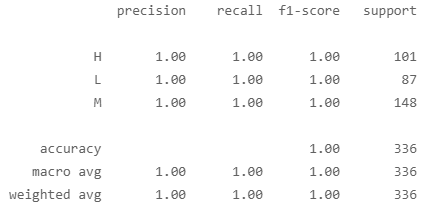


bestModel = search.best\_estimator\_

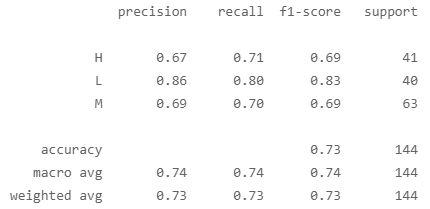
yhat\_train = bestModel.predict(x\_train)

yhat\_test = bestModel.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



4.4. CatBoost

#In contrast to xgboost or lightgbm, catboost is less sensitive to parameter tuning.

#Generally, the internal algorithm of catboost autonomously processes the overfitting and polymorphism problems,

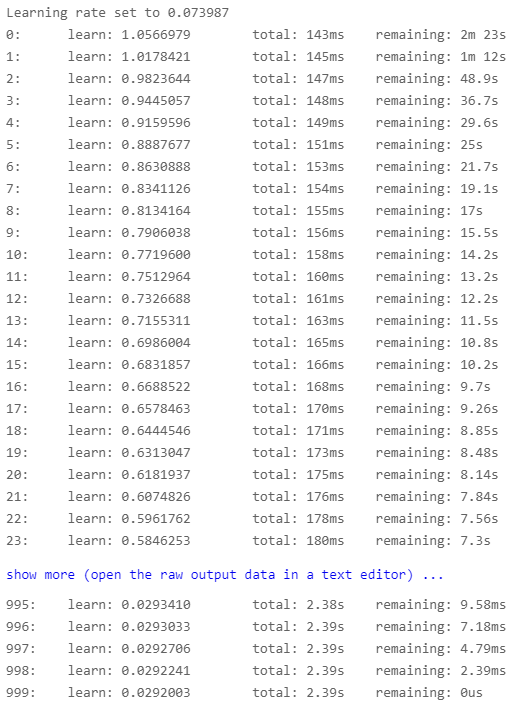
#which is why parameter tuning is not a must for catboost models.

import catboost as cb

from sklearn.model\_selection import cross\_val\_score

model = cb.CatBoostClassifier()

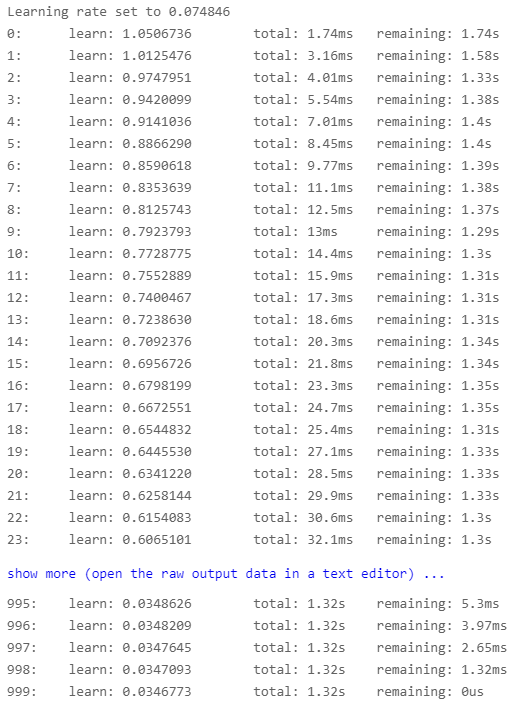
result = cross\_val\_score(model, x\_train, y\_train, cv=folds)



result.mean()



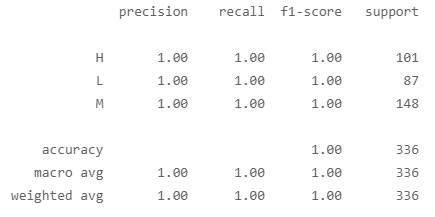
model = cb.CatBoostClassifier().fit(x\_train, y\_train)



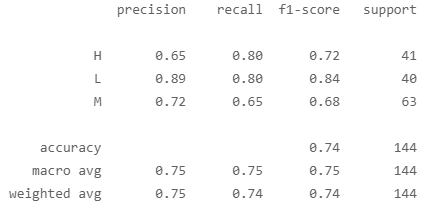
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))

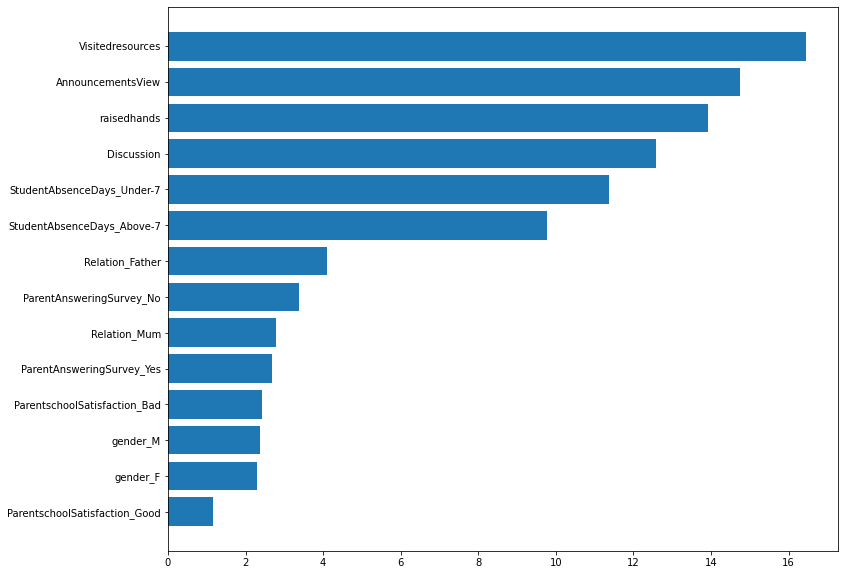


importance = model.feature\_importances\_

indices = np.argsort(importance)

plt.figure(figsize=(12,10))

plt.barh(x\_train.columns[indices], importance[indices], align='center')



4.5. Neural Network

from sklearn.neural\_network import MLPClassifier

nnModel = MLPClassifier()

params = dict(

    hidden\_layer\_sizes = [ (2048), (16,16,16,16), (128,16), (256,64,8) ],

    activation = ['logistic', 'tanh', 'relu'],

    alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

    learning\_rate = ['constant', 'invscaling', 'adaptive'],

    max\_iter = [200, 1000, 10000]

)

search = GridSearchCV(nnModel, param\_grid = params, cv=folds)

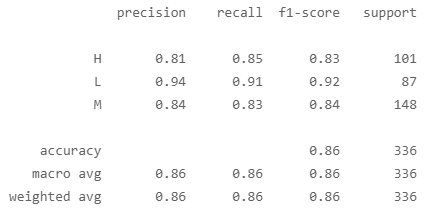
search.fit(x\_train, y\_train)

nnModel = MLPClassifier(hidden\_layer\_sizes=(4096, 512, 8), activation='logistic', alpha=0.001, learning\_rate='invscaling', max\_iter=100000).fit(x\_train, y\_train)

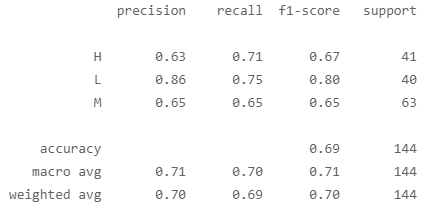
yhat\_train = nnModel.predict(x\_train)

yhat\_test = nnModel.predict(x\_test)

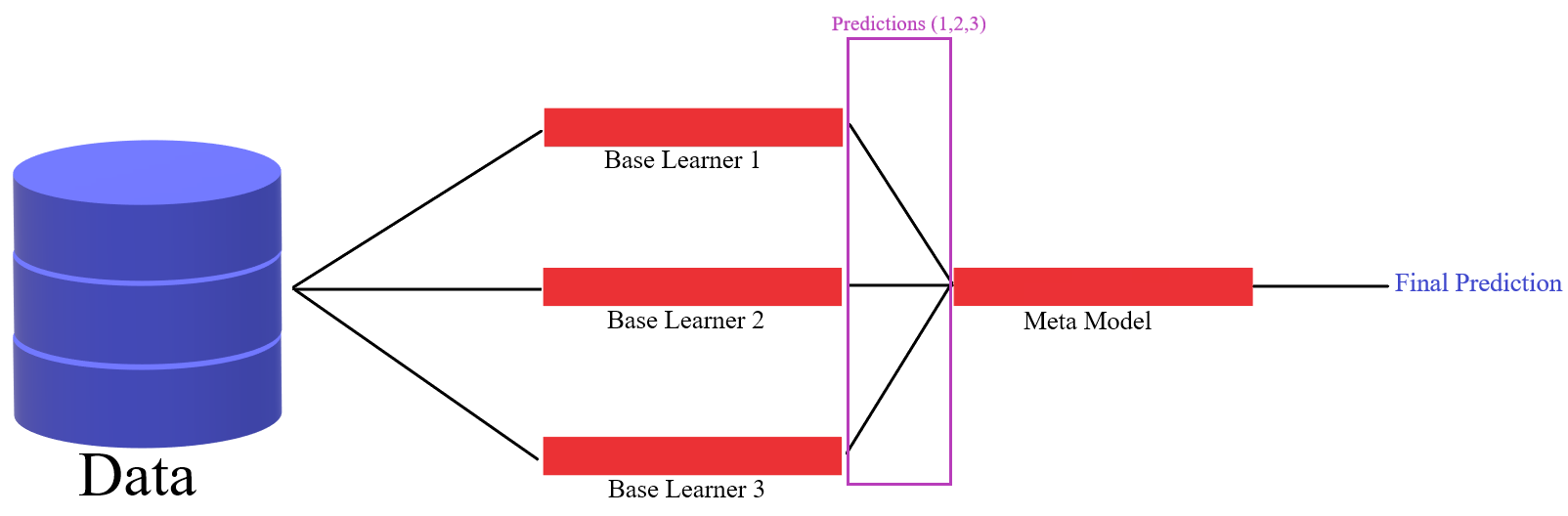
print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



4.6. Stacking Models



from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import StackingClassifier, RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split, StratifiedShuffleSplit

estimator = [

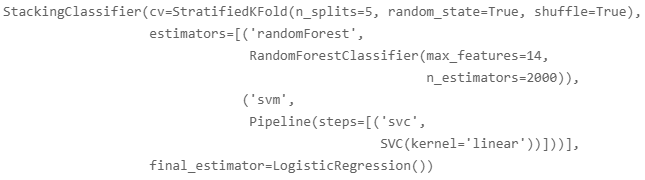
    ('randomForest', RandomForestClassifier(n\_estimators=2000, max\_features=14)),

    ('svm', make\_pipeline(SVC(kernel='linear')))

]

model = StackingClassifier(estimators=estimator, final\_estimator=LogisticRegression(), cv=folds)

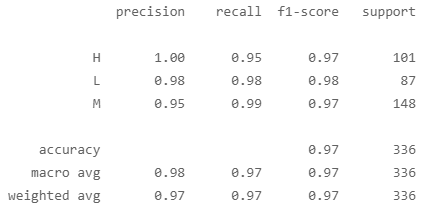
model.fit(x\_train, y\_train)



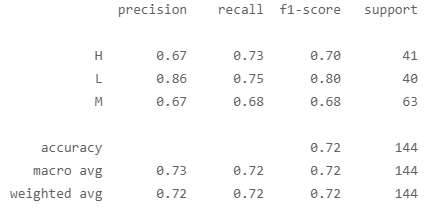
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



layer1 = [

  ('svm', SVC(C=1000, gamma=0.0001) ),

  ('knn', KNeighborsClassifier(n\_neighbors=4) ),

  ('randomForest', RandomForestClassifier(n\_estimators=2000, criterion='entropy', max\_depth=None, min\_samples\_leaf=3,) ),

]

layer2 = [

  ('xgb', xgb.XGBClassifier(colsample\_bytree=0.2, max\_depth=5, n\_estimators=2000, reg\_lambda=100) ),

  ('lgb', lgb.LGBMClassifier(colsample\_bytree=0.5, min\_data\_in\_leaf=5, n\_estimators=2000, num\_leaves=32) ),

  ('cb', cb.CatBoostClassifier( verbose = 0 ) ),

]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, train\_size = 0.8 )

folds = StratifiedKFold(n\_splits=10, shuffle=True)

layer2 = StackingClassifier( estimators=layer2, final\_estimator=LogisticRegression(), cv=folds )

model = StackingClassifier( estimators=layer1, final\_estimator=layer2, cv=folds )

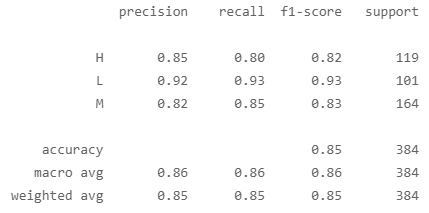
model.fit( x\_train, y\_train )



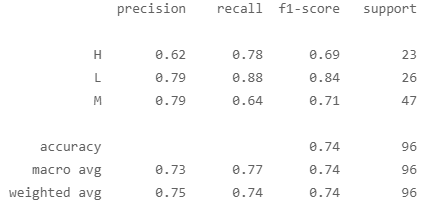
yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

print(classification\_report(y\_train, yhat\_train))



print(classification\_report(y\_test, yhat\_test))



4.6. Keras

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

y\_label\_encode = encoder.fit\_transform(y)

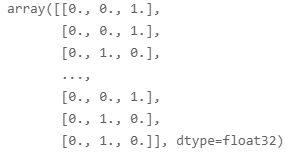
encoder.classes\_



import tensorflow as tf

y\_categorical = tf.keras.utils.to\_categorical(y\_label\_encode, 3)

y\_categorical



x = tf.constant(x, dtype=tf.float64)

x.shape

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(8192, activation='relu', input\_shape=x.shape))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(1024, activation='relu'))

model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(128, activation='relu'))

model.add(tf.keras.layers.Dense(3, activation='softmax'))

model.compile(

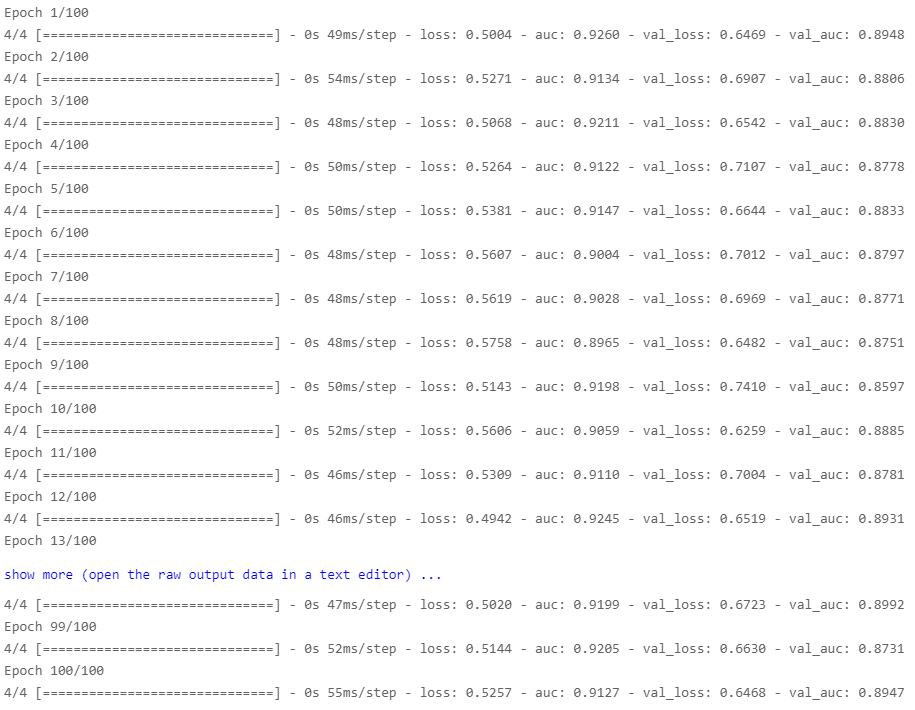
    loss='categorical\_crossentropy',

    optimizer=tf.keras.optimizers.Adam(0.01),

    metrics=['AUC']

)

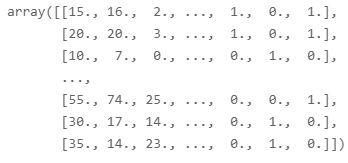
hist = model.fit(x, y\_categorical, epochs=100, validation\_split=0.2, batch\_size=100)



4.7. CNN

x\_array = x.numpy()

x\_array



x\_reshape = x\_array.reshape(-1, 14, 1)

print( x\_reshape.shape )

print( y\_categorical.shape )



model = tf.keras.models.Sequential()

model.add( tf.keras.layers.Conv1D(64, 3, activation='relu', input\_shape=(14,1)))

model.add( tf.keras.layers.MaxPool1D(3) )

model.add( tf.keras.layers.Dropout(0.5) )

model.add( tf.keras.layers.Flatten() )

model.add( tf.keras.layers.Dense(128, activation='relu') )

model.add( tf.keras.layers.Dropout(0.5) )

model.add( tf.keras.layers.Dense(3, activation='softmax') )

model.compile(

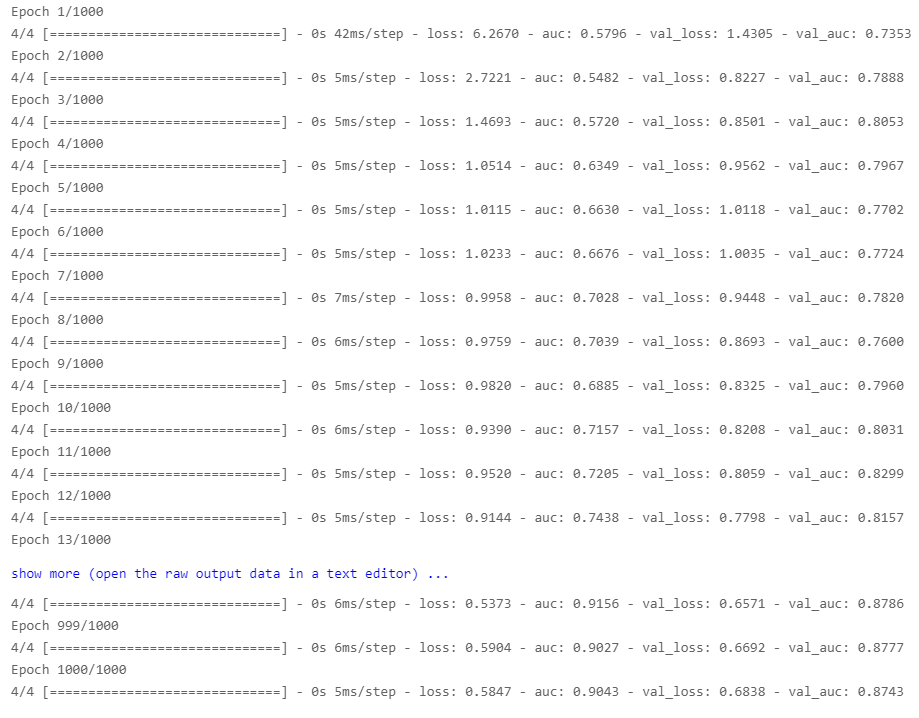
    loss='categorical\_crossentropy',

    optimizer = tf.keras.optimizers.Adam(0.01),

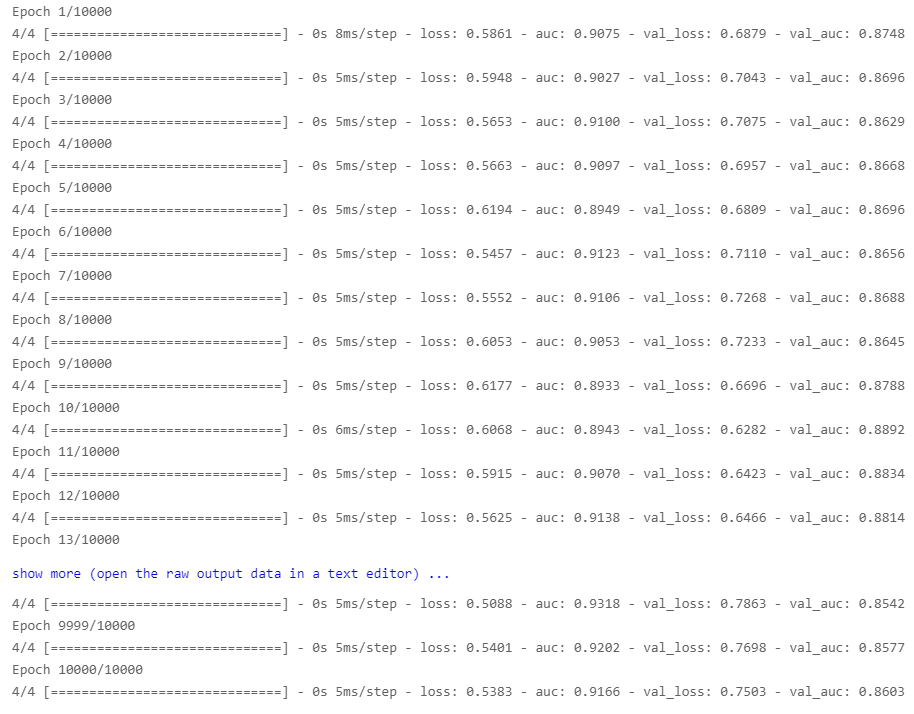
    metrics = ['AUC']

)

hist = model.fit(x\_reshape, y\_categorical, epochs = 1000, validation\_split=0.2, batch\_size=100 )



hist = model.fit(x\_reshape, y\_categorical, epochs = 10000, validation\_split=0.2, batch\_size=100 )



plt.plot(hist.history['loss'], label='train loss(errors)')

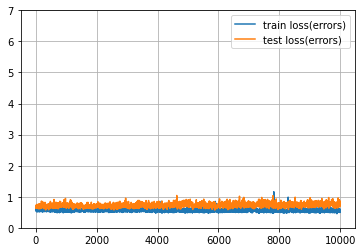
plt.plot(hist.history['val\_loss'], label='test loss(errors)')

plt.legend()

plt.ylim(0,7)

plt.grid()

plt.show()



plt.plot(hist.history['loss'], label='train loss(errors)')

plt.plot(hist.history['val\_loss'], label='test loss(errors)')

plt.legend()

plt.grid()

plt.show()



plt.plot(hist.history['auc'], label='train auc')

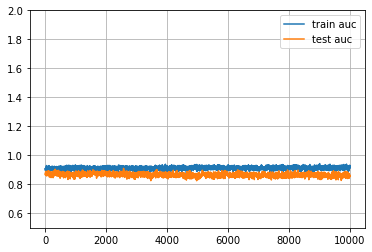
plt.plot(hist.history['val\_auc'], label='test auc')

plt.legend()

plt.ylim(0.5, 2)

plt.grid()

plt.show()



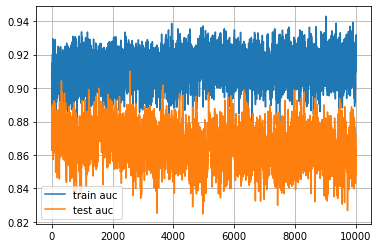
plt.plot(hist.history['auc'], label='train auc')

plt.plot(hist.history['val\_auc'], label='test auc')

plt.legend()

plt.grid()

plt.show()



5. Appendix (interpreting Encoded Y variable vs. original Y variable: comparison)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

rawData = pd.read\_csv('Your File Path\\xAPI-Edu-Data.csv')

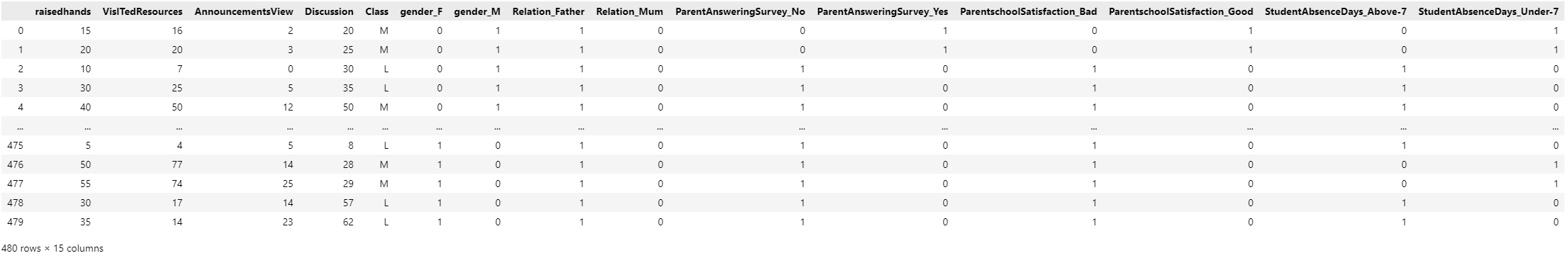
drop\_columns = ['NationalITy', 'PlaceofBirth', 'StageID', 'GradeID', 'SectionID', 'Topic', 'Semester', ]

x = rawData.drop(columns=drop\_columns)

cat\_columns = ['gender', 'Relation', 'ParentAnsweringSurvey', 'ParentschoolSatisfaction', 'StudentAbsenceDays', ]

x = pd.get\_dummies(data=x, columns=cat\_columns)

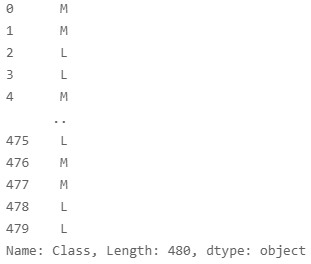
x



y = x['Class']

x = x.drop(columns='Class')

y



from sklearn.model\_selection import train\_test\_split, StratifiedKFold, GridSearchCV

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, train\_size=0.7)

from sklearn.linear\_model import LogisticRegression

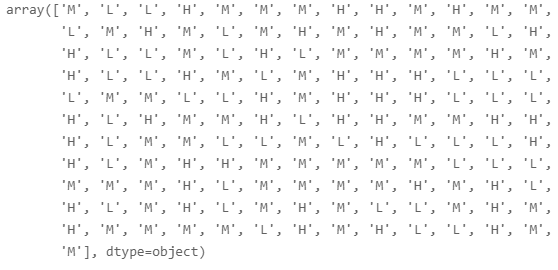
from sklearn.metrics import classification\_report

folds = StratifiedKFold(n\_splits=5, random\_state=True, shuffle=True)

model = LogisticRegression()

model.fit(x\_train, y\_train)

model.predict(x\_test)

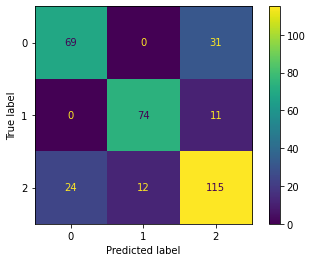


from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

yhat\_train = model.predict(x\_train)

yaht\_test = model.predict(x\_test)

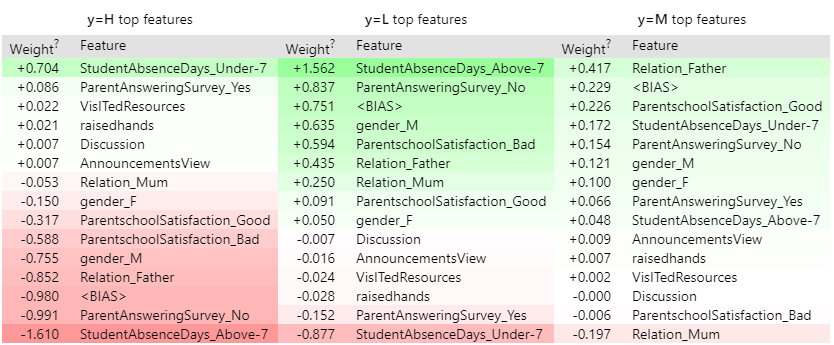
ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train, yhat\_train),).plot()



import eli5

from eli5.sklearn import PermutationImportance

eli5.show\_weights(model, feature\_names = x.columns.tolist())



#Encoded Y variable

from sklearn import preprocessing

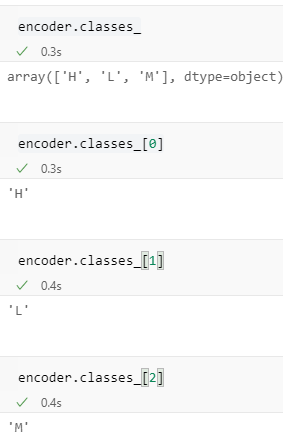
encoder = preprocessing.LabelEncoder()

y\_train\_label\_encode = encoder.fit\_transform(y\_train)

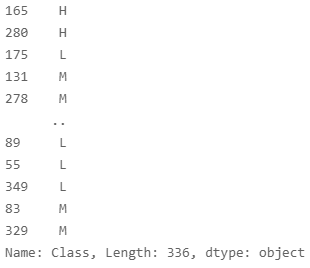
y\_test\_label\_encode = encoder.fit\_transform(y\_test)

model = LogisticRegression(max\_iter=100000000).fit(x\_train, y\_train\_label\_encode)

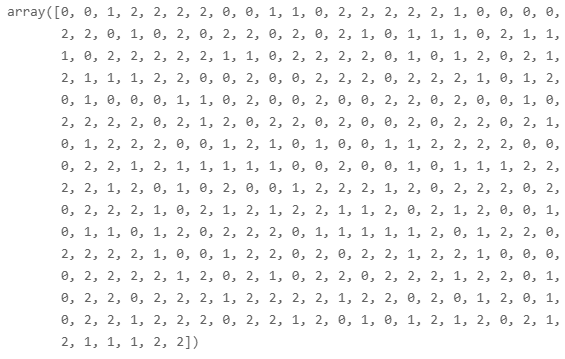
encoder.classes\_



y\_train



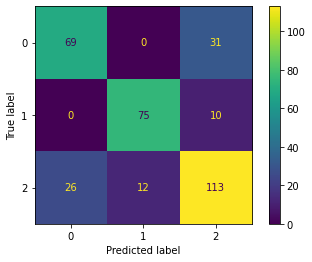
y\_train\_label\_encode



yhat\_train = model.predict(x\_train)

yhat\_test = model.predict(x\_test)

ConfusionMatrixDisplay(confusion\_matrix=confusion\_matrix(y\_train\_label\_encode, yhat\_train),).plot()



eli5.show\_weights(model, feature\_names = x.columns.tolist())

