36106-25AU-AT2-25589351-experiment-4

April 25, 2025

1 Experiment Notebook

1.1 0. Setup Environment

1.1.1 0.a Install Environment and Mandatory Packages

[1]:
```
# Do not modify this code
!pip install -q utstd

from utstd.folders import *
from utstd.ipyrenders import *

at = AtFolder(
    course_code=36106,
    assignment="AT2",
)
at.run()
```

```
                          0.0/1.6 MB
? eta -:--:--
                       0.1/1.6
MB 4.9 MB/s eta 0:00:01
                       0.8/1.6
MB 11.0 MB/s eta 0:00:01
                       1.6/1.6 MB
16.4 MB/s eta 0:00:01
                       1.6/1.6 MB 11.7
MB/s eta 0:00:00
Mounted at /content/gdrive

You can now save your data files in:
/content/gdrive/MyDrive/36106/assignment/AT2/data
```

1.1.2 0.b Disable Warnings Messages

```
[ ]: # Do not modify this code
     import warnings
     warnings.simplefilter(action='ignore')
```

1.1.3 0.c Install Additional Packages

If you are using additional packages, you need to install them here using the command: `! pip install <package_name>`

```
[ ]: # <Student to fill this section>
```

1.1.4 0.d Import Packages

```
[2]: # <Student to fill this section>
     import pandas as pd
     import altair as alt
     from sklearn.model_selection import cross_val_score, cross_val_predict
     from sklearn.metrics import recall_score, f1_score, classification_report,␣
      ↪confusion_matrix, ConfusionMatrixDisplay
     from sklearn.model_selection import GridSearchCV
     from sklearn.preprocessing import StandardScaler
     from sklearn.preprocessing import LabelEncoder
     import matplotlib.pyplot as plt
```

1.2 A. Project Description

```
[3]: # <Student to fill this section>
     student_name = "Fatemeh Elyasifar"
     student_id = "25589351"
```

```
[ ]: # Do not modify this code
     print_tile(size="h1", key='student_name', value=student_name)
```

```
<IPython.core.display.HTML object>
```

```
[ ]: # Do not modify this code
     print_tile(size="h1", key='student_id', value=student_id)
```

```
<IPython.core.display.HTML object>
```

```
[4]: print("Student Name:", student_name)
     print("Student ID:", student_id)
```

```
Student Name: Fatemeh Elyasifar
Student ID: 25589351
```

[5]:
```
# <Student to fill this section>
business_objective = """

The objective is to develop a reliable and interpretable machine learning model␣
    ↪to predict student performance at the end of the semester
using academic, behavioral, and demographic data. The model aims to help␣
    ↪university staff, student support officers, and academic advisors identify␣
    ↪at-risk students (those with poor or average performance),
with a target F1-score and recall above 80%, enabling timely outreach and␣
    ↪efficient allocation of support services.
The focus is on improving recall for underperforming students while maintaining␣
    ↪balanced performance.
The project also identifies key predictors and includes feature tuning to␣
    ↪enhance accuracy and relevance.

Accurate predictions help the university support underperforming students,␣
    ↪improve outcomes, and reduce dropout rates.
Inaccurate results risk missing students in need or misusing limited resources.
Therefore, maintaining a strong balance between precision and recall is␣
    ↪essential to ensure the model is both effective and trustworthy.
High recall is especially important to capture as many at-risk students as␣
    ↪possible.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='business_objective', value=business_objective)
```

```
<IPython.core.display.HTML object>
```

[6]:
```
print("Business Objective:", business_objective)
```

```
Business Objective:

The objective is to develop a reliable and interpretable machine learning model
to predict student performance at the end of the semester
using academic, behavioral, and demographic data. The model aims to help
university staff, student support officers, and academic advisors identify at-
risk students (those with poor or average performance),
with a target F1-score and recall above 80%, enabling timely outreach and
efficient allocation of support services.
The focus is on improving recall for underperforming students while maintaining
balanced performance.
The project also identifies key predictors and includes feature tuning to
enhance accuracy and relevance.
```

```
Accurate predictions help the university support underperforming students,
improve outcomes, and reduce dropout rates.
Inaccurate results risk missing students in need or misusing limited resources.
Therefore, maintaining a strong balance between precision and recall is
essential to ensure the model is both effective and trustworthy.
High recall is especially important to capture as many at-risk students as
possible.
```

1.3 B. Experiment Description

[7]:
```
# Do not modify this code
experiment_id = "4"
print_tile(size="h1", key='experiment_id', value=experiment_id)
```

```
<IPython.core.display.HTML object>
```

[8]:
```
print("Experiment ID:", experiment_id)
```

```
Experiment ID: 4
```

[9]:
```
# <Student to fill this section>
experiment_hypothesis = """
Present the hypothesis you want to test, the question you want to answer or the␣
  ↪insight you are seeking.
Explain the reasons why you think it is worthwhile considering it

Adding new features related to academic status and attendance, along with␣
  ↪applying label encoding, will enhance the Extra Trees model's ability to␣
  ↪detect at-risk students.
Enriching the dataset with meaningful indicators and reducing categorical␣
  ↪dimensionality can improve interpretability and recall without sacrificing␣
  ↪overall performance.

These transformations will help the Extra Trees model capture more informative␣
  ↪patterns, making it more effective and reliable for early academic␣
  ↪intervention.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='experiment_hypothesis', value=experiment_hypothesis)
```

```
<IPython.core.display.HTML object>
```

[10]:
```
print("Experiment Hypothesis:", experiment_hypothesis)
```

```
Experiment Hypothesis:
Present the hypothesis you want to test, the question you want to answer or the
insight you are seeking.
Explain the reasons why you think it is worthwhile considering it

Adding new features related to academic status and attendance, along with
applying label encoding, will enhance the Extra Trees model's ability to detect
at-risk students.
Enriching the dataset with meaningful indicators and reducing categorical
dimensionality can improve interpretability and recall without sacrificing
overall performance.

These transformations will help the Extra Trees model capture more informative
patterns, making it more effective and reliable for early academic intervention.
```

[11]:
```
# <Student to fill this section>
experiment_expectations = """
Detail what will be the expected outcome of the experiment. If possible,␣
  ↪estimate the goal you are expecting.
List the possible scenarios resulting from this experiment.

The updated Extra Trees model is expected to achieve improved or at least␣
  ↪maintained performance, with a weighted F1 score of 0.76 or higher on the␣
  ↪validation set.
By introducing features like academic_status_level and attendance_flag, the␣
  ↪model may gain better context for identifying students at risk, especially␣
  ↪in underperforming classes.
Label encoding is expected to reduce dimensionality while preserving␣
  ↪categorical information, potentially improving generalization.

Possible scenarios include:
- Improved recall and F1-scores for classe 1 due to the added features, leading␣
  ↪to a stronger business impact.
- Similar overall performance but with improved interpretability and feature␣
  ↪importance insights.
- Slight performance drop if the new features introduce noise or label encoding␣
  ↪reduces model flexibility, suggesting the need for alternative encoding or␣
  ↪feature selection.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='experiment_expectations',␣
  ↪value=experiment_expectations)
```

```
<IPython.core.display.HTML object>
```

```
[12]: print("Experiment Expectations:", experiment_expectations)
```

```
Experiment Expectations:
Detail what will be the expected outcome of the experiment. If possible,
estimate the goal you are expecting.
List the possible scenarios resulting from this experiment.

The updated Extra Trees model is expected to achieve improved or at least
maintained performance, with a weighted F1 score of 0.76 or higher on the
validation set.
By introducing features like academic_status_level and attendance_flag, the
model may gain better context for identifying students at risk, especially in
underperforming classes.
Label encoding is expected to reduce dimensionality while preserving categorical
information, potentially improving generalization.

Possible scenarios include:
- Improved recall and F1-scores for classe 1 due to the added features, leading
to a stronger business impact.
- Similar overall performance but with improved interpretability and feature
importance insights.
- Slight performance drop if the new features introduce noise or label encoding
reduces model flexibility, suggesting the need for alternative encoding or
feature selection.
```

1.4 C. Data Understanding

```
[ ]: # Do not modify this code
     # Load training data
     try:
       X_train = pd.read_csv(at.folder_path / 'X_train.csv')
       y_train = pd.read_csv(at.folder_path / 'y_train.csv')

       X_val = pd.read_csv(at.folder_path / 'X_val.csv')
       y_val = pd.read_csv(at.folder_path / 'y_val.csv')

       X_test = pd.read_csv(at.folder_path / 'X_test.csv')
       y_test = pd.read_csv(at.folder_path / 'y_test.csv')
     except Exception as e:
       print(e)
```

```
[ ]: try:
       df_eng = pd.read_csv(at.folder_path / 'df_eng.csv')

     except Exception as e:
```

```
print(e)
```

1.5 D. Feature Selection

[ ]:
```
# <Student to fill this section>

features_list = df_eng.columns.tolist()
```

[13]:
```
# <Student to fill this section>
feature_selection_explanations = """
Provide a rationale on why you are selected these features but also why you␣
↪decided to remove other ones

These features give a clear picture of each student’s background, behavior, and␣
↪academics, and were picked because they likely affect performance.
They should help the model make better predictions.
Less relevant features were removed based on their low impact on the target␣
↪variable.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_selection_explanations',␣
↪value=feature_selection_explanations)
```

```
<IPython.core.display.HTML object>
```

[14]:
```
print("Feature Selection Explanations:", feature_selection_explanations)
```

```
Feature Selection Explanations:
Provide a rationale on why you are selected these features but also why you
decided to remove other ones

These features give a clear picture of each student’s background, behavior, and
academics, and were picked because they likely affect performance.
They should help the model make better predictions.
Less relevant features were removed based on their low impact on the target
variable.
```

1.6 E. Feature Engineering

1.6.1 E.1 New Feature "academic_status_level"

```
[ ]: # <Student to fill this section>
df_eng['academic_status_level'] = df_eng['on_probation'] + 2 *␣
  ↪df_eng['is_suspended']
df_eng.drop(columns=['on_probation', 'is_suspended'], inplace=True)
```

```
[15]: # <Student to fill this section>
feature_engineering_1_explanations = """
Provide some explanations on why you believe it is important to create this␣
  ↪feature and its impacts

The new feature 'academic_status_level' combines 'on_probation' and␣
  ↪'is_suspended' into a single numeric scale to reflect the severity of a␣
  ↪student's academic standing.
This transformation simplifies the representation of academic risk into one␣
  ↪interpretable metric:
0 = no issue, 1 = on probation, 2 = suspended, 3 = both.

By encoding this relationship numerically, the model can better capture the␣
  ↪influence of academic warnings on performance,
while reducing dimensionality and potential multicollinearity between the␣
  ↪original binary columns.
"""
```

```
[ ]: # Do not modify this code
print_tile(size="h3", key='feature_engineering_1_explanations',␣
  ↪value=feature_engineering_1_explanations)
```

```
<IPython.core.display.HTML object>
```

```
[16]: print("Feature Engineering Explanations:", feature_engineering_1_explanations)
```

```
Feature Engineering Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The new feature 'academic_status_level' combines 'on_probation' and
'is_suspended' into a single numeric scale to reflect the severity of a
student's academic standing.
This transformation simplifies the representation of academic risk into one
interpretable metric:
0 = no issue, 1 = on probation, 2 = suspended, 3 = both.

By encoding this relationship numerically, the model can better capture the
influence of academic warnings on performance,
while reducing dimensionality and potential multicollinearity between the
original binary columns.
```

1.6.2 E.2 New Feature "attendance_flag"

[ ]:
```
# <Student to fill this section>
df_eng['attendance_flag'] = (df_eng['average_attendance'] < 75).astype(int)
df_eng.drop(columns=['average_attendance'], inplace=True)
```

[17]:
```
# <Student to fill this section>
feature_engineering_2_explanations = """
Provide some explanations on why you believe it is important to create this␣
 ↪feature and its impacts

The new feature 'attendance_flag' was created by converting the continuous␣
 ↪'average_attendance' variable into a binary indicator,
where 1 denotes students with attendance below 75% - a commonly used threshold␣
 ↪for concern.

This transformation simplifies the data and highlights students at risk due to␣
 ↪low attendance,
making it easier for the model to detect performance patterns related to␣
 ↪engagement without being influenced by minor attendance fluctuations.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_engineering_2_explanations',␣
 ↪value=feature_engineering_2_explanations)
```

```
<IPython.core.display.HTML object>
```

[18]:
```
print("Feature Engineering Explanations:", feature_engineering_2_explanations)
```

```
Feature Engineering Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The new feature 'attendance_flag' was created by converting the continuous
'average_attendance' variable into a binary indicator,
where 1 denotes students with attendance below 75% - a commonly used threshold
for concern.

This transformation simplifies the data and highlights students at risk due to
low attendance,
making it easier for the model to detect performance patterns related to
engagement without being influenced by minor attendance fluctuations.
```

1.7 F. Data Preparation

1.7.1 F.1 Data Transformation

[ ]:
```
# <Student to fill this section>
boolean_cols = ['scholarship', 'university_transport',␣
↪'has_consulted_teacher','co_curricular', 'health_issues', 'disabilities']

for col in boolean_cols:
    df_eng[col] = df_eng[col].map({'Yes': 1, 'No': 0})

df_eng["has_diploma"] = df_eng["has_diploma"].map({True: 1, False: 0})
```

[ ]:
```
target_mapping = {
    'Excellent': 3,
    'Good': 2,
    'Average': 1,
    'Poor': 0
}

df_eng["target"] = df_eng["target"].map(target_mapping)
```

[ ]:
```
eng_mapping = {
    'Advance': 2,
    'Intermediate': 1,
    'Basic': 0
}

df_eng['english_proficiency'] = df_eng['english_proficiency'].map(eng_mapping)
```

[19]:
```
# <Student to fill this section>
data_transformation_1_explanations = """
Provide some explanations on why you believe it is important to perform this␣
↪data transformation and its impacts

Binary and ordinal categorical features were mapped to numerical values to␣
↪ensure compatibility with machine learning models.
Boolean columns with 'Yes'/'No' responses were converted to 1/0, and the target␣
↪variable was mapped from performance labels to ordinal scores (0-3).
Similarly, 'english_proficiency' was mapped from textual levels to a numerical␣
↪scale.

These transformations simplify model interpretation, improve training␣
↪efficiency, and maintain the ordinal meaning of certain features,
while ensuring all input features are in numerical form.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_transformation_1_explanations',␣
↪value=data_transformation_1_explanations)
```

```
<IPython.core.display.HTML object>
```

[20]:
```
print("Data Transformation Explanations:", data_transformation_1_explanations)
```

```
Data Transformation Explanations:
Provide some explanations on why you believe it is important to perform this
data transformation and its impacts

Binary and ordinal categorical features were mapped to numerical values to
ensure compatibility with machine learning models.
Boolean columns with 'Yes'/'No' responses were converted to 1/0, and the target
variable was mapped from performance labels to ordinal scores (0-3).
Similarly, 'english_proficiency' was mapped from textual levels to a numerical
scale.

These transformations simplify model interpretation, improve training
efficiency, and maintain the ordinal meaning of certain features,
while ensuring all input features are in numerical form.
```

1.7.2 F.2 Data Transformation

[ ]:
```
# <Student to fill this section>

df_encoded = df_eng.copy()

string_cols = df_encoded.select_dtypes(include=['object', 'string']).columns.
↪tolist()

label_encoders = {}

for col in string_cols:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col].astype(str))
    label_encoders[col] = le
```

[21]:
```
# <Student to fill this section>
data_transformation_2_explanations = """
Provide some explanations on why you believe it is important to perform this␣
↪data transformation and its impacts
```

```
Label encoding was applied to all string-based categorical features to convert␣
  ↪them into numeric format.
This method is suitable because tree-based models do not assume any ordinal␣
  ↪relationship between the encoded values and they simply use them to split␣
  ↪data based on feature conditions.
Label encoding keeps the feature space compact, improves model training speed,␣
  ↪and avoids the complexity of one-hot encoding, while preserving category␣
  ↪information.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_transformation_2_explanations',␣
  ↪value=data_transformation_2_explanations)
```

```
<IPython.core.display.HTML object>
```

[22]:
```
print("Data Transformation Explanations:", data_transformation_2_explanations)
```

```
Data Transformation Explanations:
Provide some explanations on why you believe it is important to perform this
data transformation and its impacts

Label encoding was applied to all string-based categorical features to convert
them into numeric format.
This method is suitable because tree-based models do not assume any ordinal
relationship between the encoded values and they simply use them to split data
based on feature conditions.
Label encoding keeps the feature space compact, improves model training speed,
and avoids the complexity of one-hot encoding, while preserving category
information.
```

1.7.3 F.3 Data Transformation

[ ]:
```
df_sorted = df_encoded.sort_values(by='admission_year')

total_len = len(df_sorted)
train_end = int(0.7 * total_len)
val_end = int(0.9 * total_len)  # 70% + 20% = 90%

train_df = df_sorted.iloc[:train_end]
val_df = df_sorted.iloc[train_end:val_end]
test_df = df_sorted.iloc[val_end:]

print(f"Train: {len(train_df)} rows")
print(f"Validation: {len(val_df)} rows")
print(f"Test: {len(test_df)} rows")
```

```
Train: 706 rows
Validation: 202 rows
Test: 101 rows
```

[ ]:
```
X_train = train_df.drop(columns=['target'])
y_train = train_df['target']

X_val = val_df.drop(columns=['target'])
y_val = val_df['target']

X_test = test_df.drop(columns=['target'])
y_test = test_df['target']
```

[ ]:
```
# <Student to fill this section>
standard_scaler = StandardScaler()

column_names = X_train.columns.tolist()

X_train = pd.DataFrame(standard_scaler.fit_transform(X_train),␣
 ↪columns=column_names)
X_val = pd.DataFrame(standard_scaler.transform(X_val), columns=column_names)
X_test = pd.DataFrame(standard_scaler.transform(X_test), columns=column_names)
```

[23]:
```
# <Student to fill this section>
data_transformation_3_explanations = """
Provide some explanations on why you believe it is important to perform this␣
 ↪data transformation and its impacts

Standard scaling was applied to ensure all numerical features are on the same␣
 ↪scale, with a mean of 0 and a standard deviation of 1.
This transformation is important for models that are sensitive to feature␣
 ↪magnitude, as it improves training efficiency and model performance.

The scaler was fitted on the training data and then applied to the validation␣
 ↪and test sets to avoid data leakage.
This ensures consistent scaling across all datasets while preserving the␣
 ↪integrity of unseen data during evaluation.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_transformation_3_explanations',␣
 ↪value=data_transformation_3_explanations)
```

```
<IPython.core.display.HTML object>
```

[24]:
```
print("Data Transformation Explanations:", data_transformation_3_explanations)
```

```
Data Transformation Explanations:
Provide some explanations on why you believe it is important to perform this
data transformation and its impacts

Standard scaling was applied to ensure all numerical features are on the same
scale, with a mean of 0 and a standard deviation of 1.
This transformation is important for models that are sensitive to feature
magnitude, as it improves training efficiency and model performance.

The scaler was fitted on the training data and then applied to the validation
and test sets to avoid data leakage.
This ensures consistent scaling across all datasets while preserving the
integrity of unseen data during evaluation.
```

1.7.4 F.4 Data Transformation

[ ]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

[ ]:
```
X_train_resampled.shape
```

[ ]:
```
(1408, 32)
```

[25]:
```
# <Student to fill this section>
data_transformation_4_explanations = """
Provide some explanations on why you believe it is important to perform this␣
  ↪data transformation and its impacts

SMOTE was applied to the training set to address class imbalance across all␣
  ↪classes, including multiple minority classes.
By generating synthetic examples for each underrepresented class, it helps the␣
  ↪model learn more equally from all categories,
improving recall and reducing bias. This enhances the model's ability to␣
  ↪identify students at different risk levels.
After resampling, the training set now contains 1,408 samples.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_transformation_4_explanations',␣
  ↪value=data_transformation_4_explanations)
```

```
<IPython.core.display.HTML object>
```

```
[26]: print("Data Transformation Explanations:", data_transformation_4_explanations)
```

```
Data Transformation Explanations:
Provide some explanations on why you believe it is important to perform this
data transformation and its impacts

SMOTE was applied to the training set to address class imbalance across all
classes, including multiple minority classes.
By generating synthetic examples for each underrepresented class, it helps the
model learn more equally from all categories,
improving recall and reducing bias. This enhances the model's ability to
identify students at different risk levels.
After resampling, the training set now contains 1,408 samples.
```

1.8 G. Train Machine Learning Model

1.8.1 G.1 Import Algorithm

```
[ ]: # <Student to fill this section>
from sklearn.ensemble import ExtraTreesClassifier
```

```
[27]: # <Student to fill this section>
algorithm_selection_explanations = """
Provide some explanations on why you believe this algorithm is a good fit

Extra Trees is a good fit for this task because it combines the strength of␣
  ↪multiple randomized decision trees to improve accuracy and reduce␣
  ↪overfitting.
It performs well with high-dimensional data, handles non-linear relationships,␣
  ↪and is robust to noise.
Its ensemble nature also provides better generalization compared to a single␣
  ↪decision tree, making it suitable for predicting student performance across␣
  ↪varied feature sets.
"""
```

```
[ ]: # Do not modify this code
print_tile(size="h3", key='algorithm_selection_explanations',␣
  ↪value=algorithm_selection_explanations)
```

```
<IPython.core.display.HTML object>
```

```
[28]: print("Algorithm Selection Explanations:", algorithm_selection_explanations)
```

```
Algorithm Selection Explanations:
Provide some explanations on why you believe this algorithm is a good fit
```

```
Extra Trees is a good fit for this task because it combines the strength of
multiple randomized decision trees to improve accuracy and reduce overfitting.
It performs well with high-dimensional data, handles non-linear relationships,
and is robust to noise.
Its ensemble nature also provides better generalization compared to a single
decision tree, making it suitable for predicting student performance across
varied feature sets.
```

1.8.2 G.2 Set Hyperparameters

[ ]:
```
# <Student to fill this section>
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [5, 10, 20],
    'min_samples_leaf': [2, 5, 7],
    'criterion': ['gini', 'entropy'],
    'class_weight': ['balanced']
}
```

[29]:
```
# <Student to fill this section>
hyperparameters_selection_explanations = """
Explain why you are tuning these hyperparameters

These hyperparameters are tuned to improve model performance and control␣
↪overfitting.
- 'n_estimators' determines the number of trees in the ensemble, which can␣
↪affect stability and accuracy.
- 'max_depth' controls the depth of each tree, helping balance bias and␣
↪variance.
- 'min_samples_leaf' prevents trees from learning overly specific patterns,␣
↪improving generalization.
- 'criterion' changes the splitting metric, allowing evaluation of both Gini␣
↪and entropy for better splits.
- 'class_weight' is set to 'balanced' to handle class imbalance introduced by␣
↪the original dataset distribution.
Tuning these helps the model generalize better and perform well across all␣
↪student categories.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='hyperparameters_selection_explanations',␣
↪value=hyperparameters_selection_explanations)
```

```
<IPython.core.display.HTML object>
```

[30]:
```
print("Hyperparameters Selection Explanations:",␣
↪hyperparameters_selection_explanations)
```

```
Hyperparameters Selection Explanations:
Explain why you are tuning these hyperparameters

These hyperparameters are tuned to improve model performance and control
overfitting.
- 'n_estimators' determines the number of trees in the ensemble, which can
affect stability and accuracy.
- 'max_depth' controls the depth of each tree, helping balance bias and
variance.
- 'min_samples_leaf' prevents trees from learning overly specific patterns,
improving generalization.
- 'criterion' changes the splitting metric, allowing evaluation of both Gini and
entropy for better splits.
- 'class_weight' is set to 'balanced' to handle class imbalance introduced by
the original dataset distribution.
Tuning these helps the model generalize better and perform well across all
student categories.
```

1.8.3 G.3 Fit Model

[ ]:
```
# <Student to fill this section>
et_model = ExtraTreesClassifier(random_state=42)

grid_search_et = GridSearchCV(
    estimator=et_model,
    param_grid=param_grid,
    cv=5,
    scoring='f1_weighted',
)

grid_search_et.fit(X_train_resampled, y_train_resampled)

print("Best Extra Trees parameters:", grid_search_et.best_params_)
print("Best Weighted F1 Score:", round(grid_search_et.best_score_, 4))

results_df = pd.DataFrame(grid_search_et.cv_results_)
results_df = results_df[[
    'mean_test_score', 'std_test_score', 'params'
]]
results_df = results_df.sort_values(by='mean_test_score', ascending=False).
↪reset_index(drop=True)
# Display full params in the results DataFrame
pd.set_option('display.max_colwidth', None)
```

```
results_df
```

```
Best Extra Trees parameters: {'class_weight': 'balanced', 'criterion':
'entropy', 'max_depth': 20, 'min_samples_leaf': 2, 'n_estimators': 200}
Best Weighted F1 Score: 0.9396
```

[ ]:
```
    mean_test_score  std_test_score  \
0          0.939553        0.018822
1          0.937455        0.016301
2          0.936600        0.019466
3          0.932398        0.019456
4          0.919060        0.019992
5          0.917413        0.022251
6          0.913928        0.016044
7          0.913367        0.022311
8          0.910964        0.021170
9          0.910280        0.016695
10         0.909920        0.014588
11         0.907709        0.019981
12         0.901111        0.015409
13         0.899583        0.014347
14         0.898476        0.020005
15         0.897983        0.021543
16         0.897111        0.012756
17         0.896936        0.019860
18         0.896129        0.021871
19         0.895728        0.023659
20         0.889827        0.017588
21         0.889721        0.018636
22         0.889552        0.017136
23         0.888117        0.016381
24         0.856420        0.010650
25         0.854166        0.024449
26         0.853649        0.022626
27         0.852122        0.025730
28         0.851267        0.019182
29         0.851090        0.021493
30         0.850421        0.017628
31         0.849633        0.015939
32         0.848615        0.023963
33         0.848517        0.020972
34         0.848512        0.014377
35         0.847120        0.021053

                                               params
0   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 2, 'n_estimators': 200}
1   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 2, 'n_estimators': 100}
2      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 2, 'n_estimators': 200}
3      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 2, 'n_estimators': 100}
4   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 2, 'n_estimators': 200}
5      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 2, 'n_estimators': 100}
6      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 2, 'n_estimators': 200}
7   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 2, 'n_estimators': 100}
8      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 5, 'n_estimators': 100}
9   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 5, 'n_estimators': 100}
10  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 5, 'n_estimators': 200}
11     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 5, 'n_estimators': 200}
12  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 7, 'n_estimators': 200}
13  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 5, 'n_estimators': 200}
14     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 5, 'n_estimators': 200}
15     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 7, 'n_estimators': 100}
16  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 20,
'min_samples_leaf': 7, 'n_estimators': 100}
17     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 7, 'n_estimators': 200}
18     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 5, 'n_estimators': 100}
19  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 5, 'n_estimators': 100}
20  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 7, 'n_estimators': 200}
21     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 7, 'n_estimators': 200}
22  {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 7, 'n_estimators': 100}
23     {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 10,
'min_samples_leaf': 7, 'n_estimators': 100}
```

```
24   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 5, 'n_estimators': 100}
25   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 2, 'n_estimators': 100}
26      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 2, 'n_estimators': 200}
27   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 2, 'n_estimators': 200}
28   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 7, 'n_estimators': 100}
29      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 7, 'n_estimators': 200}
30      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 7, 'n_estimators': 100}
31   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 5, 'n_estimators': 200}
32      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 5, 'n_estimators': 200}
33   {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 5,
'min_samples_leaf': 7, 'n_estimators': 200}
34      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 5, 'n_estimators': 100}
35      {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 5,
'min_samples_leaf': 2, 'n_estimators': 100}
```

[ ]:
```
et_model = ExtraTreesClassifier(class_weight= 'balanced', criterion= 'entropy',␣
 ↪max_depth= 20, min_samples_leaf= 2, n_estimators= 200, random_state=42)
et_model.fit(X_train_resampled, y_train_resampled)

cv_scores = cross_val_score(et_model, X_train_resampled, y_train_resampled,␣
 ↪cv=5, scoring='f1_weighted')
print("Cross-Validated Weighted F1 Scores (Train Set):", cv_scores)
print("Mean CV Weighted F1 Score:", cv_scores.mean())
```

```
Cross-Validated Weighted F1 Scores (Train Set): [0.91885794 0.93627
0.93183163 0.9357769  0.97502653]
Mean CV Weighted F1 Score: 0.9395526001015376
```

1.8.4 G.4 Model Technical Performance

[ ]:
```
# <Student to fill this section>

y_preds = et_model.predict(X_val)

print("ExtraTrees Evaluation:\n")
print("\nRecall_score:", recall_score(y_val, y_preds, average='weighted'))
print("\nf1_score:", f1_score(y_val, y_preds, average='weighted'))
```

```python
print("\nClassification Report:\n", classification_report(y_val, y_preds))
print("\nConfusion Matrix:\n", confusion_matrix(y_val, y_preds))
```

```
ExtraTrees Evaluation:


Recall_score: 0.806930693069307

f1_score: 0.8094187296464495

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.93      0.94       103
           1       0.81      0.67      0.73        51
           2       0.56      0.77      0.65        35
           3       0.55      0.46      0.50        13

    accuracy                           0.81       202
   macro avg       0.72      0.71      0.71       202
weighted avg       0.82      0.81      0.81       202


Confusion Matrix:
 [[96  4  3  0]
 [ 5 34 11  1]
 [ 0  4 27  4]
 [ 0  0  7  6]]
```

[31]:
```python
# <Student to fill this section>
model_performance_explanations = """
Provide some explanations on model performance

The Extra Trees model achieved strong overall performance, with a weighted F1␣
  ↪score of 0.9396 in cross-validation and 0.81 on the validation set,
demonstrating an improvement over the previous experiment.
The recall and F1 scores on validation data indicate a good balance between␣
  ↪precision and recall, making the model reliable for practical use.

The best hyperparameters were selected using GridSearchCV, as manually testing␣
  ↪all combinations would be time-consuming,
and a systematic exploration of all possible options was preferred.

The model performed exceptionally well for the 'Poor' class (label 0), with a␣
  ↪recall of 0.93 and an F1-score of 0.94, correctly identifying 96 out of 103␣
  ↪students.
```

```
This high recall is crucial for early intervention, providing appropriate␣
  ↪support and aligns well with the business objective of identifying at-risk␣
  ↪students.

Performance for the 'Average' class (label 1) also improved, with a recall of 0.
  ↪67 and an F1-score of 0.73.
This shows the model's growing ability to distinguish borderline cases and␣
  ↪reduce misclassification into more extreme categories.

While performance for 'Good' and 'Excellent' classes remains moderate, the␣
  ↪improvements in 'Poor' and 'Average' are highly valuable from a business␣
  ↪perspective,
ensuring that the most critical groups are accurately identified.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='model_performance_explanations',␣
  ↪value=model_performance_explanations)
```

```
<IPython.core.display.HTML object>
```

[32]:
```
print("Model Performance Explanations:", model_performance_explanations)
```

```
Model Performance Explanations:
Provide some explanations on model performance

The Extra Trees model achieved strong overall performance, with a weighted F1
score of 0.9396 in cross-validation and 0.81 on the validation set,
demonstrating an improvement over the previous experiment.
The recall and F1 scores on validation data indicate a good balance between
precision and recall, making the model reliable for practical use.

The best hyperparameters were selected using GridSearchCV, as manually testing
all combinations would be time-consuming,
and a systematic exploration of all possible options was preferred.

The model performed exceptionally well for the 'Poor' class (label 0), with a
recall of 0.93 and an F1-score of 0.94, correctly identifying 96 out of 103
students.
This high recall is crucial for early intervention, providing appropriate
support and aligns well with the business objective of identifying at-risk
students.

Performance for the 'Average' class (label 1) also improved, with a recall of
0.67 and an F1-score of 0.73.
This shows the model's growing ability to distinguish borderline cases and
reduce misclassification into more extreme categories.
```

```
While performance for 'Good' and 'Excellent' classes remains moderate, the
improvements in 'Poor' and 'Average' are highly valuable from a business
perspective,
ensuring that the most critical groups are accurately identified.
```

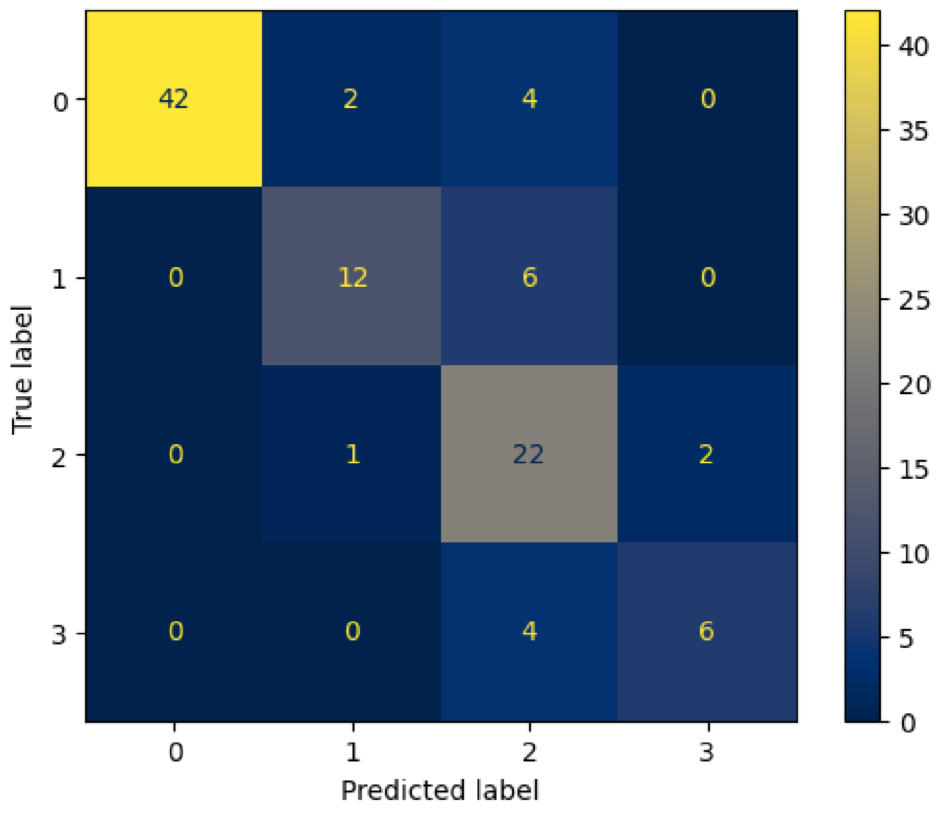
1.8.5 G.5 Business Impact from Current Model Performance

```
[ ]: # <Student to fill this section>

     y_pred_final = et_model.predict(X_test)

     print("ExtraTrees Evaluation:\n")
     print("\nRecall_score:", recall_score(y_test, y_pred_final, average='weighted'))
     print("\nf1_score:", f1_score(y_test, y_pred_final, average='weighted'))
     print("\nClassification Report:\n", classification_report(y_test, y_pred_final))
```

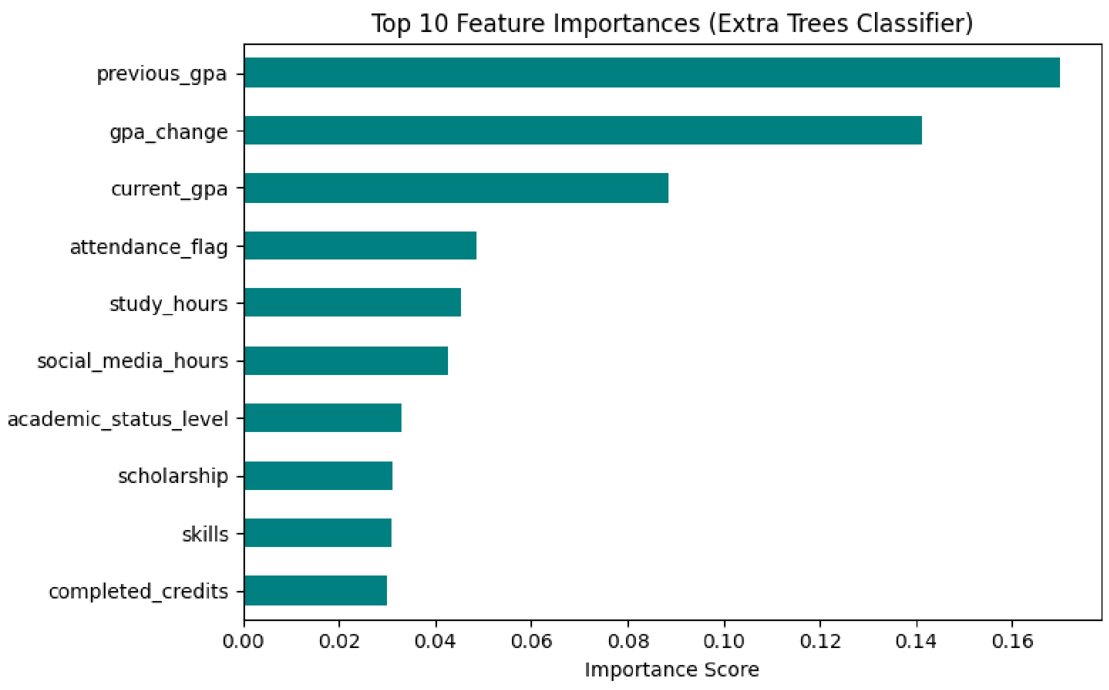
```
ExtraTrees Evaluation:


Recall_score: 0.8118811881188119

f1_score: 0.8177263628002145

Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.88      0.93        48
           1       0.80      0.67      0.73        18
           2       0.61      0.88      0.72        25
           3       0.75      0.60      0.67        10

    accuracy                           0.81       101
   macro avg       0.79      0.76      0.76       101
weighted avg       0.84      0.81      0.82       101
```

```
[ ]: ConfusionMatrixDisplay.from_estimator(et_model, X_test, y_test, cmap='cividis')
```

```
[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
     0x787d2819fc90>
```



```
[ ]: # Get feature importances
importances = et_model.feature_importances_

feature_names = X_train_resampled.columns
feat_importances = pd.Series(importances, index=feature_names)

feat_importances.nlargest(10).plot(kind='barh', figsize=(8, 5), color='teal')
plt.xlabel('Importance Score')
plt.title('Top 10 Feature Importances (Extra Trees Classifier)')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



[33]:
```
# <Student to fill this section>
business_impacts_explanations = """
Interpret the results of the experiments related to the business objective set␣
  ↪earlier. Estimate the impacts of the incorrect results for the business␣
  ↪(some results may have more impact compared to others)

The model performed well on unseen test data, achieving an F1 score of 0.82 and␣
  ↪a recall of 0.81,
which may support its reliability for real-world application. These results␣
  ↪align closely with the business objective of identifying at-risk students␣
  ↪for early intervention.

The model demonstrated excellent performance in predicting the 'Poor' class␣
  ↪(label 0) and the 'Average" class (label 1), with a recall of 0.88 and 0.67␣
  ↪respectively.
This shows higher recall compared to the previous experiment, meaning the model␣
  ↪is better at identifying most of the students who actually need support.
This supports targeted academic support and resource allocation and helps␣
  ↪reduce the risk of misclassifying borderline students.
Similarly, the 'Good' (label 2) and 'Excellent' (label 3) classes also saw␣
  ↪decent improvements, which supports better student segmentation.

Misclassifications are mostly within adjacent categories (e.g., 'Good' vs.␣
  ↪'Excellent'),
which have lower business impact compared to errors in identifying 'Poor'␣
  ↪students.
However, predicting a high-performing student as 'Poor' could waste resources␣
  ↪and reduce trust in the system.
As the highest priority group is correctly captured with high confidence, the␣
  ↪model successfully supports the business goal of timely outreach and␣
  ↪efficient allocation of support services.

The plot also shows that new features such as gpa_change and attendance_flag␣
  ↪have relatively high importance in predicting student performance,
along with study_hours and social_media_hours.
Scholarship and skills also appear among the top 10 most important features.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='business_impacts_explanations',␣
  ↪value=business_impacts_explanations)
```

```
<IPython.core.display.HTML object>
```

[34]:
```
print("Business Impact Explanations:", business_impacts_explanations)
```

```
Business Impact Explanations:
Interpret the results of the experiments related to the business objective set
earlier. Estimate the impacts of the incorrect results for the business (some
results may have more impact compared to others)

The model performed well on unseen test data, achieving an F1 score of 0.82 and
a recall of 0.81,
which may support its reliability for real-world application. These results
align closely with the business objective of identifying at-risk students for
early intervention.

The model demonstrated excellent performance in predicting the 'Poor' class
(label 0) and the 'Average" class (label 1), with a recall of 0.88 and 0.67
respectively.
This shows higher recall compared to the previous experiment, meaning the model
is better at identifying most of the students who actually need support.
This supports targeted academic support and resource allocation and helps reduce
the risk of misclassifying borderline students.
Similarly, the 'Good' (label 2) and 'Excellent' (label 3) classes also saw
decent improvements, which supports better student segmentation.

Misclassifications are mostly within adjacent categories (e.g., 'Good' vs.
'Excellent'),
which have lower business impact compared to errors in identifying 'Poor'
students.
```

```
However, predicting a high-performing student as 'Poor' could waste resources
and reduce trust in the system.
As the highest priority group is correctly captured with high confidence, the
model successfully supports the business goal of timely outreach and efficient
allocation of support services.

The plot also shows that new features such as gpa_change and attendance_flag
have relatively high importance in predicting student performance,
along with study_hours and social_media_hours.
Scholarship and skills also appear among the top 10 most important features.
```

1.9 H. Experiment Outcomes

[35]:
```
# <Student to fill this section>
experiment_outcome = "Hypothesis Confirmed" # Either 'Hypothesis Confirmed',␣
↪'Hypothesis Partially Confirmed' or 'Hypothesis Rejected'
```

[ ]:
```
# Do not modify this code
print_tile(size="h2", key='experiment_outcomes_explanations',␣
↪value=experiment_outcome)
```

```
<IPython.core.display.HTML object>
```

[36]:
```
print("Experiment Outcome:", experiment_outcome)
```

```
Experiment Outcome: Hypothesis Confirmed
```

[38]:
```
# <Student to fill this section>
experiment_results_explanations = """
Reflect on the outcome of the experiment and list the new insights you gained␣
↪from it. Provide rationale for pursuing more experimentation with the␣
↪current approach or call out if you think it is a dead end.
Given the results achieved and the overall objective of the project, list the␣
↪potential next steps and experiments. For each of them assess the expected␣
↪uplift or gains and rank them accordingly. If the experiment achieved the␣
↪required outcome for the business, recommend the steps to deploy this␣
↪solution into production.

The best combination of hyperparameters was: class_weight='balanced',␣
↪criterion='entropy', max_depth=20, min_samples_leaf=2, and n_estimators=200.
The model achieved a cross-validated weighted F1 score of 0.9396 and a␣
↪validation F1 score of 0.81, both higher than in the previous experiment.
Performance improved especially for the 'Poor' and 'Average' classes,␣
↪confirming that adding features (academic_status_level and attendance_flag)␣
↪and applying label encoding helped the model generalize more effectively.
```

```
Interestingly, the test set showed better F1 and recall scores than the␣
 ↪validation set, suggesting strong generalization and no signs of overfitting.
This highlights the reliability of the current feature set and preprocessing␣
 ↪pipeline when applied to unseen data.

New insights gained include:
- Simple label encoding is effective with Extra Trees and helps avoid␣
 ↪unnecessary dimensionality.
- Carefully engineered features can significantly boost class-wise recall,␣
 ↪especially in critical categories.
- GPA and study hours are strong predictors of student performance,
highlighting the importance of academic history and study habits in identifying␣
 ↪at-risk students.

Next steps include:

1. **Test performance on new, external datasets (e.g., from another semester)**
   - Expected gain: High (validates robustness and readiness for deployment)
   - Priority: High

While the model now meets both technical and business objectives, especially in␣
 ↪reliably identifying at-risk students,
it is not yet ready for deployment. Although this is a good result, it may be␣
 ↪due to the test set not fully representing real-world unseen data,
and the model's performance may not generalize in practice.
The current training set includes a relatively small number of samples (1,408),␣
 ↪with some synthetic data generated through SMOTE,
and may not reflect up-to-date student behaviors or academic trends.
Further testing on a larger and more current dataset is strongly recommended
to confirm the model's reliability and ensure it remains effective in␣
 ↪real-world use.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h2", key='experiment_results_explanations',␣
 ↪value=experiment_results_explanations)
```

```
<IPython.core.display.HTML object>
```

[39]:
```
print("Experiment Results Explanations:", experiment_results_explanations)
```

```
Experiment Results Explanations:
Reflect on the outcome of the experiment and list the new insights you gained
from it. Provide rationale for pursuing more experimentation with the current
approach or call out if you think it is a dead end.
Given the results achieved and the overall objective of the project, list the
potential next steps and experiments. For each of them assess the expected
```

```
uplift or gains and rank them accordingly. If the experiment achieved the
required outcome for the business, recommend the steps to deploy this solution
into production.

The best combination of hyperparameters was: class_weight='balanced',
criterion='entropy', max_depth=20, min_samples_leaf=2, and n_estimators=200.
The model achieved a cross-validated weighted F1 score of 0.9396 and a
validation F1 score of 0.81, both higher than in the previous experiment.
Performance improved especially for the 'Poor' and 'Average' classes, confirming
that adding features (academic_status_level and attendance_flag) and applying
label encoding helped the model generalize more effectively.

Interestingly, the test set showed better F1 and recall scores than the
validation set, suggesting strong generalization and no signs of overfitting.
This highlights the reliability of the current feature set and preprocessing
pipeline when applied to unseen data.

New insights gained include:
- Simple label encoding is effective with Extra Trees and helps avoid
unnecessary dimensionality.
- Carefully engineered features can significantly boost class-wise recall,
especially in critical categories.
- GPA and study hours are strong predictors of student performance,
highlighting the importance of academic history and study habits in identifying
at-risk students.

Next steps include:

1. **Test performance on new, external datasets (e.g., from another semester)**
   - Expected gain: High (validates robustness and readiness for deployment)
   - Priority: High

While the model now meets both technical and business objectives, especially in
reliably identifying at-risk students,
it is not yet ready for deployment. Although this is a good result, it may be
due to the test set not fully representing real-world unseen data,
and the model's performance may not generalize in practice.
The current training set includes a relatively small number of samples (1,408),
with some synthetic data generated through SMOTE,
and may not reflect up-to-date student behaviors or academic trends.
Further testing on a larger and more current dataset is strongly recommended
to confirm the model’s reliability and ensure it remains effective in real-world
use.
```