36106-25AU-AT2-25589351-experiment-0

April 25, 2025

1 Experiment Notebook

1.1 0. Setup Environment

1.1.1 0.a Install Environment and Mandatory Packages

[6]:
```
# 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 -:--:--
                       0.2/1.6
MB 5.0 MB/s eta 0:00:01
                       1.6/1.6 MB
22.6 MB/s eta 0:00:01
                       1.6/1.6 MB 17.5
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

[10]:
```
# Do not modify this code
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

1.1.3 0.c Install Additional Packages

1.1.4 0.d Import Packages

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

[7]:
```
# <Student to fill this section>
!apt-get update > /dev/null 2>&1
!apt-get install -y texlive texlive-xetex texlive-latex-extra pandoc > /dev/
↪null 2>&1
```

[11]:
```
# <Student to fill this section>
import pandas as pd
import altair as alt
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import recall_score, f1_score, classification_report,␣
↪confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
```

1.2 A. Project Description

[13]:
```
# <Student to fill this section>
student_name = "Fatemeh Elyasifar"
student_id = "25589351"
```

[20]:
```
# Do not modify this code
print_tile(size="h1", key='student_name', value=student_name)
```

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

[21]:
```
# Do not modify this code
print_tile(size="h1", key='student_id', value=student_id)
```

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

[14]:
```python
print("Student Name:", student_name)
print("Student ID:", student_id)
```

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

[16]:
```python
# <Student to fill this section>
business_objective = """
Explain clearly what is the goal of this project for the business. How will the␣
↪results be used? What will be the impact of accurate or incorrect results?

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.
"""
```

[22]:
```python
# Do not modify this code
print_tile(size="h3", key='business_objective', value=business_objective)
```

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

[17]:
```python
print("Business Objective:", business_objective)
```

```
Business Objective:
Explain clearly what is the goal of this project for the business. How will the
results be used? What will be the impact of accurate or incorrect results?

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

[23]:
```python
# Do not modify this code
experiment_id = "0"
print_tile(size="h1", key='experiment_id', value=experiment_id)
```

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

[24]:
```python
print("Experiment ID:", experiment_id)
```

```
Experiment ID: 0
```

[19]:
```python
# <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

The hypothesis is that students' academic performance can be accurately predicted using a combination of study habits, attendance, GPA, and behavioral indicators.
Features such as previous GPA and study hours are expected to have the strongest influence on performance.
Investigating the importance of these features is valuable, as it can guide support staff in focusing on the most impactful factors when designing intervention strategies,
ultimately improving student outcomes.
```

```
"""
```

[25]:
```
# Do not modify this code
print_tile(size="h3", key='experiment_hypothesis', value=experiment_hypothesis)
```

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

[26]:
```
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

The hypothesis is that students' academic performance can be accurately
predicted using a combination of study habits, attendance, GPA, and behavioral
indicators.
Features such as previous GPA and study hours are expected to have the strongest
influence on performance.
Investigating the importance of these features is valuable, as it can guide
support staff in focusing on the most impactful factors when designing
intervention strategies,
ultimately improving student outcomes.
```

[28]:
```
# <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.

Data errors and missing values will be identified, and necessary data␣
↪transformations and feature engineering will be performed to prepare the␣
↪dataset for model training.
The data will also be split into three parts: training, validation, and test␣
↪sets.
A baseline model will be used to set initial F1 and recall scores of␣
↪approximately 50, providing a benchmark for assessing the performance of␣
↪future models.

Possible scenarios include:
- High F1 and recall, supporting effective early intervention.
- Moderate performance, indicating a need for better feature selection or model␣
↪tuning.
- Low F1 and recall scores from the baseline model, highlighting the need for␣
↪more advanced modeling.
"""
```

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

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

[29]:
```python
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.

Data errors and missing values will be identified, and necessary data
transformations and feature engineering will be performed to prepare the dataset
for model training.
The data will also be split into three parts: training, validation, and test
sets.
A baseline model will be used to set initial F1 and recall scores of
approximately 50, providing a benchmark for assessing the performance of future
models.

Possible scenarios include:
- High F1 and recall, supporting effective early intervention.
- Moderate performance, indicating a need for better feature selection or model
tuning.
- Low F1 and recall scores from the baseline model, highlighting the need for
more advanced modeling.
```

1.4 C. Data Understanding

[31]:
```python
# Do not modify this code
try:
  df = pd.read_csv(at.folder_path / "students_performance.csv")
except Exception as e:
  print(e)
```

1.4.1 C.1 Explore Dataset

[ ]:
```python
# <Student to fill this section>
df.head()
```

[ ]:
```
   student_id          full_name   age                      email  \
0           7        Lauren Moon  22.0  kimberlypark@example.org
```

```
1          11        Larry Green  22.0      smithamy@example.net
2          15    Alexander Scott  20.0         qlee@example.org
3          18  Jonathan Thornton  21.0       cmorgan@example.com
4          20        Susan Smith  21.0         xtodd@example.com

   phone_number  gender birth_country secondary_address  building_number  \
0  (08)35431944  Female            AU           Unit 93                0
1  (07)35774291    Male            AU           Level 2               43
2      20356212    Male            NZ              937/               96
3     0627-6253    Male            AU           Level 3                5
4     9957-3583  Female            AU               00/                5

       street_name  … social_media_hours average_attendance  \
0      April Amble  …                2.0              100.0
1      Davis Crest  …                2.0               90.0
2     Emily Little  …                1.0               95.0
3  William Pathway  …                3.0               95.0
4     Martin Close  …                2.0               96.0

                         skills skills_development_hours  \
0  Web development skill(Frontend)                     1.0
1                    Programming                     1.0
2                    Programming                     3.0
3                    Programming                     1.0
4  Web development skill(Frontend)                     1.0

                 area_of_interest  previous_gpa current_gpa  \
0                      Networking          3.80        3.64
1                    Data Science          3.40        3.53
2  Machine Learning / Deep Learning          3.93        3.89
3         Artificial Intelligence          3.10        3.50
4                 Web Development          3.81        3.65

  completed_credits has_diploma house_income
0              35.0       False      32500.0
1              35.0       False      20000.0
2              35.0       False      30000.0
3              35.0       False      25000.0
4              34.0       False      30000.0

[5 rows x 45 columns]
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 45 columns):
```

```
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   student_id                1009 non-null   int64  
 1   full_name                 1009 non-null   object 
 2   age                       1009 non-null   float64
 3   email                     1009 non-null   object 
 4   phone_number              1009 non-null   object 
 5   gender                    1009 non-null   object 
 6   birth_country             1009 non-null   object 
 7   secondary_address         1009 non-null   object 
 8   building_number           1009 non-null   int64  
 9   street_name               1009 non-null   object 
 10  street_suffix             1009 non-null   object 
 11  city                      1009 non-null   object 
 12  postcode                  1009 non-null   int64  
 13  state_abbr                1009 non-null   object 
 14  admission_year            1009 non-null   float64
 15  hsc_year                  1009 non-null   float64
 16  program                   1009 non-null   object 
 17  scholarship               1009 non-null   object 
 18  university_transport      1009 non-null   object 
 19  learning_mode             1009 non-null   object 
 20  has_phone                 1009 non-null   object 
 21  has_laptop                1009 non-null   object 
 22  english_proficiency       1009 non-null   object 
 23  on_probation              1009 non-null   object 
 24  is_suspended              1009 non-null   object 
 25  has_consulted_teacher     1009 non-null   object 
 26  relationship              1009 non-null   object 
 27  co_curricular             1009 non-null   object 
 28  living_arrangement        1009 non-null   object 
 29  health_issues             1009 non-null   object 
 30  disabilities              1009 non-null   object 
 31  target                    1009 non-null   object 
 32  current _semester         1009 non-null   float64
 33  study_hours               1009 non-null   float64
 34  study_sessions            1009 non-null   float64
 35  social_media_hours        1009 non-null   float64
 36  average_attendance        1009 non-null   float64
 37  skills                    1008 non-null   object 
 38  skills_development_hours  1009 non-null   float64
 39  area_of_interest          1002 non-null   object 
 40  previous_gpa              1009 non-null   float64
 41  current_gpa               1009 non-null   float64
 42  completed_credits         1009 non-null   float64
 43  has_diploma               1009 non-null   bool   
 44  house_income              1009 non-null   float64
dtypes: bool(1), float64(13), int64(3), object(28)
```

```
memory usage: 348.0+ KB
```

[ ]:
```
df.describe()
```

[ ]:
```
        student_id          age  building_number     postcode  admission_year  \
count  1009.000000  1009.000000      1009.000000  1009.000000     1009.000000
mean    673.108028    21.368285       180.305253  3153.528246     2040.321110
std     311.377223     1.614943       273.531962  1768.243964      629.677177
min       7.000000    18.000000         0.000000   202.000000     2013.000000
25%     410.000000    20.000000         6.000000  2602.000000     2020.000000
50%     685.000000    21.000000        46.000000  2691.000000     2021.000000
75%     941.000000    22.000000       237.000000  2960.000000     2022.000000
max    1193.000000    26.000000       998.000000  9941.000000    22022.000000

          hsc_year  current _semester  study_hours  study_sessions  \
count  1009.000000        1009.000000  1009.000000     1009.000000
mean   2019.251734          43.000991     3.334616        2.066898
std       1.346681         266.874155     2.096762        1.034492
min    2012.000000           1.000000     0.000000        0.000000
25%    2019.000000           3.000000     2.000000        1.000000
50%    2020.000000           8.000000     3.000000        2.000000
75%    2020.000000          10.000000     4.000000        2.000000
max    2028.000000        2022.000000    30.000000       10.000000

       social_media_hours  average_attendance  skills_development_hours  \
count         1009.000000         1009.000000               1009.000000
mean             3.439296           88.111001                  2.224975
std              2.439363           16.079094                  1.473957
min              0.000000            0.000000                  0.000000
25%              2.000000           80.000000                  1.000000
50%              3.000000           95.000000                  2.000000
75%              4.000000          100.000000                  3.000000
max             20.000000          100.000000                 20.000000

       previous_gpa  current_gpa  completed_credits  house_income
count   1009.000000  1009.000000        1009.000000  1.009000e+03
mean       2.756482     3.211343          76.936571  6.349576e+04
std        0.858012     0.731698          47.733885  7.927658e+04
min        0.000000     0.000000           0.000000  2.530000e+03
25%        2.110000     2.880000          24.000000  3.000000e+04
50%        2.770000     3.390000          85.000000  5.000000e+04
75%        3.480000     3.710000         122.000000  7.700000e+04
max        5.000000     4.670000         147.000000  2.000000e+06
```

[ ]:
```
missing = df.isnull().sum().sort_values(ascending=False)
print("Missing values:\n", missing[missing > 0])
```

```
Missing values:
```

```
 area_of_interest    7
skills              1
dtype: int64
```

[ ]:
```
duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")
```

```
Number of duplicate rows: 0
```

[ ]:
```
df['admission_year'].unique()
```

[ ]:
```
array([ 2021.,  2022.,  2020.,  2018.,  2019.,  2017.,  2014.,  2016.,
        2015.,  2013., 22022.,  2023.])
```

[ ]:
```
df['current _semester'].unique()
```

[ ]:
```
array([4.000e+00, 2.000e+00, 8.000e+00, 1.200e+01, 1.100e+01, 9.000e+00,
       1.300e+01, 5.000e+00, 1.400e+01, 1.000e+01, 7.000e+00, 2.200e+01,
       1.500e+01, 1.800e+01, 2.022e+03, 1.000e+00, 1.700e+01, 3.000e+00,
       6.000e+00, 1.900e+01, 2.400e+01])
```

[33]:
```
# <Student to fill this section>
dataset_insights = """
provide a detailed analysis on the dataset, its dimensions, information,␣
 ↪issues, ...

The dataset contains information on 1,009 students and consists of 45 columns,␣
 ↪including demographic details, academic history, behavioural metrics,
and performance indicators. It offers a comprehensive view of factors␣
 ↪potentially influencing student outcomes.

Only 'skills' (1 missing value) and 'area_of_interest' (7 missing values) have␣
 ↪missing data,
and no duplicate rows were found, indicating strong data integrity.

The dataset includes a mix of data types:
- Numerical (e.g., prevoius_gpa, average_attendance, study_hours,␣
 ↪social_media_hours)
- Categorical (e.g., gender, program, skills)
- Boolean (e.g., has_diploma)

Some variables, like 'current _semester', contain inconsistent naming (extra␣
 ↪space), which may require renaming for compatibility.
Additionally, columns such as 'admission_year' and 'hsc_year' include data␣
 ↪entry errors (e.g., admission_year max = 22022) that need correction.
```

```
Socioeconomic indicators like 'house_income' show a large range (min = 2,530 to␣
↪max = 2,000,000), suggesting possible data skew or outliers.

Overall, the dataset is suitable for predictive modeling but requires␣
↪preprocessing,
including error correction, missing value handling, and minor cleaning of␣
↪column names.
"""
```

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

<IPython.core.display.HTML object>

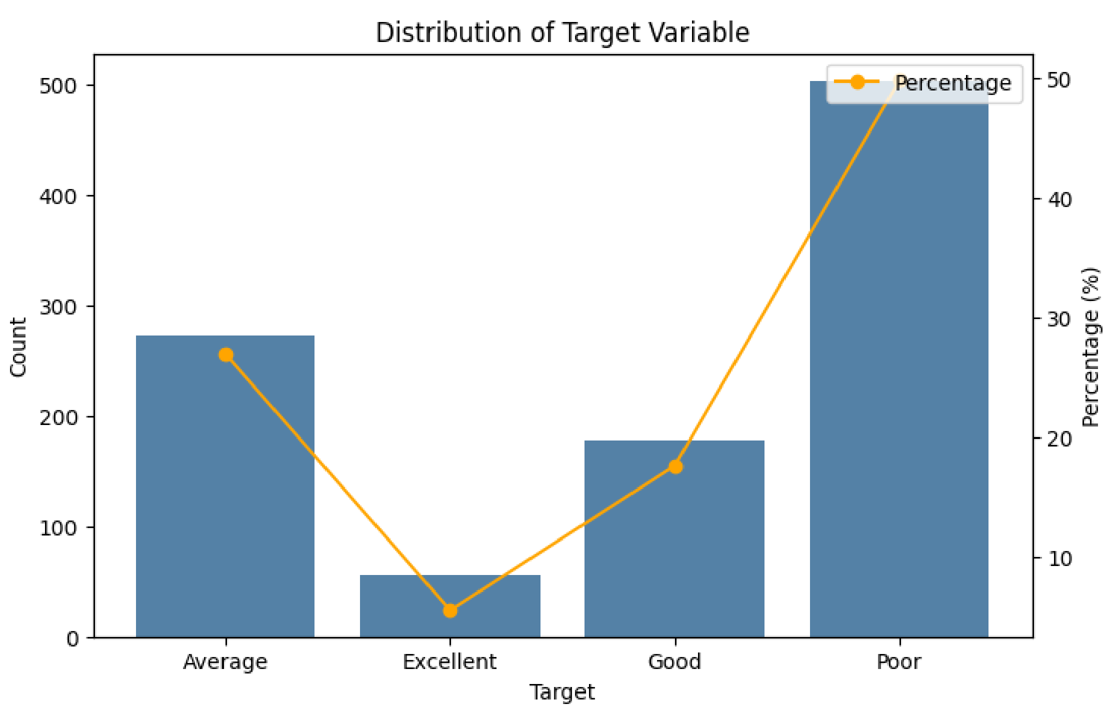
[34]:
```
print("Dataset Insights:", dataset_insights)
```

```
Dataset Insights:
provide a detailed analysis on the dataset, its dimensions, information, issues,
…

The dataset contains information on 1,009 students and consists of 45 columns,
including demographic details, academic history, behavioural metrics,
and performance indicators. It offers a comprehensive view of factors
potentially influencing student outcomes.

Only 'skills' (1 missing value) and 'area_of_interest' (7 missing values) have
missing data,
and no duplicate rows were found, indicating strong data integrity.

The dataset includes a mix of data types:
- Numerical (e.g., prevoius_gpa, average_attendance, study_hours,
social_media_hours)
- Categorical (e.g., gender, program, skills)
- Boolean (e.g., has_diploma)

Some variables, like 'current _semester', contain inconsistent naming (extra
space), which may require renaming for compatibility.
Additionally, columns such as 'admission_year' and 'hsc_year' include data entry
errors (e.g., admission_year max = 22022) that need correction.

Socioeconomic indicators like 'house_income' show a large range (min = 2,530 to
max = 2,000,000), suggesting possible data skew or outliers.

Overall, the dataset is suitable for predictive modeling but requires
preprocessing,
including error correction, missing value handling, and minor cleaning of column
names.
```

1.4.2 C.2 Explore Target Variable

[ ]:
```
# <Student to fill this section>
target_name = 'target'
```

[ ]:
```
# Count and percentage
target_counts = df['target'].value_counts().sort_index()
target_percent = (target_counts / target_counts.sum()) * 100

fig, ax1 = plt.subplots(figsize=(8, 5))

sns.countplot(x='target', data=df, order=target_counts.index, ax=ax1,␣
↪color='#4682B4')
ax1.set_ylabel("Count")
ax1.set_xlabel("Target")
ax1.set_title("Distribution of Target Variable")

ax2 = ax1.twinx()
ax2.plot(target_counts.index, target_percent, color='orange', marker='o',␣
↪label='Percentage')
ax2.set_ylabel("Percentage (%)")
ax2.legend(loc="upper right")

plt.show()
```



[ ]:
```
target_counts = df['target'].value_counts(normalize=True)
print("Target proportions:\n", target_counts)
```

```
Target proportions:
 target
Poor         0.498513
Average      0.269574
Good         0.176412
Excellent    0.055500
Name: proportion, dtype: float64
```

[35]:
```
# <Student to fill this section>
target_insights = """
provide a detailed analysis on the target variable, its distribution,␣
↪limitations, issues, ...

The target variable represents student performance categorized into four levels:
↪ Poor, Average, Good, and Excellent.
Its distribution is highly imbalanced, with nearly 50% of students classified␣
↪as 'Poor', followed by 27% as 'Average'.
Only 17.6% are labeled as 'Good', and just 5.5% as 'Excellent'.

This imbalance presents a significant challenge for classification models, as␣
↪they may become biased toward predicting the majority class ('Poor'),
leading to low recall for minority classes like 'Excellent'.

The limited number of 'Excellent' cases makes it difficult for the model to␣
↪learn distinguishing patterns for high-performing students,
which may affect both model performance and fairness.

To address this, techniques such as stratified sampling or resampling methods␣
↪(e.g., SMOTE) may be needed during model training.

The visual distribution further reinforces the imbalance, emphasizing the␣
↪importance of using appropriate evaluation metrics
(e.g., recall, F1-score) and validation strategies that account for minority␣
↪class representation.
"""
```

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

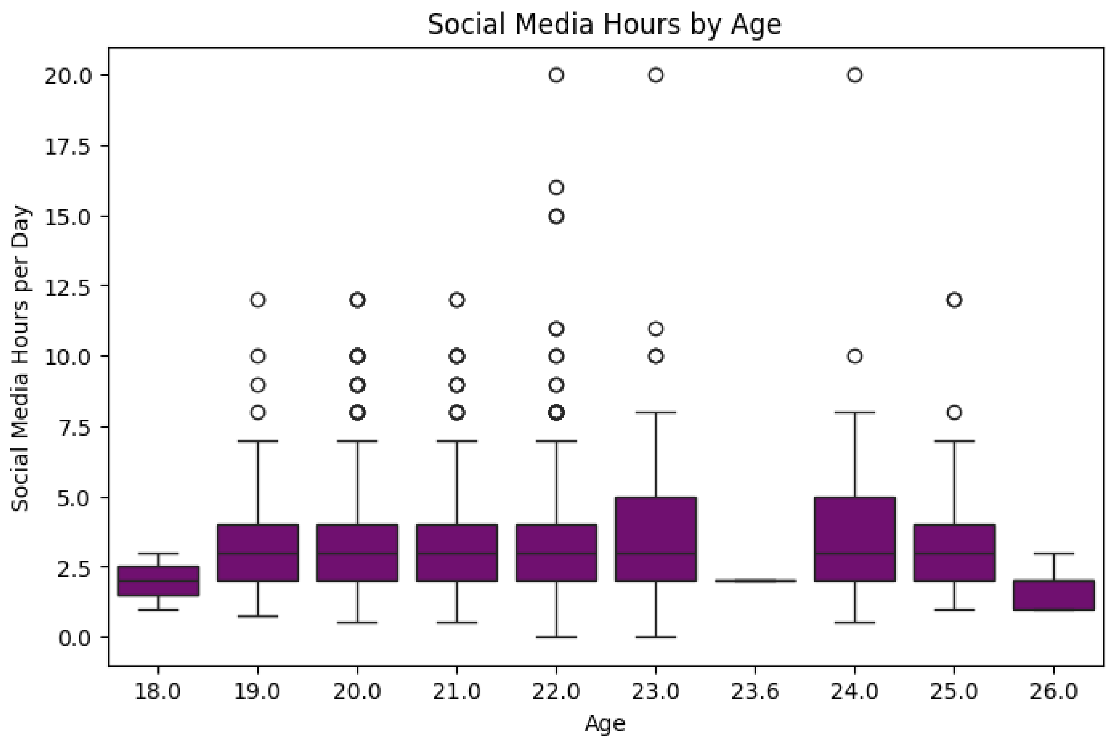
```
<IPython.core.display.HTML object>
```

[36]:
```
print("Target Insights:", target_insights)
```

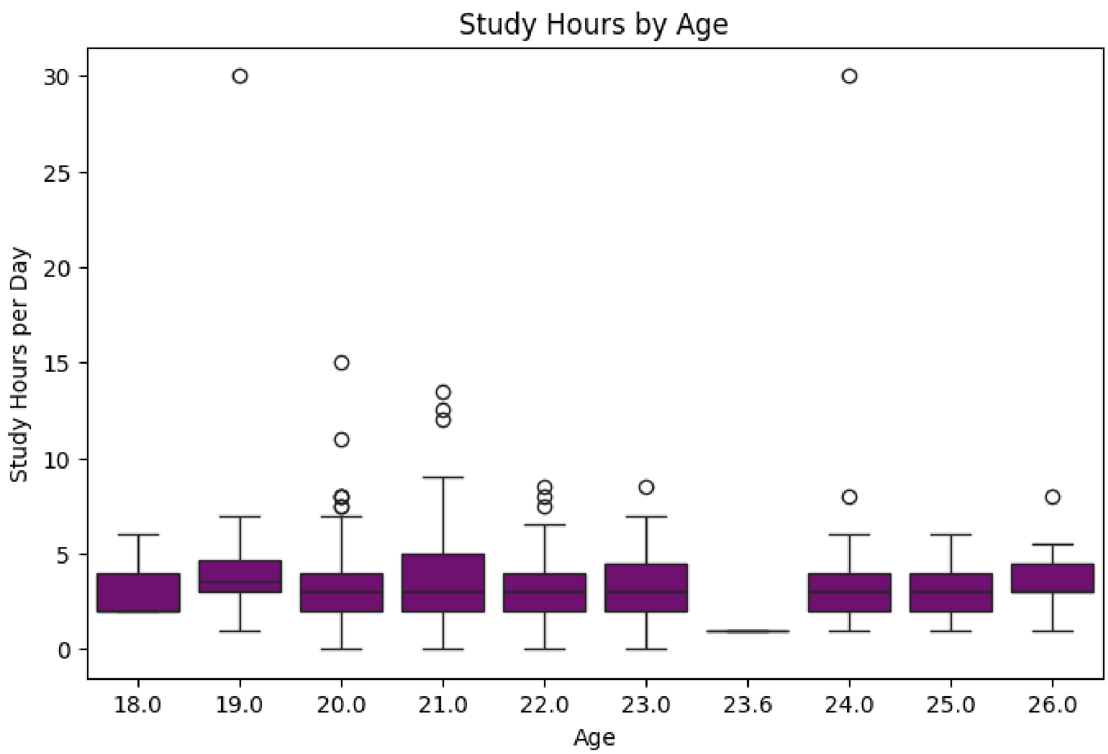
```
Target Insights:
provide a detailed analysis on the target variable, its distribution,
limitations, issues, …

The target variable represents student performance categorized into four levels:
Poor, Average, Good, and Excellent.
Its distribution is highly imbalanced, with nearly 50% of students classified as
'Poor', followed by 27% as 'Average'.
Only 17.6% are labeled as 'Good', and just 5.5% as 'Excellent'.

This imbalance presents a significant challenge for classification models, as
they may become biased toward predicting the majority class ('Poor'),
leading to low recall for minority classes like 'Excellent'.

The limited number of 'Excellent' cases makes it difficult for the model to
learn distinguishing patterns for high-performing students,
which may affect both model performance and fairness.

To address this, techniques such as stratified sampling or resampling methods
(e.g., SMOTE) may be needed during model training.

The visual distribution further reinforces the imbalance, emphasizing the
importance of using appropriate evaluation metrics
(e.g., recall, F1-score) and validation strategies that account for minority
class representation.
```

1.4.3 C.4 Explore Feature of Interest age

[ ]:
```
# <Student to fill this section>
plt.figure(figsize=(8, 5))
sns.boxplot(x='age', y='social_media_hours', data=df, color='purple')
plt.title('Social Media Hours by Age')
plt.xlabel('Age')
plt.ylabel('Social Media Hours per Day')
plt.show()
```



[ ]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(x='age', y='study_hours', data=df, color='purple')
plt.title('Study Hours by Age')
plt.xlabel('Age')
plt.ylabel('Study Hours per Day')
plt.show()
```



[37]:
```
# <Student to fill this section>
feature_1_insights = """
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...

The feature 'age' is a discrete numerical variable ranging from 18 to 26.

From the 'Social Media Hours by Age' plot, a wider spread of outliers is␣
↪observed among students aged around 19 to 25.
Notably, some students aged 22 or 23 report spending no time on social media,␣
↪while others in the same age group spend up to 20 hours per day.
This suggests that students between 19 and 23 are generally more active on␣
↪social platforms,
which may influence their study habits and academic focus.

In contrast, the 'Study Hours by Age' plot reveals a consistent median across␣
↪most age groups, typically around 3-4 hours per day,
though some extreme outliers are present. There is no clear age-related trend␣
↪in study hours, indicating that age alone may not directly influence
study discipline, but could interact with other behavioral or contextual␣
↪features.
```

```
A key limitation is that age is unevenly distributed. This reduces its␣
↪effectiveness as a standalone predictor, though it may still add value when␣
↪combined with other features.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_1_insights', value=feature_1_insights)
```

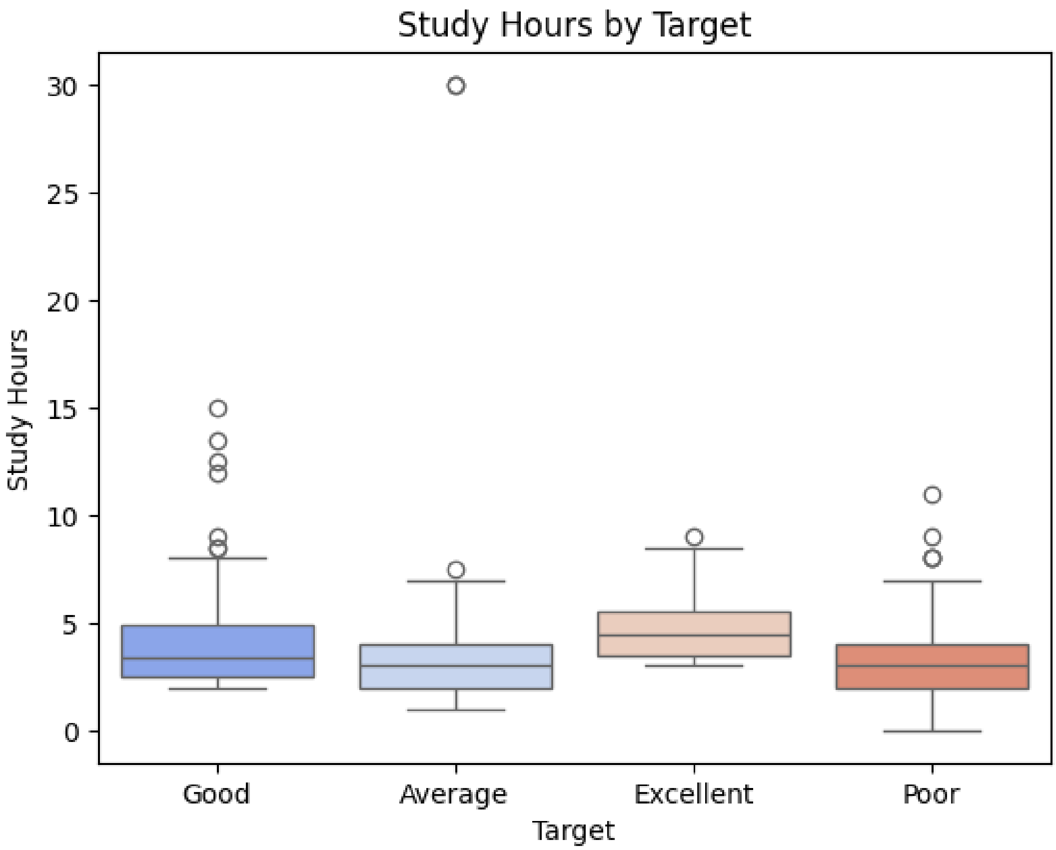
```
<IPython.core.display.HTML object>
```

[38]:
```
print("Feature 1 Insights:", feature_1_insights)
```

```
Feature 1 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The feature 'age' is a discrete numerical variable ranging from 18 to 26.

From the 'Social Media Hours by Age' plot, a wider spread of outliers is
observed among students aged around 19 to 25.
Notably, some students aged 22 or 23 report spending no time on social media,
while others in the same age group spend up to 20 hours per day.
This suggests that students between 19 and 23 are generally more active on
social platforms,
which may influence their study habits and academic focus.

In contrast, the 'Study Hours by Age' plot reveals a consistent median across
most age groups, typically around 3-4 hours per day,
though some extreme outliers are present. There is no clear age-related trend in
study hours, indicating that age alone may not directly influence
study discipline, but could interact with other behavioral or contextual
features.

A key limitation is that age is unevenly distributed. This reduces its
effectiveness as a standalone predictor, though it may still add value when
combined with other features.
```

1.4.4 C.5 Explore Feature of Interest study_hours

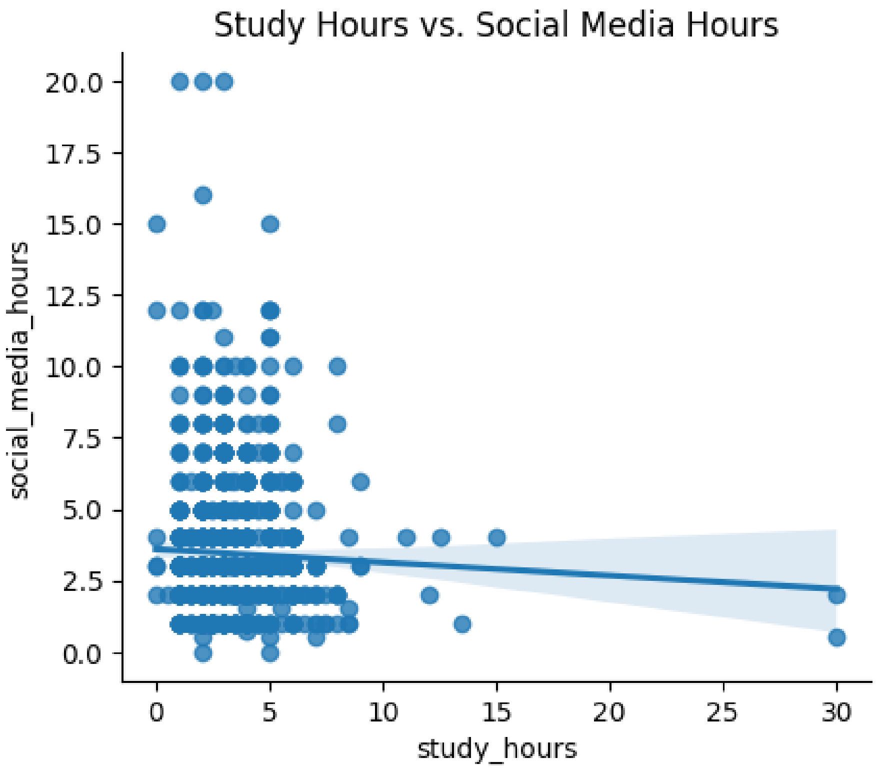
[ ]:
```
# <Student to fill this section>
sns.boxplot(x='target', y='study_hours', data=df, palette="coolwarm")
plt.title("Study Hours by Target")
plt.xlabel("Target")
plt.ylabel("Study Hours")
plt.show()
```



[ ]:
```
print(df.groupby('target')['study_hours'].describe())
```

```
           count      mean       std  min  25%      50%    75%   max
target
Average    272.0  3.156354  2.785342  1.0  2.0  3.00000  4.000  30.0
Excellent   56.0  4.774063  1.642459  3.0  3.5  4.50000  5.500   9.0
Good       178.0  3.862599  2.111532  2.0  2.5  3.34753  4.875  15.0
Poor       503.0  3.083915  1.536852  0.0  2.0  3.00000  4.000  11.0
```

[ ]:
```
sns.lmplot(x='study_hours', y='social_media_hours', data=df, height=4, aspect=1.
↪2)
plt.title('Study Hours vs. Social Media Hours')
plt.show()
```



[39]:
```
# <Student to fill this section>
feature_2_insights = """
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...

The feature 'study_hours' captures the average number of hours students spend␣
↪studying each day.

The 'Study Hours by Target' boxplot shows that 'Excellent' and 'Good' students␣
↪tend to study more, with medians around 4.5 and 3.3 hours.
In contrast, 'Average' and 'Poor' groups show lower medians and more␣
↪variability, especially in the 'Poor' group.
This suggests that although some students invest significant time in studying,
it does not always translate to better academic outcomes, possibly due to␣
↪ineffective study methods or external challenges.

Descriptive stats confirm this trend: the 'Excellent' group has the highest␣
↪mean (4.77), while 'Poor' is lowest (3.08).
Some high outliers (e.g. 30 hours) may indicate misreporting or rare intensive␣
↪effort.
```

```
The scatter plot shows a weak negative trend with social media usage-students␣
↪who study more tend to use social media less,
though the relationship is not strong.

Study hours appear moderately linked to performance but may be affected by␣
↪factors like study quality, motivation, or time management.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_2_insights', value=feature_2_insights)
```

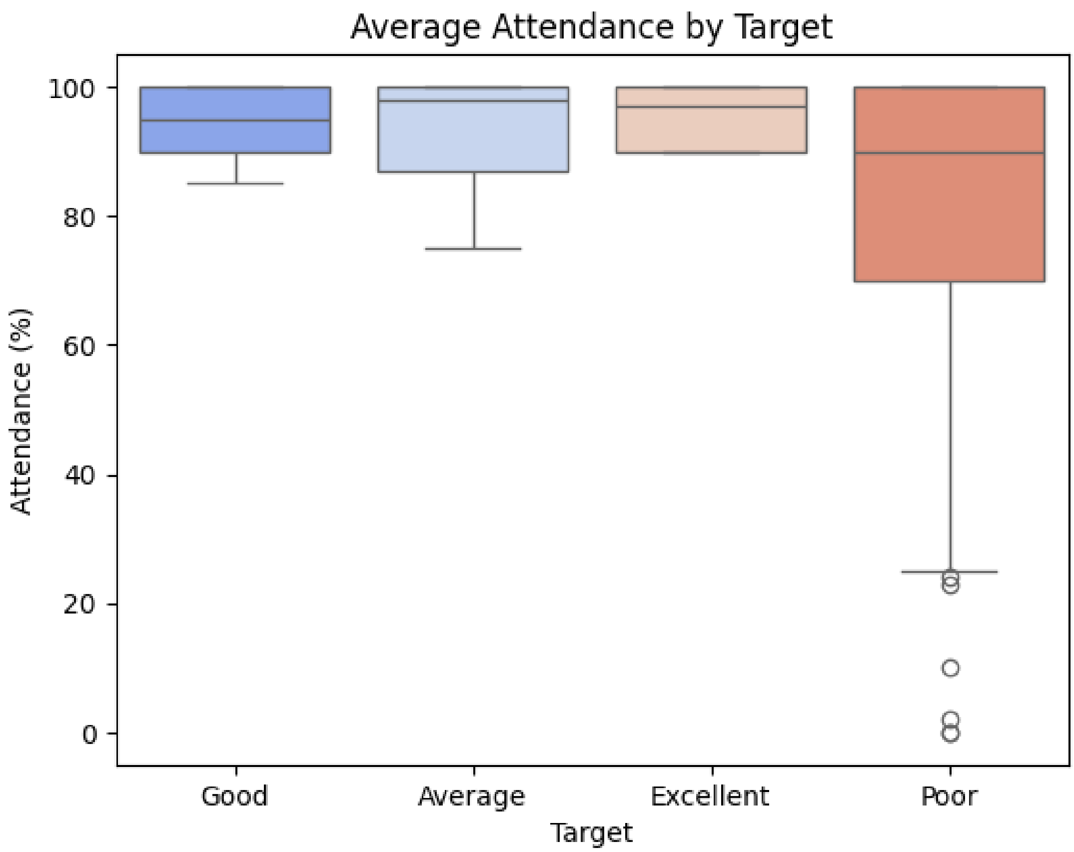
```
<IPython.core.display.HTML object>
```

[40]:
```
print("Feature 2 Insights:", feature_2_insights)
```

```
Feature 2 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The feature 'study_hours' captures the average number of hours students spend
studying each day.

The 'Study Hours by Target' boxplot shows that 'Excellent' and 'Good' students
tend to study more, with medians around 4.5 and 3.3 hours.
In contrast, 'Average' and 'Poor' groups show lower medians and more
variability, especially in the 'Poor' group.
This suggests that although some students invest significant time in studying,
it does not always translate to better academic outcomes, possibly due to
ineffective study methods or external challenges.

Descriptive stats confirm this trend: the 'Excellent' group has the highest mean
(4.77), while 'Poor' is lowest (3.08).
Some high outliers (e.g. 30 hours) may indicate misreporting or rare intensive
effort.

The scatter plot shows a weak negative trend with social media usage-students
who study more tend to use social media less,
though the relationship is not strong.

Study hours appear moderately linked to performance but may be affected by
factors like study quality, motivation, or time management.
```

1.4.5 C.6 Explore Feature of Interest average_attendance

[ ]:
```
# <Student to fill this section>
sns.boxplot(x='target', y='average_attendance', data=df, palette="coolwarm")
plt.title("Average Attendance by Target")
plt.xlabel("Target")
plt.ylabel("Attendance (%)")
plt.show()
```

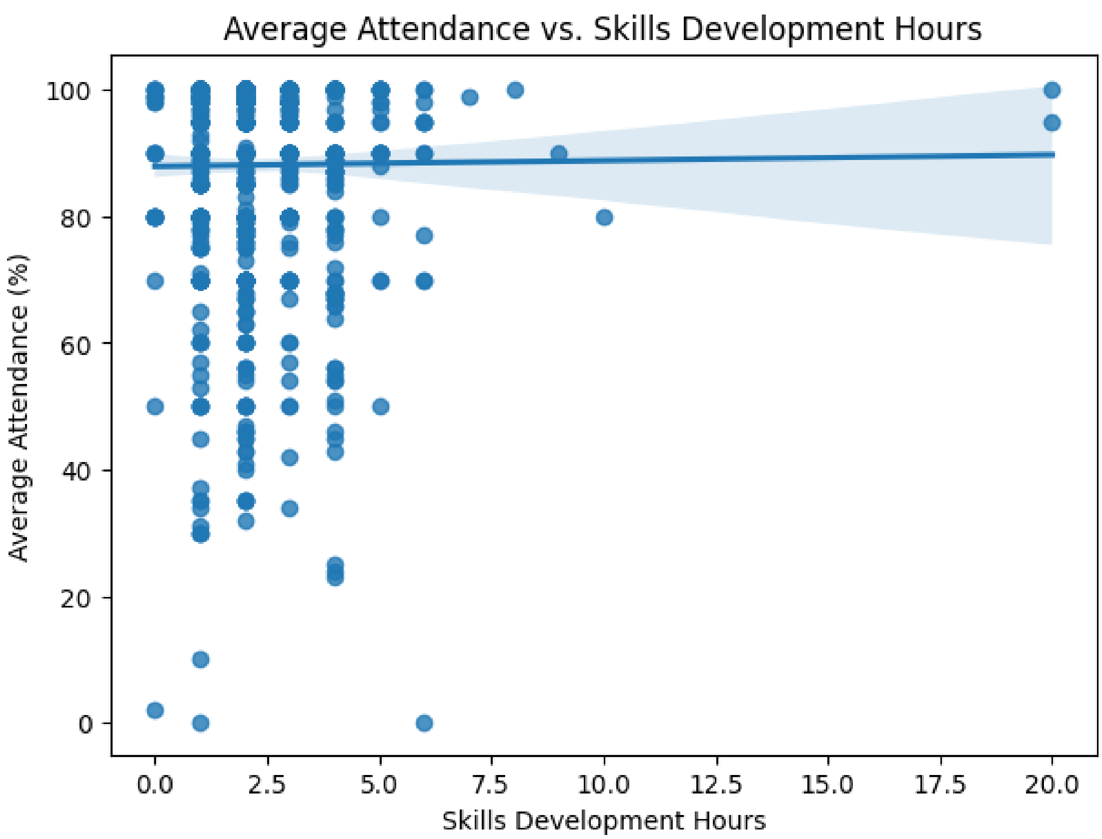


[ ]:
```
print(df.groupby('target')['average_attendance'].describe())
```

```
           count       mean        std   min   25%   50%    75%    max
target
Average    272.0  93.169118   8.144502  75.0  87.0  98.0  100.0  100.0
Excellent   56.0  95.714286   4.180971  90.0  90.0  97.0  100.0  100.0
Good       178.0  95.337079   4.436479  85.0  90.0  95.0  100.0  100.0
Poor       503.0  81.972167  19.947022   0.0  70.0  90.0  100.0  100.0
```

[ ]:
```
plt.figure(figsize=(7, 5))
sns.regplot(x='skills_development_hours', y='average_attendance', data=df)
```

```
plt.title('Average Attendance vs. Skills Development Hours')
plt.xlabel('Skills Development Hours')
plt.ylabel('Average Attendance (%)')
plt.show()
```



[41]:
```
# <Student to fill this section>
feature_3_insights = """
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...

The feature 'average_attendance' represents the percentage of classes a student␣
↪attended and ranges from 0% to 100%.

The analysis shows students in the 'Excellent', 'Good', and 'Average'␣
↪categories tend to have consistently high attendance,
with medians above 95%. In contrast, the 'Poor' group shows more variation,␣
↪including several students with extremely low or zero attendance.
This highlights attendance as a strong distinguishing feature between lower and␣
↪higher performing students.
```

```
The scatter plot comparing attendance with skills development hours shows a␣
  ↪slight positive trend,
indicating that students who attend more classes may also invest more in␣
  ↪developing their skills, though the correlation is weak.

A limitation of this feature is the ceiling effect: many students report 100%␣
  ↪attendance, which may reduce its sensitivity as a predictor at the high end.
Additionally, attendance alone may not capture the quality of engagement or␣
  ↪participation.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_3_insights', value=feature_3_insights)
```

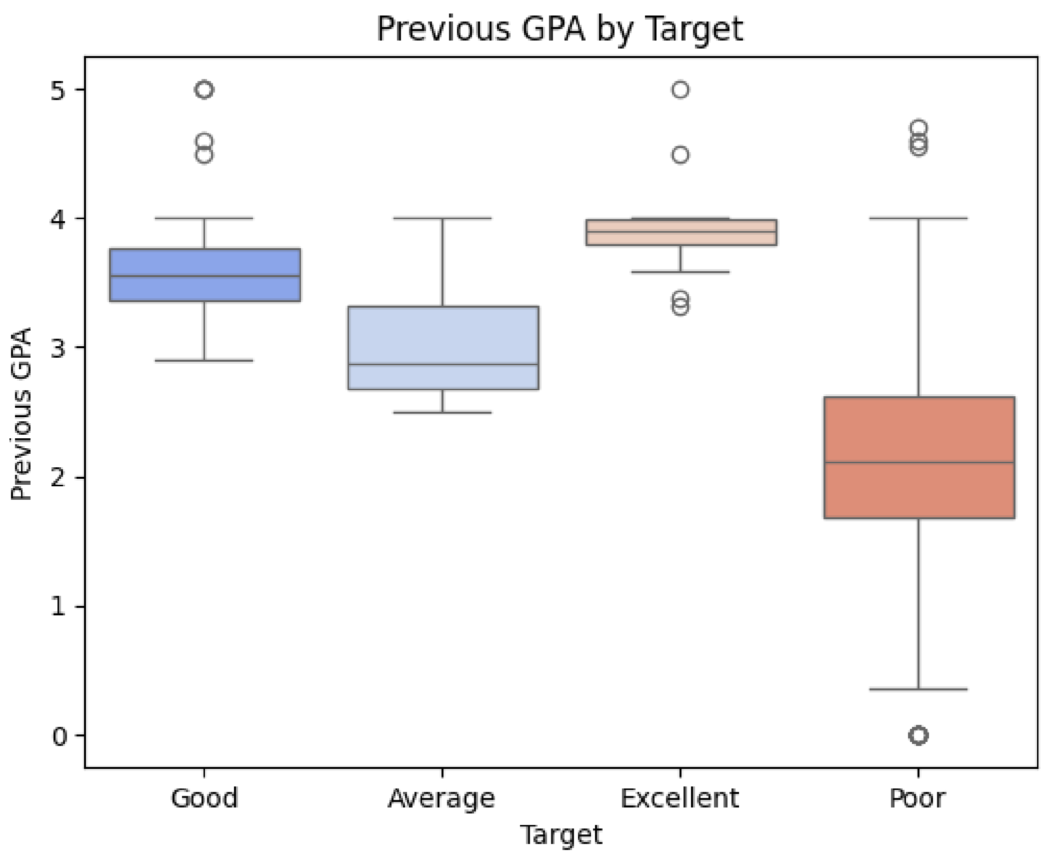
```
<IPython.core.display.HTML object>
```

[42]:
```
print("Feature 3 Insights:", feature_3_insights)
```

```
Feature 3 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The feature 'average_attendance' represents the percentage of classes a student
attended and ranges from 0% to 100%.

The analysis shows students in the 'Excellent', 'Good', and 'Average' categories
tend to have consistently high attendance,
with medians above 95%. In contrast, the 'Poor' group shows more variation,
including several students with extremely low or zero attendance.
This highlights attendance as a strong distinguishing feature between lower and
higher performing students.

The scatter plot comparing attendance with skills development hours shows a
slight positive trend,
indicating that students who attend more classes may also invest more in
developing their skills, though the correlation is weak.

A limitation of this feature is the ceiling effect: many students report 100%
attendance, which may reduce its sensitivity as a predictor at the high end.
Additionally, attendance alone may not capture the quality of engagement or
participation.
```

1.4.6 C.7 Explore Feature of Interest previous_gpa

[ ]:
```
sns.boxplot(x='target', y='previous_gpa', data=df, palette="coolwarm")
plt.title("Previous GPA by Target")
plt.xlabel("Target")
plt.ylabel("Previous GPA")
plt.show()
```



[ ]:
```
print(df.groupby('target')['previous_gpa'].describe())
```

```
            count      mean       std   min     25%   50%     75%  max
target
Average     272.0  3.019449  0.415803  2.50  2.6700  2.87  3.3225  4.0
Excellent    56.0  3.884286  0.226907  3.31  3.7975  3.90  3.9800  5.0
Good        178.0  3.567416  0.345559  2.90  3.3625  3.56  3.7575  5.0
Poor        503.0  2.201750  0.785035  0.00  1.6750  2.11  2.6150  4.7
```

[43]:
```
# <Student to fill this section>
feature_4_insights = """
```

```
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...

The feature 'previous_gpa' is a continuous numerical variable representing␣
↪students’ academic performance at the beginning of the previous semester.

The boxplot reveals a strong relationship between previous GPA and current␣
↪performance categories.
'Excellent' students show the highest GPA distribution with a narrow spread,␣
↪centered around 3.9,
while the 'Poor' group displays a wide spread with a median near 2.1 and␣
↪several low outliers even below 1.0.

This suggests that previous academic success is a strong predictor of current␣
↪performance.
However, some outliers-students with previously high GPAs now in the 'Poor'␣
↪category-indicate that GPA alone may not be sufficient for accurately␣
↪identifying at-risk students.

A limitation is that GPA scales may differ between contexts or programs.
Still, this feature provides one of the most consistent and predictive patterns␣
↪in the dataset.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_4_insights', value=feature_4_insights)
```

<IPython.core.display.HTML object>

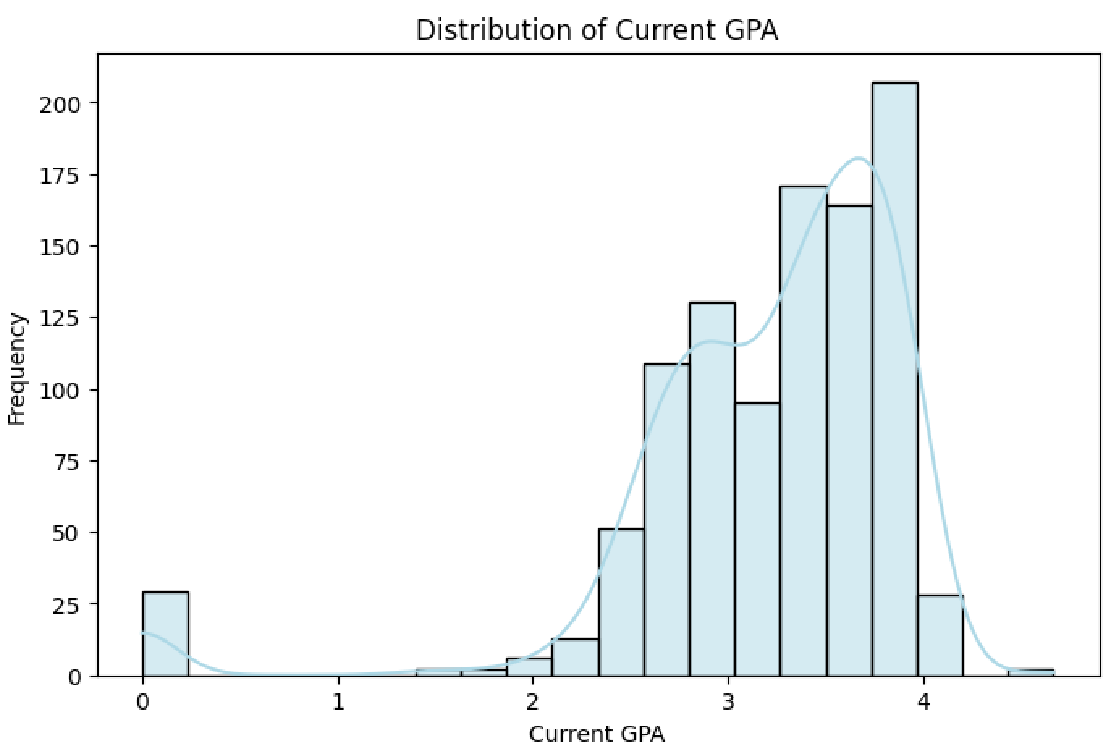
[45]:
```
print("Feature 4 Insights:", feature_4_insights)
```

```
Feature 4 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The feature 'previous_gpa' is a continuous numerical variable representing
students’ academic performance at the beginning of the previous semester.

The boxplot reveals a strong relationship between previous GPA and current
performance categories.
'Excellent' students show the highest GPA distribution with a narrow spread,
centered around 3.9,
while the 'Poor' group displays a wide spread with a median near 2.1 and several
low outliers even below 1.0.

This suggests that previous academic success is a strong predictor of current
performance.
```

```
However, some outliers-students with previously high GPAs now in the 'Poor'
category-indicate that GPA alone may not be sufficient for accurately
identifying at-risk students.

A limitation is that GPA scales may differ between contexts or programs.
Still, this feature provides one of the most consistent and predictive patterns
in the dataset.
```

1.4.7 C.8 Explore Feature of Interest current_gpa

[ ]:
```python
plt.figure(figsize=(8, 5))
sns.histplot(df['current_gpa'], bins=20, kde=True, color='lightblue')
plt.title("Distribution of Current GPA")
plt.xlabel("Current GPA")
plt.ylabel("Frequency")
plt.show()
```



[46]:
```python
feature_5_insights = """
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...
```

```
The feature 'current_gpa' is a continuous variable showing a right-skewed␣
↪distribution,
with most students scoring between 3.0 and 4.0. A few outliers near 0 may␣
↪reflect poor performance or data issues.
The skewness suggests normalization may help in modeling. As a performance␣
↪indicator, it should be used carefully to avoid overlap with the target␣
↪variable.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_5_insights', value=feature_5_insights)
```

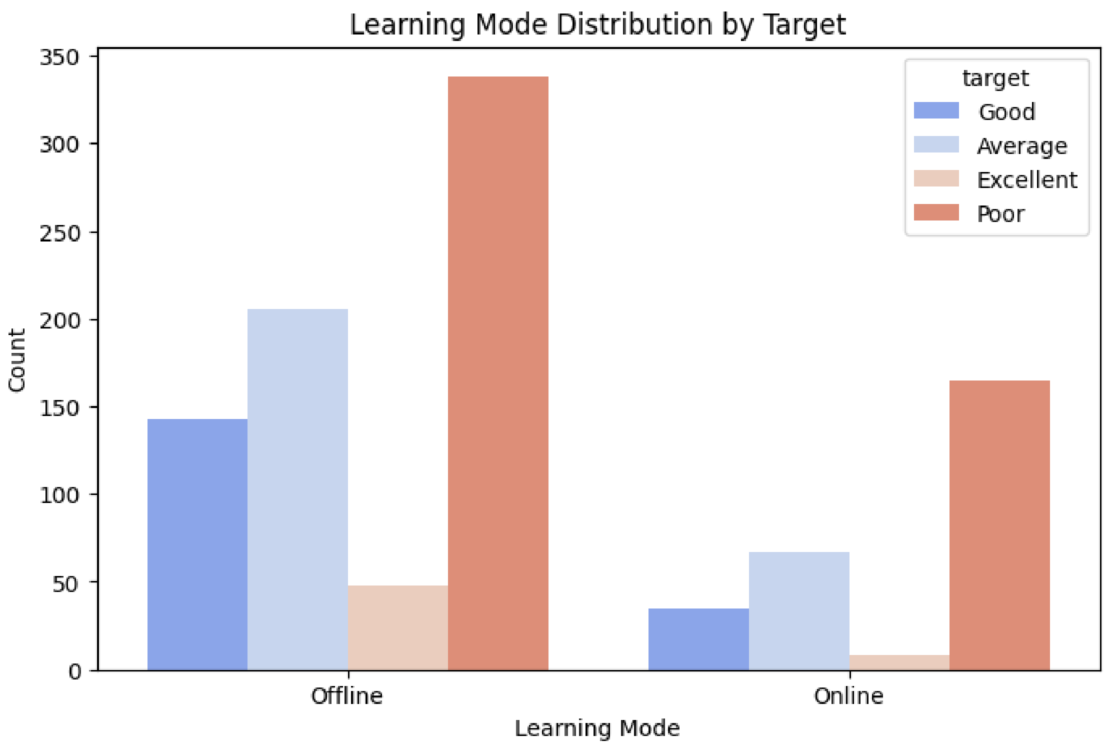
<IPython.core.display.HTML object>

[47]:
```
print("Feature 5 Insights:", feature_5_insights)
```

```
Feature 5 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The feature 'current_gpa' is a continuous variable showing a right-skewed
distribution,
with most students scoring between 3.0 and 4.0. A few outliers near 0 may
reflect poor performance or data issues.
The skewness suggests normalization may help in modeling. As a performance
indicator, it should be used carefully to avoid overlap with the target
variable.
```

1.4.8 C.9 Explore Feature of Interest learning_mode

[ ]:
```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='learning_mode', hue='target', palette='coolwarm')
plt.title("Learning Mode Distribution by Target")
plt.xlabel("Learning Mode")
plt.ylabel("Count")
plt.show()
```



[48]:
```
feature_6_insights = """
provide a detailed analysis on the selected feature, its distribution,␣
↪limitations, issues, ...

The 'learning_mode' feature is a categorical variable with two categories:␣
↪Online and Offline.
Most students in the 'Poor' and 'Average' categories are in offline mode,
while fewer high-performing students (Good or Excellent) appear in online mode.
This may suggest that learning mode is associated with academic performance.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_6_insights', value=feature_6_insights)
```

<IPython.core.display.HTML object>

[49]:
```
print("Feature 6 Insights:", feature_6_insights)
```

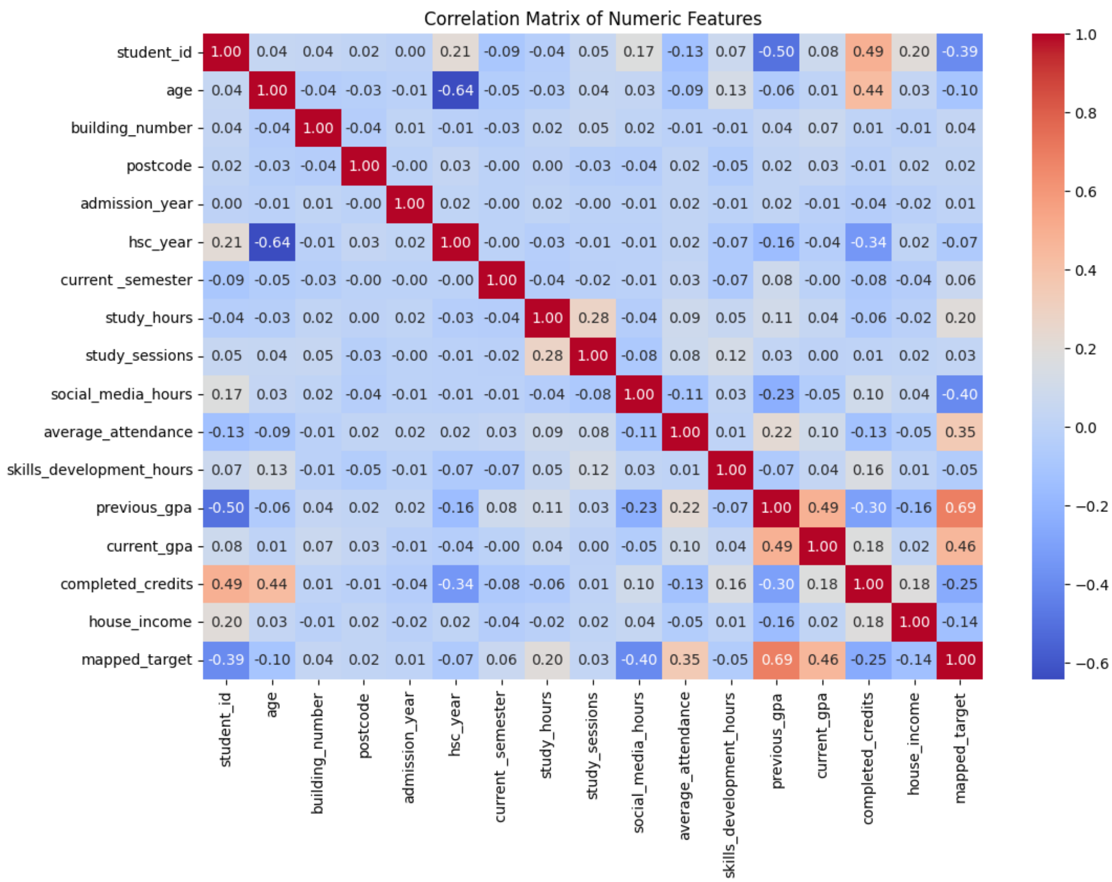
```
Feature 6 Insights:
provide a detailed analysis on the selected feature, its distribution,
limitations, issues, …

The 'learning_mode' feature is a categorical variable with two categories:
```

```
Online and Offline.
Most students in the 'Poor' and 'Average' categories are in offline mode,
while fewer high-performing students (Good or Excellent) appear in online mode.
This may suggest that learning mode is associated with academic performance.
```

1.5 D. Feature Selection

1.5.1 D.1 Approach "Correlation Matrix Including Encoded Target"

[ ]:
```
# <Student to fill this section>
df_mapped = df.copy()
df_mapped['mapped_target'] = df_mapped['target'].map({'Excellent': 3, 'Good':␣
↪2, 'Average': 1, 'Poor' : 0})
df_mapped.columns = df_mapped.columns.str.strip()

numeric_df = df_mapped.select_dtypes(include=['float64', 'int64'])

corr_matrix = numeric_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Matrix of Numeric Features")
plt.show()
```



[50]:

```
# <Student to fill this section>
feature_selection_1_insights = """
provide an explanation on why you use this approach for feature selection and␣
↪describe its results

Correlation analysis was used as the first approach for feature selection to␣
↪identify numerical variables that are most strongly associated
with the target variable and with each other. This method provides a␣
↪straightforward and interpretable way to assess linear relationships between␣
↪features.

From the heatmap, 'previous_gpa' (0.69), 'current_gpa' (0.46), and␣
↪'average_attendance' (0.35) showed the strongest positive correlations
with the mapped target, making them valuable predictors. On the other hand,␣
↪'social_media_hours' (-0.40) was negatively correlated with performance,
suggesting that higher social media usage is associated with a greater␣
↪likelihood of poor academic results in this semester.
Features like 'student_id' and 'postcode' showed little to no relevance and␣
↪were excluded.
```

```
This approach also helped identify variables that are highly correlated with␣
↪each other (e.g., 'previous_gpa' and 'current_gpa').
Overall, correlation analysis provided a quick and interpretable way to␣
↪prioritize relevant features and exclude redundant or non-informative ones.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_selection_1_insights',␣
↪value=feature_selection_1_insights)
```

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

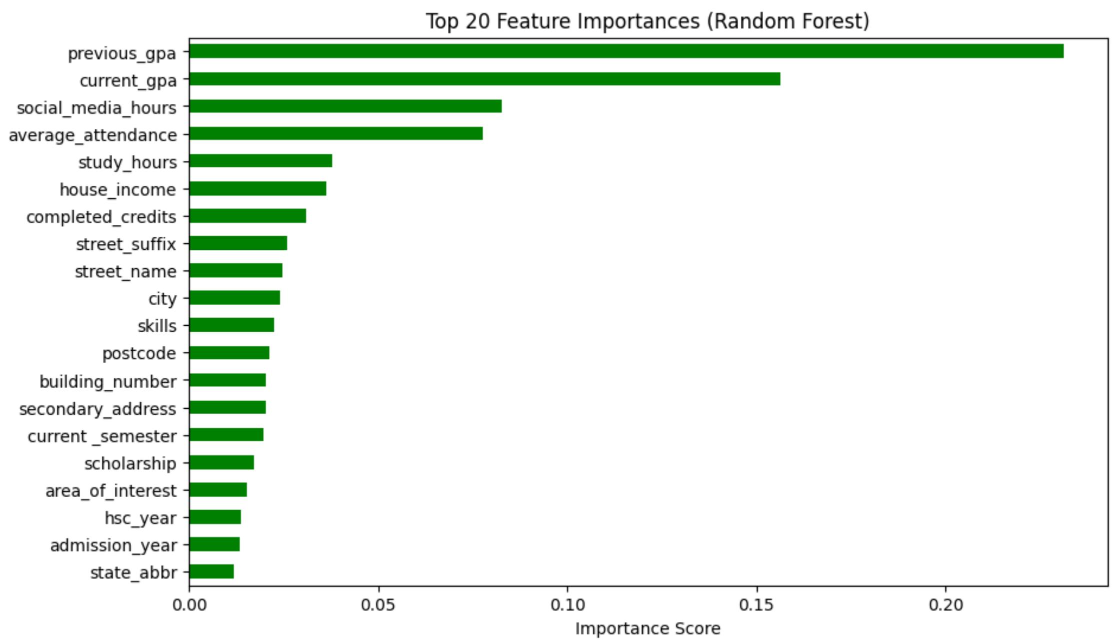
[51]:
```
print("Feature Selection 1 Insights:", feature_selection_1_insights)
```

```
Feature Selection 1 Insights:
provide an explanation on why you use this approach for feature selection and
describe its results

Correlation analysis was used as the first approach for feature selection to
identify numerical variables that are most strongly associated
with the target variable and with each other. This method provides a
straightforward and interpretable way to assess linear relationships between
features.

From the heatmap, 'previous_gpa' (0.69), 'current_gpa' (0.46), and
'average_attendance' (0.35) showed the strongest positive correlations
with the mapped target, making them valuable predictors. On the other hand,
'social_media_hours' (-0.40) was negatively correlated with performance,
suggesting that higher social media usage is associated with a greater
likelihood of poor academic results in this semester.
Features like 'student_id' and 'postcode' showed little to no relevance and were
excluded.

This approach also helped identify variables that are highly correlated with
each other (e.g., 'previous_gpa' and 'current_gpa').
Overall, correlation analysis provided a quick and interpretable way to
prioritize relevant features and exclude redundant or non-informative ones.
```

1.5.2 D.2 Approach "Feature Importance via Tree-Based Model"

[ ]:
```
# <Student to fill this section>
df_model = df.copy()

label_encoders = {}
for col in df_model.select_dtypes(include='object').columns:
```

```
    if col != 'target':
        le = LabelEncoder()
        df_model[col] = le.fit_transform(df_model[col].astype(str))
        label_encoders[col] = le

df_model['target_encoded'] = LabelEncoder().fit_transform(df_model['target'])

X = df_model.drop(columns=['student_id', 'full_name', 'email', 'phone_number',␣
 ↪'target', 'target_encoded'])
y = df_model['target_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Fit Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

importances = pd.Series(rf.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)

# Plot top 20 features
plt.figure(figsize=(10, 6))
importances[:20].plot(kind='barh', color='green')
plt.title("Top 20 Feature Importances (Random Forest)")
plt.gca().invert_yaxis()
plt.xlabel("Importance Score")
plt.show()

print(importances.head(20))
```




```
previous_gpa          0.231373
current_gpa           0.156369
social_media_hours    0.082872
average_attendance    0.077679
study_hours           0.037838
house_income          0.036463
completed_credits     0.030887
street_suffix         0.025873
street_name           0.024757
city                  0.024025
skills                0.022692
postcode              0.021341
building_number       0.020369
secondary_address     0.020331
current _semester     0.019608
scholarship           0.017182
area_of_interest      0.015329
hsc_year              0.013813
admission_year        0.013421
state_abbr            0.012009
dtype: float64
```

[ ]:
```
df['street_name'].value_counts()
```

[ ]:
```
street_name
Kayla Key            2
```

```
Laura Outlook        2
Amanda Triangle      2
Jennifer Chase       2
Robert Anchorage     2
                    ..
Simmons Front        1
Barker Parklands     1
Best Thoroughfare    1
Lindsey Boulevard    1
John Tor             1
Name: count, Length: 1003, dtype: int64
```

[ ]:
```
df['city'].value_counts()
```

[ ]:
```
city
Johnsonmouth       3
Kimberlyberg       3
Hicksshire         2
New Victoria       2
New Mary           2
                  ..
Middletonfurt      1
West Tiffanytown   1
Dawnville          1
West Heather       1
New Angelberg      1
Name: count, Length: 985, dtype: int64
```

[52]:
```
# <Student to fill this section>
feature_selection_2_insights = """
provide an explanation on why you use this approach for feature selection and␣
↪describe its results

The second approach for feature selection used a Random Forest model to␣
↪evaluate feature importance based on how much each variable contributed
to reducing prediction error. This method is useful as it captures non-linear␣
↪relationships and interactions between variables, making it more robust
than simple correlation-based approaches.

While the model initially highlighted features like 'street_name' and 'city'␣
↪among the top 20, these variables were found to have high cardinality
with nearly unique values for each student. Such features are unlikely to␣
↪provide generalizable insights and may lead to overfitting,
as the model could memorize rather than learn patterns.

More reliable features identified by this method include 'previous_gpa',␣
↪'current_gpa', 'social_media_hours', and 'average_attendance',
```

```
which also aligned with the findings from the correlation-based approach. These␣
↪features showed measurable influence on the target
and are more interpretable for modeling and decision-making.

Overall, Random Forest feature importance helped validate key predictors while␣
↪flagging irrelevant or potentially misleading ones,
supporting a refined and balanced feature set for model training.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_selection_2_insights',␣
↪value=feature_selection_2_insights)
```

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

[53]:
```
print("Feature Selection 2 Insights:", feature_selection_2_insights)
```

```
Feature Selection 2 Insights:
provide an explanation on why you use this approach for feature selection and
describe its results

The second approach for feature selection used a Random Forest model to evaluate
feature importance based on how much each variable contributed
to reducing prediction error. This method is useful as it captures non-linear
relationships and interactions between variables, making it more robust
than simple correlation-based approaches.

While the model initially highlighted features like 'street_name' and 'city'
among the top 20, these variables were found to have high cardinality
with nearly unique values for each student. Such features are unlikely to
provide generalizable insights and may lead to overfitting,
as the model could memorize rather than learn patterns.

More reliable features identified by this method include 'previous_gpa',
'current_gpa', 'social_media_hours', and 'average_attendance',
which also aligned with the findings from the correlation-based approach. These
features showed measurable influence on the target
and are more interpretable for modeling and decision-making.

Overall, Random Forest feature importance helped validate key predictors while
flagging irrelevant or potentially misleading ones,
supporting a refined and balanced feature set for model training.
```

1.5.3 D.3 Approach "Recursive Feature Elimination (RFE)"

[ ]:
```
df_rfe = df.copy()
df_rfe.columns = df_rfe.columns.str.strip()

# Encode categorical variables
label_encoders = {}
for col in df_rfe.select_dtypes(include='object').columns:
    if col != 'target':
        le = LabelEncoder()
        df_rfe[col] = le.fit_transform(df_rfe[col].astype(str))
        label_encoders[col] = le

target_mapping = {'Poor': 0, 'Average': 1, 'Good': 2, 'Excellent': 3}
df_rfe['target_encoded'] = df_rfe['target'].map(target_mapping)

X = df_rfe.drop(columns=['student_id', 'full_name', 'email', 'phone_number',␣
↪'target', 'target_encoded'])
y = df_rfe['target_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

model = LogisticRegression(max_iter=1000)

# Train model
rfe = RFE(model, n_features_to_select=10)
rfe.fit(X_train, y_train)

selected_rfe_features = X.columns[rfe.support_]
print("Top 10 Features Selected by RFE:\n", selected_rfe_features)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Top 10 Features Selected by RFE:
 Index(['gender', 'university_transport', 'learning_mode', 'has_phone',
       'has_laptop', 'on_probation', 'relationship', 'social_media_hours',
       'previous_gpa', 'current_gpa'],
      dtype='object')
```

[54]:
```
# <Student to fill this section>
feature_selection_3_insights = """
provide an explanation on why you use this approach for feature selection and␣
↪describe its results

Recursive Feature Elimination (RFE) was used as the third approach to␣
↪iteratively select the top features based on their contribution to model␣
↪performance.
It is especially useful for identifying a compact subset of features that␣
↪collectively offer strong predictive power.

RFE selected a mix of behavioral, academic, and contextual features, including␣
↪'previous_gpa', 'current_gpa', and 'social_media_hours',
as well as categorical variables like 'gender', 'learning_mode', and␣
↪'on_probation'. This indicates that both numerical and categorical factors
play a significant role in predicting student performance.

Compared to previous approaches, RFE provided a balanced view by including␣
↪variables that may not have shown strong individual correlation,
but contribute meaningfully in combination with others.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_selection_3_insights',␣
↪value=feature_selection_3_insights)
```

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

[55]:
```
print("Feature Selection 3 Insights:", feature_selection_3_insights)
```

```
Feature Selection 3 Insights:
provide an explanation on why you use this approach for feature selection and
describe its results

Recursive Feature Elimination (RFE) was used as the third approach to
iteratively select the top features based on their contribution to model
performance.
It is especially useful for identifying a compact subset of features that
collectively offer strong predictive power.

RFE selected a mix of behavioral, academic, and contextual features, including
'previous_gpa', 'current_gpa', and 'social_media_hours',
as well as categorical variables like 'gender', 'learning_mode', and
'on_probation'. This indicates that both numerical and categorical factors
play a significant role in predicting student performance.

Compared to previous approaches, RFE provided a balanced view by including
```

variables that may not have shown strong individual correlation,
but contribute meaningfully in combination with others.

1.5.4 D.3 Approach "Ethical Consideration"

[ ]:
```
categorical_cols = df.select_dtypes(include='object').columns
high_cardinality = df[categorical_cols].nunique().sort_values()
high_cardinality[high_cardinality > 100]
```

[ ]:
```
street_suffix          200
secondary_address      587
city                   985
full_name             1001
street_name           1003
email                 1006
phone_number          1009
dtype: int64
```

[56]:
```
# <Student to fill this section>
feature_selection_4_insights = """
provide an explanation on why you use this approach for feature selection and␣
  ↪describe its results

This feature selection approach was based on identifying low-variance and␣
  ↪low-importance features by observing the number of unique values.
Features like 'full_name', 'email', and 'phone_number' have nearly unique␣
  ↪values for every student,
offering no generalizable patterns for prediction. As such, they were excluded␣
  ↪from modeling.
This helps reduce noise, improve model performance, and prevent overfitting.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_selection_4_insights',␣
  ↪value=feature_selection_4_insights)
```

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

[57]:
```
print("Feature Selection 4 Insights:", feature_selection_4_insights)
```

```
Feature Selection 4 Insights:
provide an explanation on why you use this approach for feature selection and
describe its results

This feature selection approach was based on identifying low-variance and low-
importance features by observing the number of unique values.
Features like 'full_name', 'email', and 'phone_number' have nearly unique values
```

```
for every student,
offering no generalizable patterns for prediction. As such, they were excluded
from modeling.
This helps reduce noise, improve model performance, and prevent overfitting.
```

1.6 D.z Final Selection of Features

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

features_list = [
    'age','gender','state_abbr','admission_year','hsc_year','scholarship',
  ␣
 ↪'university_transport','learning_mode','has_phone','has_laptop','english_proficiency',
  ␣
 ↪'on_probation','is_suspended','has_consulted_teacher','relationship','co_curricular',
    'living_arrangement','health_issues','disabilities','target','current␣
 ↪_semester','study_hours',
  ␣
 ↪'study_sessions','social_media_hours','average_attendance','skills','skills_development_hou
  ␣
 ↪'area_of_interest','previous_gpa','current_gpa','completed_credits','has_diploma','house_in
]
```

[58]:
```python
# <Student to fill this section>
feature_selection_explanations = """
provide a quick explanation on the features selected

The final set of selected features includes a balanced mix of demographic,␣
 ↪academic, behavioral, and contextual variables.
This selection was informed by three approaches-correlation analysis, Random␣
 ↪Forest importance, and RFE to ensure both individual relevance and combined␣
 ↪predictive power,
while also considering ethical implications.

Key academic predictors such as 'previous_gpa', 'current_gpa', 'study_hours',␣
 ↪and 'average_attendance' were included due to their strong association with␣
 ↪performance.
Behavioral features like 'social_media_hours' and 'skills_development_hours'␣
 ↪add further context to engagement levels.
Categorical features such as 'gender', 'scholarship', and 'learning_mode' help␣
 ↪capture group differences, while support related variables like␣
 ↪'on_probation' and 'has_consulted_teacher'
reflect academic standing and intervention history.

This feature set offers a comprehensive foundation for building a robust and␣
 ↪interpretable predictive model.
```

```
"""
```

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

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

[59]:
```
print("Feature Selection Explanations:", feature_selection_explanations)
```

```
Feature Selection Explanations:
provide a quick explanation on the features selected

The final set of selected features includes a balanced mix of demographic,
academic, behavioral, and contextual variables.
This selection was informed by three approaches-correlation analysis, Random
Forest importance, and RFE to ensure both individual relevance and combined
predictive power,
while also considering ethical implications.

Key academic predictors such as 'previous_gpa', 'current_gpa', 'study_hours',
and 'average_attendance' were included due to their strong association with
performance.
Behavioral features like 'social_media_hours' and 'skills_development_hours' add
further context to engagement levels.
Categorical features such as 'gender', 'scholarship', and 'learning_mode' help
capture group differences, while support related variables like 'on_probation'
and 'has_consulted_teacher'
reflect academic standing and intervention history.

This feature set offers a comprehensive foundation for building a robust and
interpretable predictive model.
```

1.7 E. Data Cleaning

[ ]:
```
# Do not modify this code
try:
  df_clean = df[features_list].copy()
except Exception as e:
  print(e)
```

1.7.1 E.1 Fixing "Data Types"

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

[ ]:
```
age                          float64
gender                        object
state_abbr                    object
admission_year               float64
hsc_year                     float64
scholarship                   object
university_transport          object
learning_mode                 object
has_phone                     object
has_laptop                    object
english_proficiency           object
on_probation                  object
is_suspended                  object
has_consulted_teacher         object
relationship                  object
co_curricular                 object
living_arrangement            object
health_issues                 object
disabilities                  object
target                        object
current _semester            float64
study_hours                  float64
study_sessions               float64
social_media_hours           float64
average_attendance           float64
skills                        object
skills_development_hours     float64
area_of_interest              object
previous_gpa                 float64
current_gpa                  float64
completed_credits            float64
has_diploma                     bool
house_income                 float64
dtype: object
```

[ ]:
```
obj = []
for col in df_clean.columns:
  if df_clean[col].dtype == 'object' or df_clean[col].dtype == 'bool':
    obj.append(col)

df_clean[obj] = df_clean[obj].astype('string')
```

[60]:
```
# <Student to fill this section>
data_cleaning_1_explanations = """
Provide some explanations on why you believe it is important to fix this issue␣
↪and its impacts

Object-type columns were converted to strings to ensure consistency and avoid␣
↪errors during encoding or feature extraction.
This step is important for correct interpretation of categorical values (e.g.,␣
↪gender, skills), especially when applying label encoding.
All numerical values were left as float to preserve their quantitative meaning.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_cleaning_1_explanations',␣
↪value=data_cleaning_1_explanations)
```

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

[61]:
```
print("Data Cleaning 1 Explanations:", data_cleaning_1_explanations)
```

```
Data Cleaning 1 Explanations:
Provide some explanations on why you believe it is important to fix this issue
and its impacts

Object-type columns were converted to strings to ensure consistency and avoid
errors during encoding or feature extraction.
This step is important for correct interpretation of categorical values (e.g.,
gender, skills), especially when applying label encoding.
All numerical values were left as float to preserve their quantitative meaning.
```

1.7.2 E.2 Fixing "Errors"

[ ]:
```
# <Student to fill this section>
df_clean.columns = df_clean.columns.str.strip().str.replace(r'\s+', '',␣
↪regex=True)
```

[ ]:
```
print(df_clean['current_semester'].unique())

df_clean.loc[df_clean['current_semester'] == 2022.0, 'current_semester'] = (
    (2023 - df_clean['admission_year']) * 2
)
print("..........................................")
df_clean['current_semester'] = df_clean['current_semester'].astype(int)

print(df_clean['current_semester'].unique())
```

```
[4.000e+00 2.000e+00 8.000e+00 1.200e+01 1.100e+01 9.000e+00 1.300e+01
 5.000e+00 1.400e+01 1.000e+01 7.000e+00 2.200e+01 1.500e+01 1.800e+01
 2.022e+03 1.000e+00 1.700e+01 3.000e+00 6.000e+00 1.900e+01 2.400e+01]
…
[ 4  2  8 12 11  9 13  5 14 10  7 22 15 18  1 17  3  6 19 24]
```

[ ]:
```
print(df_clean['admission_year'].unique())

df_clean.loc[df_clean['admission_year'] == 22022.0, 'admission_year'] = 2022

print("...........................................")
print(df_clean['admission_year'].unique())
```

```
[ 2021.  2022.  2020.  2018.  2019.  2017.  2014.  2016.  2015.  2013.
 22022.  2023.]
…
[2021. 2022. 2020. 2018. 2019. 2017. 2014. 2016. 2015. 2013. 2023.]
```

[62]:
```
# <Student to fill this section>
data_cleaning_2_explanations = """
Provide some explanations on why you believe it is important to fix this issue␣
↪and its impacts

This step addressed three key data issues. First, an extra space in the column␣
↪name 'current _semester' was removed to ensure proper referencing.
Second, an incorrect value (2022.0) in the 'current_semester' column was␣
↪replaced using an estimated value: (2023 - admission_year) * 2,
assuming two semesters per year.
Additionally, a data entry error where 'admission_year' was recorded as 22022.0␣
↪was corrected to 2022.
This was likely caused by accidental double-clicking or holding the '2' key␣
↪during data entry.
Fixing these issues improves data consistency and accuracy, reducing the risk␣
↪of calculation or model errors during preprocessing.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_cleaning_2_explanations',␣
↪value=data_cleaning_2_explanations)
```

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

[63]:
```
print("Data Cleaning 2 Explanations:", data_cleaning_2_explanations)
```

```
Data Cleaning 2 Explanations:
Provide some explanations on why you believe it is important to fix this issue
and its impacts
```

```
This step addressed three key data issues. First, an extra space in the column
name 'current _semester' was removed to ensure proper referencing.
Second, an incorrect value (2022.0) in the 'current_semester' column was
replaced using an estimated value: (2023 - admission_year) * 2,
assuming two semesters per year.
Additionally, a data entry error where 'admission_year' was recorded as 22022.0
was corrected to 2022.
This was likely caused by accidental double-clicking or holding the '2' key
during data entry.
Fixing these issues improves data consistency and accuracy, reducing the risk of
calculation or model errors during preprocessing.
```

1.7.3 E.3 Fixing "Missing Values and Duplicates"

[ ]:
```python
# <Student to fill this section>
df_clean.duplicated().sum()
```

[ ]:
```
np.int64(0)
```

[ ]:
```python
df_clean.isna().sum()[df_clean.isna().sum() > 0]
```

[ ]:
```
skills              1
area_of_interest    7
dtype: int64
```

[ ]:
```python
# Fill missing 'skills' with its mode
df_clean['skills'].fillna(df_clean['skills'].mode()[0], inplace=True)

# Fill missing 'area_of_interest' with its mode
df_clean['area_of_interest'].fillna(df_clean['area_of_interest'].mode()[0],␣
 ↪inplace=True)
```

[ ]:
```python
df_clean.isna().sum()[df_clean.isna().sum() > 0]
```

[ ]:
```
Series([], dtype: int64)
```

[64]:
```python
# <Student to fill this section>
data_cleaning_3_explanations = """
Provide some explanations on why you believe it is important to fix this issue␣
 ↪and its impacts

No duplicate records were detected. Handling missing values is essential to␣
 ↪ensure data completeness and avoid issues during model training.
In this step, missing values in 'skills' and 'area_of_interest' were replaced␣
 ↪with their respective modes,
which represent the most frequent values in these categorical variables.
```

```
This approach preserves all rows, prevents data loss, and maintains consistency␣
 ↪for encoding and analysis.
Ignoring these missing values could lead to errors or introduce bias during␣
 ↪feature transformation and modeling.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='data_cleaning_3_explanations',␣
 ↪value=data_cleaning_3_explanations)
```

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

[65]:
```
print("Data Cleaning 3 Explanations:", data_cleaning_3_explanations)
```

```
Data Cleaning 3 Explanations:
Provide some explanations on why you believe it is important to fix this issue
and its impacts

No duplicate records were detected. Handling missing values is essential to
ensure data completeness and avoid issues during model training.
In this step, missing values in 'skills' and 'area_of_interest' were replaced
with their respective modes,
which represent the most frequent values in these categorical variables.

This approach preserves all rows, prevents data loss, and maintains consistency
for encoding and analysis.
Ignoring these missing values could lead to errors or introduce bias during
feature transformation and modeling.
```

1.8 F. Feature Engineering

[ ]:
```
# Do not modify this code
try:
  df_eng = df_clean.copy()
except Exception as e:
  print(e)
```

1.8.1 F.1 New Feature "gpa_change"

[ ]:
```
# <Student to fill this section>
df_eng['gpa_change'] = df_eng['current_gpa'] + df_eng['previous_gpa']
```

[66]:
```
# <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 'gpa_change' feature was created to capture academic progress or decline␣
↪over time.
This helps identify students whose performance is improving or deteriorating,␣
↪providing a dynamic view beyond static GPA scores.
It adds value by highlighting recent trends that may signal risk, which is␣
↪critical for early intervention strategies.
"""
```

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

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

[67]:
```
print("Feature Engineering 1 Explanations:", feature_engineering_1_explanations)
```

```
Feature Engineering 1 Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The 'gpa_change' feature was created to capture academic progress or decline
over time.
This helps identify students whose performance is improving or deteriorating,
providing a dynamic view beyond static GPA scores.
It adds value by highlighting recent trends that may signal risk, which is
critical for early intervention strategies.
```

1.8.2 F.2 New Feature "academic_gap_years"

[ ]:
```
# <Student to fill this section>
df_eng['academic_gap_years'] = df_eng['admission_year'] - df_eng['hsc_year']
df_eng.drop(columns=['hsc_year'], inplace=True)
```

[68]:
```
# <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 'academic_gap_years' feature was created to measure the time gap between␣
↪completing high school and starting university.
```

```
This gap may reflect factors such as gap years, delays, or non-traditional␣
  ↪education paths, which could influence student performance.
After creating this feature, 'hsc_year' was dropped as its information was␣
  ↪fully captured within the new variable.
"""
```

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

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

[69]:
```
print("Feature Engineering 2 Explanations:", feature_engineering_2_explanations)
```

```
Feature Engineering 2 Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The 'academic_gap_years' feature was created to measure the time gap between
completing high school and starting university.
This gap may reflect factors such as gap years, delays, or non-traditional
education paths, which could influence student performance.
After creating this feature, 'hsc_year' was dropped as its information was fully
captured within the new variable.
```

1.8.3 F.3 New Feature "study_efficiency"

[ ]:
```
# <Student to fill this section>
df_eng['study_efficiency'] = df_eng['study_hours'] / (df_eng['study_sessions']␣
  ↪+ 1e-5)
```

[70]:
```
# <Student to fill this section>
feature_engineering_3_explanations = """
Provide some explanations on why you believe it is important to create this␣
  ↪feature and its impacts

The 'study_efficiency' feature was created to estimate how much time a student␣
  ↪spends per study session.
This provides insight into study habits and focus, which may influence academic␣
  ↪outcomes more meaningfully than raw study time or session count alone.
It captures qualitative aspects of study behavior and helps differentiate␣
  ↪between short, frequent sessions and longer, focused ones.
"""
```

[ ]:
```
# Do not modify this code
```

```
print_tile(size="h3", key='feature_engineering_3_explanations',␣
 ↪value=feature_engineering_3_explanations)
```

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

[71]:
```
print("Feature Engineering 3 Explanations:", feature_engineering_3_explanations)
```

```
Feature Engineering 3 Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The 'study_efficiency' feature was created to estimate how much time a student
spends per study session.
This provides insight into study habits and focus, which may influence academic
outcomes more meaningfully than raw study time or session count alone.
It captures qualitative aspects of study behavior and helps differentiate
between short, frequent sessions and longer, focused ones.
```

1.8.4 F.4 Fixing "resource_access_score"

[ ]:
```
df_eng['has_phone'] = df_eng['has_phone'].map({'Yes': 1, 'No': 0})
df_eng['has_laptop'] = df_eng['has_laptop'].map({'Yes': 1, 'No': 0})
df_eng['resource_access_score'] = df_eng['has_phone'] + df_eng['has_laptop']
df_eng.drop(columns=['has_phone', 'has_laptop'], inplace=True)
```

[ ]:
```
# Save df_eng for more feature engineering in future experiments if needed.
try:
  df_eng.to_csv(at.folder_path / 'df_eng.csv', index=False)
except Exception as e:
  print(e)
```

[72]:
```
# <Student to fill this section>
feature_engineering_4_explanations = """
Provide some explanations on why you believe it is important to create this␣
 ↪feature and its impacts

The 'resource_access_score' feature was created by combining binary indicators␣
 ↪for 'has_phone' and 'has_laptop' into a single score reflecting students'␣
 ↪access to essential digital learning tools.
This captures the level of technological readiness, which can impact␣
 ↪participation, productivity, and overall academic success.
After generating this feature, the original columns were dropped to simplify␣
 ↪the dataset and avoid redundancy.
"""
```

[ ]:
```
# Do not modify this code
print_tile(size="h3", key='feature_engineering_4_explanations',␣
↪value=feature_engineering_4_explanations)
```

<IPython.core.display.HTML object>

[73]:
```
print("Feature Engineering 4 Explanations:", feature_engineering_4_explanations)
```

```
Feature Engineering 4 Explanations:
Provide some explanations on why you believe it is important to create this
feature and its impacts

The 'resource_access_score' feature was created by combining binary indicators
for 'has_phone' and 'has_laptop' into a single score reflecting students' access
to essential digital learning tools.
This captures the level of technological readiness, which can impact
participation, productivity, and overall academic success.
After generating this feature, the original columns were dropped to simplify the
dataset and avoid redundancy.
```

1.9 G. Data Preparation for Modeling

1.9.1 G.1 Split Datasets

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

df_sorted = df_eng.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
```

[ ]:
```
train_df['admission_year'].value_counts()
```

[ ]:
```
admission_year
2020.0    254
2021.0    241
2019.0    156
2018.0     38
2017.0      9
2016.0      3
2014.0      3
2013.0      1
2015.0      1
Name: count, dtype: int64
```

[ ]:
```python
val_df['admission_year'].value_counts()
```

[ ]:
```
admission_year
2022.0    154
2021.0     48
Name: count, dtype: int64
```

[ ]:
```python
test_df['admission_year'].value_counts()
```

[ ]:
```
admission_year
2022.0    95
2023.0     6
Name: count, dtype: int64
```

[ ]:
```python
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']
```

[74]:
```python
# <Student to fill this section>
data_splitting_explanations = """
Provide some explanations on what is the best strategy to use for data␣
↪splitting for this dataset

Since students are admitted at different times, the best strategy for splitting␣
↪this dataset is based on admission year.
By sorting the data chronologically, we simulate a real-world scenario where␣
↪the model is trained on past data and evaluated on more recent student␣
↪records.
```

```
This approach prevents data leakage and helps assess the model's ability to␣
↪generalize to future cohorts. The dataset was split into 70% for training␣
↪(older admissions),
20% for validation (more recent), and 10% for testing (latest admissions),␣
↪ensuring that model evaluation is based on fresh and unseen data.
"""
```

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

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

[75]:
```
print("Data Splitting Explanations:", data_splitting_explanations)
```

```
Data Splitting Explanations:
Provide some explanations on what is the best strategy to use for data splitting
for this dataset

Since students are admitted at different times, the best strategy for splitting
this dataset is based on admission year.
By sorting the data chronologically, we simulate a real-world scenario where the
model is trained on past data and evaluated on more recent student records.

This approach prevents data leakage and helps assess the model's ability to
generalize to future cohorts. The dataset was split into 70% for training (older
admissions),
20% for validation (more recent), and 10% for testing (latest admissions),
ensuring that model evaluation is based on fresh and unseen data.
```

1.9.2 G.2 Data Transformation

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

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

X_train["has_diploma"] = X_train["has_diploma"].map({'True': 1, 'False': 0})
X_val["has_diploma"] = X_val["has_diploma"].map({'True': 1, 'False': 0})
X_test["has_diploma"] = X_test["has_diploma"].map({'True': 1, 'False': 0})
```

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

y_train = y_train.map(target_mapping)
y_val = y_val.map(target_mapping)
y_test = y_test.map(target_mapping)
```

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

X_train['english_proficiency'] = X_train['english_proficiency'].map(eng_mapping)
X_val['english_proficiency'] = X_val['english_proficiency'].map(eng_mapping)
X_test['english_proficiency'] = X_test['english_proficiency'].map(eng_mapping)
```

[76]:
```python
# <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.
"""
```

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

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

[77]:
```python
print("Data Transformation 1 Explanations:", data_transformation_1_explanations)
```

```
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.
```

1.9.3 G.3 Data Transformation

[ ]:
```python
# <Student to fill this section>
string_cols = X_train.select_dtypes(include=['object', 'string']).columns.
 ↪tolist()

X_combined = pd.concat([X_train, X_val, X_test], axis=0)

X_combined_encoded = pd.get_dummies(X_combined, columns=string_cols,␣
 ↪drop_first=True)

# Split back into original sets
X_train = X_combined_encoded.iloc[:len(X_train)]
X_val = X_combined_encoded.iloc[len(X_train):len(X_train)+len(X_val)]
X_test = X_combined_encoded.iloc[len(X_train)+len(X_val):]
```

[78]:
```python
# <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

This transformation applied one-hot encoding to categorical string columns␣
 ↪using get_dummies to convert them into numerical format and
avoid implying any order, ensuring accurate representation in the model.

To ensure consistency across the train, validation, and test sets, the data was␣
 ↪combined before encoding and then split back,
so that all sets contain the same features with aligned dummy columns.

This step helps preserve the full representation of categorical variables while␣
 ↪avoiding issues such as mismatched column dimensions
or missing categories in the validation and test sets.
```

```
"""
```

[79]:
```
# Do not modify this code
print_tile(size="h3", key='data_transformation_2_explanations',␣
↪value=data_transformation_2_explanations)
```

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

[80]:
```
print("Data Transformation 2 Explanations:", data_transformation_2_explanations)
```

```
Data Transformation 2 Explanations:
Provide some explanations on why you believe it is important to perform this
data transformation and its impacts

This transformation applied one-hot encoding to categorical string columns using
get_dummies to convert them into numerical format and
avoid implying any order, ensuring accurate representation in the model.

To ensure consistency across the train, validation, and test sets, the data was
combined before encoding and then split back,
so that all sets contain the same features with aligned dummy columns.

This step helps preserve the full representation of categorical variables while
avoiding issues such as mismatched column dimensions
or missing categories in the validation and test sets.
```

1.9.4 G.4 Data Transformation

[ ]:
```
# <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)
```

[81]:
```
# <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>

[82]:
```
print("Data Transformation 3 Explanations:", data_transformation_3_explanations)
```

```
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.
```

1.10 H. Save Datasets

[ ]:
```
# Do not modify this code
try:
  X_train.to_csv(at.folder_path / 'X_train.csv', index=False)
  y_train.to_csv(at.folder_path / 'y_train.csv', index=False)

  X_val.to_csv(at.folder_path / 'X_val.csv', index=False)
  y_val.to_csv(at.folder_path / 'y_val.csv', index=False)

  X_test.to_csv(at.folder_path / 'X_test.csv', index=False)
  y_test.to_csv(at.folder_path / 'y_test.csv', index=False)
except Exception as e:
  print(e)
```

1.11 I. Assess Baseline Model

1.11.1 I.1 Generate Predictions with Baseline Model

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

dummy_clf = DummyClassifier(strategy='stratified', random_state=42)
dummy_clf.fit(X_train, y_train)

y_train_preds = dummy_clf.predict(X_train)
```

1.11.2 I.2 Selection of Performance Metrics

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

print("Dummy Classifier (Baseline) Evaluation:\n")
print("\nRecall_score:\n", recall_score(y_train, y_train_preds,␣
↪average='weighted'))
print("\nf1_score:\n", f1_score(y_train, y_train_preds, average='weighted'))
print("\nClassification Report:\n", classification_report(y_train,␣
↪y_train_preds))
print("\nConfusion Matrix:\n", confusion_matrix(y_train, y_train_preds))
```

```
Dummy Classifier (Baseline) Evaluation:


Recall_score:
 0.3668555240793201

f1_score:
 0.36320588809659704

Classification Report:
               precision    recall  f1-score   support

           0       0.49      0.52      0.51       352
           1       0.29      0.29      0.29       203
           2       0.17      0.14      0.16       118
           3       0.03      0.03      0.03        33

    accuracy                           0.37       706
   macro avg       0.25      0.24      0.25       706
weighted avg       0.36      0.37      0.36       706
```

```
Confusion Matrix:
 [[183  96  53  20]
 [108  58  26  11]
 [ 61  35  17   5]
 [ 18   9   5   1]]
```

[83]:
```
# <Student to fill this section>
performance_metrics_explanations = """
Provide some explanations on why you believe the performance metrics you chose␣
↪is appropriate

Given the multiclass and imbalanced nature of the target variable, using only␣
↪accuracy would be misleading, as it favors the majority class.
Therefore, recall, and F1-score (especially the weighted average) were used to␣
↪provide a more balanced evaluation across all classes.

The weighted F1-score is particularly appropriate because it accounts for class␣
↪imbalance, giving a clearer picture of overall model performance.
The classification report provides recall scores for each category, making it␣
↪easier to evaluate how well the model identifies instances within each class.
The confusion matrix displays the number of correct and incorrect predictions␣
↪for each category,
making it easier to assess how well the model distinguishes between classes.

These metrics highlight the DummyClassifier's limited predictive ability and␣
↪serve as a fair baseline for comparison with more advanced models.
"""
```

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

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

[84]:
```
print("Performance Metrics Explanations:", performance_metrics_explanations)
```

```
Performance Metrics Explanations:
Provide some explanations on why you believe the performance metrics you chose
is appropriate

Given the multiclass and imbalanced nature of the target variable, using only
accuracy would be misleading, as it favors the majority class.
Therefore, recall, and F1-score (especially the weighted average) were used to
provide a more balanced evaluation across all classes.

The weighted F1-score is particularly appropriate because it accounts for class
imbalance, giving a clearer picture of overall model performance.
```

```
The classification report provides recall scores for each category, making it
easier to evaluate how well the model identifies instances within each class.
The confusion matrix displays the number of correct and incorrect predictions
for each category,
making it easier to assess how well the model distinguishes between classes.

These metrics highlight the DummyClassifier's limited predictive ability and
serve as a fair baseline for comparison with more advanced models.
```

1.11.3 I.3 Baseline Model Performance

[ ]:
```
# <Student to fill this section>
(y_train_preds - y_train).sum()
```

[ ]:
```
np.int64(-27)
```

[85]:
```
# <Student to fill this section>
baseline_performance_explanations = """
Provide some explanations on model performance

The DummyClassifier serves as a baseline model, predicting labels based only on␣
↪class distribution without learning any patterns.
Its weighted F1-score of 0.36 and accuracy of 0.37 reflect poor performance,␣
↪especially on minority classes like 'Excellent' and 'Good', where recall is␣
↪close to zero.

The total difference between predicted and actual target values in the training␣
↪set was 27, indicating a measurable but uncontrolled deviation in ordinal␣
↪predictions.
This reinforces the fact that the DummyClassifier cannot meaningfully capture␣
↪academic performance levels and should only be used as a benchmark for␣
↪comparison.
"""
```

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

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

[86]:
```
print("Baseline Performance Explanations:", baseline_performance_explanations)
```

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

The DummyClassifier serves as a baseline model, predicting labels based only on
class distribution without learning any patterns.
```

Its weighted F1-score of 0.36 and accuracy of 0.37 reflect poor performance, especially on minority classes like 'Excellent' and 'Good', where recall is close to zero.

The total difference between predicted and actual target values in the training set was 27, indicating a measurable but uncontrolled deviation in ordinal predictions.
This reinforces the fact that the DummyClassifier cannot meaningfully capture academic performance levels and should only be used as a benchmark for comparison.

[30]:
```
# Clear metadata for all experiment notebooks
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-0.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-1.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-2.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-3.ipynb" \
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-4.ipynb" \
```

```
--ClearMetadataPreprocessor.enabled=True \
--ClearMetadataPreprocessor.clear_cell_metadata=True \
--ClearMetadataPreprocessor.clear_notebook_metadata=True \
--ClearOutputPreprocessor.enabled=False \
--inplace

# Convert all notebooks to PDF
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-0.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-1.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-2.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-3.ipynb" --to pdf
!jupyter nbconvert "/content/gdrive/MyDrive/Colab Notebooks/
↪36106-25AU-AT2-25589351-experiment-4.ipynb" --to pdf
```

```
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-0.ipynb to notebook
[NbConvertApp] Writing 933584 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-0.ipynb
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-1.ipynb to notebook
[NbConvertApp] Writing 87327 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-1.ipynb
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-2.ipynb to notebook
[NbConvertApp] Writing 89857 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-2.ipynb
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-3.ipynb to notebook
[NbConvertApp] Writing 96347 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-3.ipynb
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-4.ipynb to notebook
[NbConvertApp] Writing 162317 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-4.ipynb
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-0.ipynb to pdf
[NbConvertApp] Support files will be in 36106-25AU-
AT2-25589351-experiment-0_files/
[NbConvertApp] Making directory ./36106-25AU-AT2-25589351-experiment-0_files
[NbConvertApp] Writing 216377 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
```

```
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 661421 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-0.pdf
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-1.ipynb to pdf
[NbConvertApp] Support files will be in 36106-25AU-
AT2-25589351-experiment-1_files/
[NbConvertApp] Making directory ./36106-25AU-AT2-25589351-experiment-1_files
[NbConvertApp] Writing 69575 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 93906 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-1.pdf
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-2.ipynb to pdf
[NbConvertApp] Support files will be in 36106-25AU-
AT2-25589351-experiment-2_files/
[NbConvertApp] Making directory ./36106-25AU-AT2-25589351-experiment-2_files
[NbConvertApp] Writing 70225 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 93187 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-2.pdf
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-3.ipynb to pdf
[NbConvertApp] Support files will be in 36106-25AU-
AT2-25589351-experiment-3_files/
[NbConvertApp] Making directory ./36106-25AU-AT2-25589351-experiment-3_files
[NbConvertApp] Writing 75563 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 97357 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-3.pdf
[NbConvertApp] Converting notebook /content/gdrive/MyDrive/Colab
```

```
Notebooks/36106-25AU-AT2-25589351-experiment-4.ipynb to pdf
[NbConvertApp] Support files will be in 36106-25AU-
AT2-25589351-experiment-4_files/
[NbConvertApp] Making directory ./36106-25AU-AT2-25589351-experiment-4_files
[NbConvertApp] Writing 102495 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 148585 bytes to /content/gdrive/MyDrive/Colab
Notebooks/36106-25AU-AT2-25589351-experiment-4.pdf
```

[ ]:

```

```