Experiment Notebook

---

0. Setup Environment

0.a Install Environment and Mandatory Packages

```
# 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()
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 17.7 MB/s eta 0:00:00
Mounted at /content/gdrive

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

0.b Disable Warnings Messages

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

0.c Install Additional Packages

0.d Import Packages

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

```
# <Student to fill this section>
```

```
# <Student to fill this section>
import pandas as pd
import altair as alt
```

A. Project Description

```
# <Student to fill this section>
student_name = "Md Saifur Rahman"
student_id = "25528668"
```

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

```
student_name
Md Saifur Rahman
```

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

student_id

25528668

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
# <Student to fill this section>
business_objective = """
I am working on a project to create a tool that will assess the current academic
progress and will predict the future academic performannce. The goal is to spot
students who might struggle or those likely to shine, so the university can step
in with tailored support, use resources wisely, and help students succeed.
This will help advisors and university leaders make smart, data-backed decisions
to boost retention, grades, and student happiness.
"""


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

business_objective

I am working on a project to create a tool that will assess the current academic progress and will predict the future academic performannce. The goal is to spot students who might struggle or those likely to shine, so the university can step in with tailored support, use resources wisely, and help students succeed. This will help advisors and university leaders make smart, data-backed decisions to boost retention.

B. Experiment Description

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

experiment_id

0

```
experiment_hypothesis = """
We think things like past grades, study habits, attendance, and skill growth are
big clues to how well a student will do this semester. Our hunch is that students
with solid GPAs, steady study routines, and good attendance will likely excel.
In this experiment, we’re digging into the data to test these ideas and build a
foundation for creating a predictive model later on.
"""


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

experiment_hypothesis

We think things like past grades, study habits, attendance, and skill growth are big clues to how well a student will do this semester. Our hunch is that students with solid GPAs, steady study routines, and good attendance will likely excel. In this experiment, we’re digging into the data to test these ideas and build a foundation for creating a predictive model later on.

```
# <Student to fill this section>
experiment_expectations = """
In this first experiment, we’re diving into the dataset to get a clear picture
of its structure, quality, and the key factors that affect how students perform.
We expect to pinpoint important things like past grades, study habits, and
engagement that will help predict success. We also think we’ll spot some data
issues—like missing info, weird outliers, or irrelevant stuff—that we’ll need to
fix to make our model reliable later. This groundwork will help us pick the right
modeling methods and evaluation plans for the next steps.
"""
```

```
# Do not modify this code
print_tile(size="h3", key='experiment_expectations', value=experiment_expectations)
```

experiment_expectations

In this first experiment, we're diving into the dataset to get a clear picture of its structure, quality, and the key factors that affect how students perform. We expect to pinpoint important things like past grades, study habits, and engagement that will help predict success. We also think we'll spot some data issues—like missing info, weird outliers, or irrelevant stuff—that we'll need to fix to make our model reliable later. This groundwork will help us pick the right modeling methods and evaluation plans for the next steps.

C. Data Understanding

```
# Do not modify this code
try:
  df = pd.read_csv("/content/drive/MyDrive/36106/AT02/students_performance.csv")
except Exception as e:
  print(e)
```

```
df.head()
```

	student_id	full_name	age	email	phone_number	gender	birth_country	secondary_address	building_number	street_n
0	7	Lauren Moon	22.0	kimberlypark@example.org	(08)35431944	Female	AU	Unit 93	0	April An
1	11	Larry Green	22.0	smithamy@example.net	(07)35774291	Male	AU	Level 2	43	Davis C
2	15	Alexander Scott	20.0	qlee@example.org	20356212	Male	NZ	937/	96	Emily L
3	18	Jonathan Thornton	21.0	cmorgan@example.com	0627-6253	Male	AU	Level 3	5	Will Path
4	20	Susan Smith	21.0	xtodd@example.com	9957-3583	Female	AU	00/	5	Martin Cl

5 rows × 45 columns

C.1 Explore Dataset

```
df.columns.tolist()
```

```
['student_id',
 'full_name',
 'age',
 'email',
 'phone_number',
 'gender',
 'birth_country',
 'secondary_address',
 'building_number',
 'street_name',
 'street_suffix',
 'city',
 'postcode',
 'state_abbr',
 'admission_year',
 'hsc_year',
 'program',
 '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_hours',
          'area_of_interest',
          'previous_gpa',
          'current_gpa',
          'completed_credits',
          'has_diploma',
          'house_income']


    # prompt: assign all the column name into feature_list

    feature_list = df.columns.tolist()



    df.info()
    df.dtypes.value_counts()
```

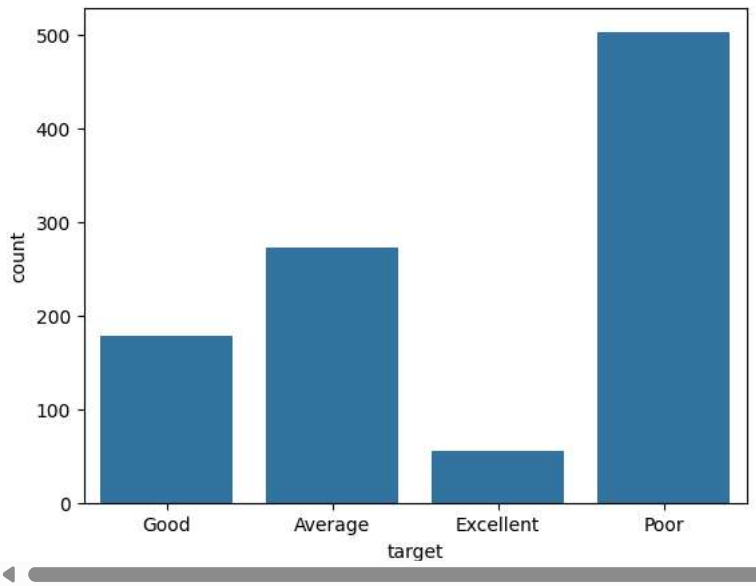
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 45 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   student_id                1009 non-null   int64
 1   full_name                 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
```

	count
object	28
float64	13

```
import seaborn as sns # should be moved at the top
df['target'].value_counts()
sns.countplot(data=df, x='target')
```

```
dtype: int64
```

```
<Axes: xlabel='target', ylabel='count'>
```




```
df.describe().T
```

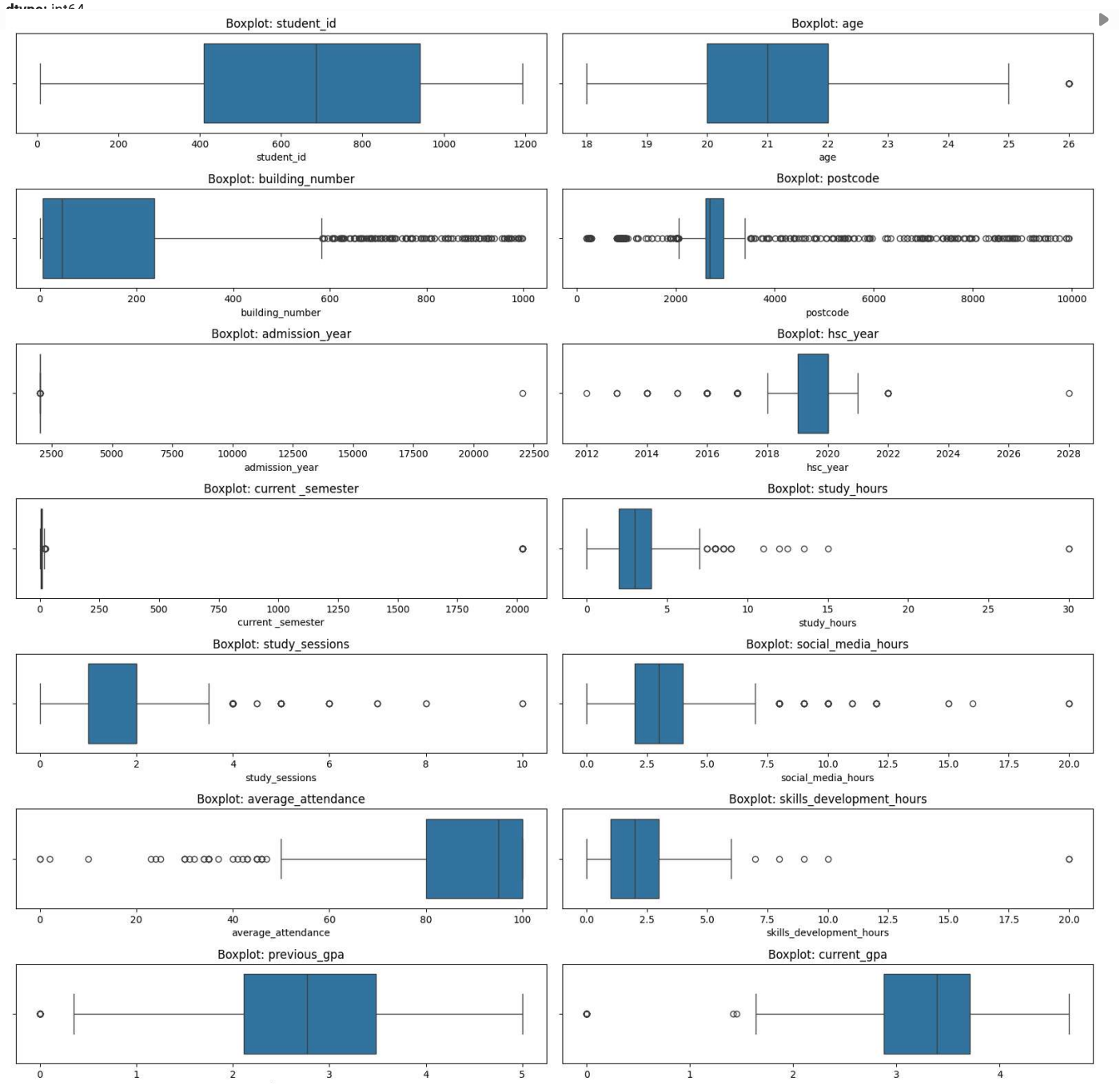
	count	mean	std	min	25%	50%	75%	max
student_id	1009.0	673.108028	311.377223	7.0	410.00	685.00	941.00	1193.00
age	1009.0	21.368285	1.614943	18.0	20.00	21.00	22.00	26.00
building_number	1009.0	180.305253	273.531962	0.0	6.00	46.00	237.00	998.00
postcode	1009.0	3153.528246	1768.243964	202.0	2602.00	2691.00	2960.00	9941.00
admission_year	1009.0	2040.321110	629.677177	2013.0	2020.00	2021.00	2022.00	22022.00
hsc_year	1009.0	2019.251734	1.346681	2012.0	2019.00	2020.00	2020.00	2028.00
current _semester	1009.0	43.000991	266.874155	1.0	3.00	8.00	10.00	2022.00
study_hours	1009.0	3.334616	2.096762	0.0	2.00	3.00	4.00	30.00
study_sessions	1009.0	2.066898	1.034492	0.0	1.00	2.00	2.00	10.00
social_media_hours	1009.0	3.439296	2.439363	0.0	2.00	3.00	4.00	20.00
average_attendance	1009.0	88.111001	16.079094	0.0	80.00	95.00	100.00	100.00
skills_development_hours	1009.0	2.224975	1.473957	0.0	1.00	2.00	3.00	20.00
previous_gpa	1009.0	2.756482	0.858012	0.0	2.11	2.77	3.48	5.00
current_gpa	1009.0	3.211343	0.731698	0.0	2.88	3.39	3.71	4.67
completed_credits	1009.0	76.936571	47.733885	0.0	24.00	85.00	122.00	147.00
house_income	1009.0	63495.758176	79276.584819	2530.0	30000.00	50000.00	77000.00	2000000.00

```
df.isnull().sum().sort_values(ascending=False)
```

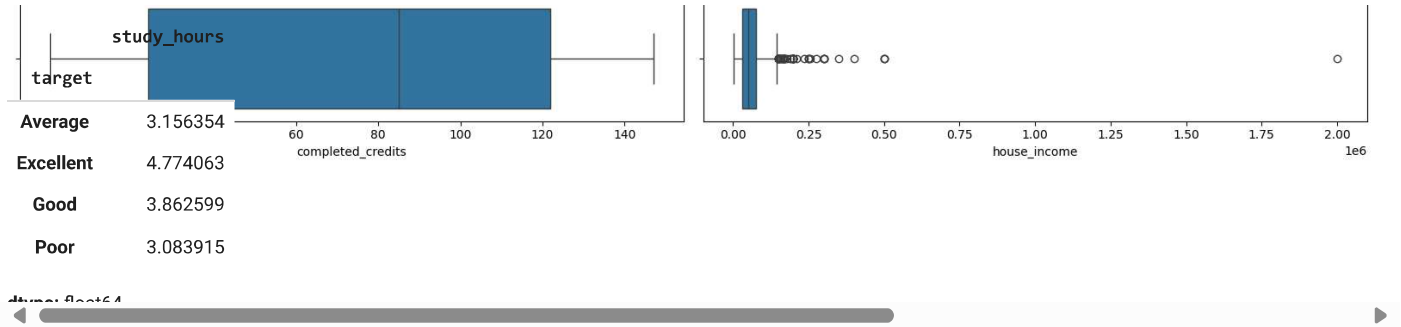
	0
area_of_interest	7
skills	1
age	0
full_name	0
student_id	0
gender	0
birth_country	0
secondary_address	0
building_number	0
street_name	0
street_suffix	0
email	0
phone_number	0
postcode	0
city	0
state_abbr	0
admission_year	0
scholarship	0
university_transport	0
hsc_year	0
program	0
has_laptop	0
english_proficiency	0
on_probation	0
is_suspended	0
has_consulted_teacher	0
relationship	0
learning_mode	0
has_phone	0
living_arrangement	0
co_curricular	0

```
import matplotlib.pyplot as plt
import seaborn as sns
# Select only numerical columns
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Plot boxplots for all numerical columns
plt.figure(figsize=(16, 20))
for i, col in enumerate(numerical_cols):
    plt.subplot(len(numerical_cols) // 2 + 1, 2, i + 1)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot: {col}')
    plt.tight_layout()

plt.show()
```

previous_gpa	0
current_gpa	0
completed_credits	0
has_diploma	0
house_income	0




```
df.groupby('target')['study_hours'].mean()
```



target	study_hours
Average	3.156354
Excellent	4.774063
Good	3.862599
Poor	3.083915


```
# <Student to fill this section>
dataset_insights = """
The dataset consists of 1,009 records and 34 features that describe various
academic, demographic, and behavioral aspects of university students.
It includes categorical features such as gender, program, and relationship, and
numerical features like study_hours, social_media_hours, previous_gpa, and
average_attendance. Exploratory analysis revealed a moderately imbalanced target
variable, with most students falling into the average performance category.
```

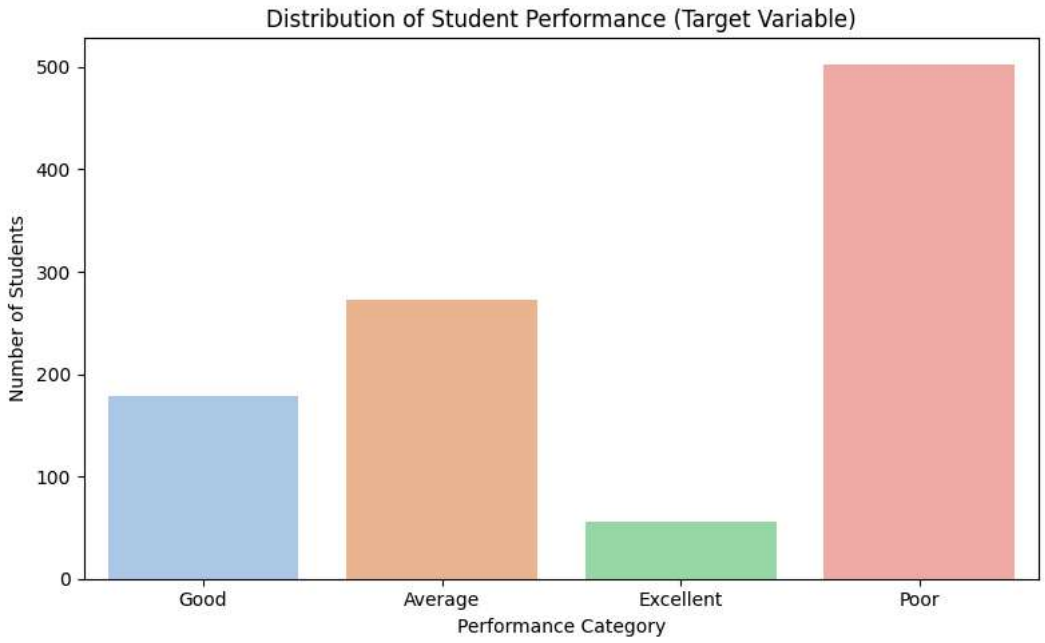
```
Several numerical features exhibited visible outliers, including study_sessions
and social_media_hours, which were addressed later using the IQR method.
A correlation heatmap highlighted strong relationships between previous_gpa and
current_gpa, supporting their use as key predictors. The feature area_of_interest
was removed due to a high proportion of missing values (around 69%).
Overall, the dataset provided a rich foundation for identifying key performance
indicators and preparing for predictive modeling.
"""


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

dataset_insights

The dataset consists of 1,009 records and 34 features that describe various academic, demographic, and behavioral aspects of university students. It includes categorical features such as gender, program, and relationship, and numerical features like study_hours, social_media_hours, previous_gpa, and average_attendance. Exploratory analysis revealed a moderately imbalanced target variable, with most students falling into the average performance category. Several numerical features exhibited visible outliers, including study_sessions and social_media_hours, which were addressed later using the IQR method. A correlation heatmap highlighted strong relationships between previous_gpa and current_gpa, supporting their use as key predictors. The feature area_of_interest was removed due to a high proportion of

C.2 Explore Target Variable

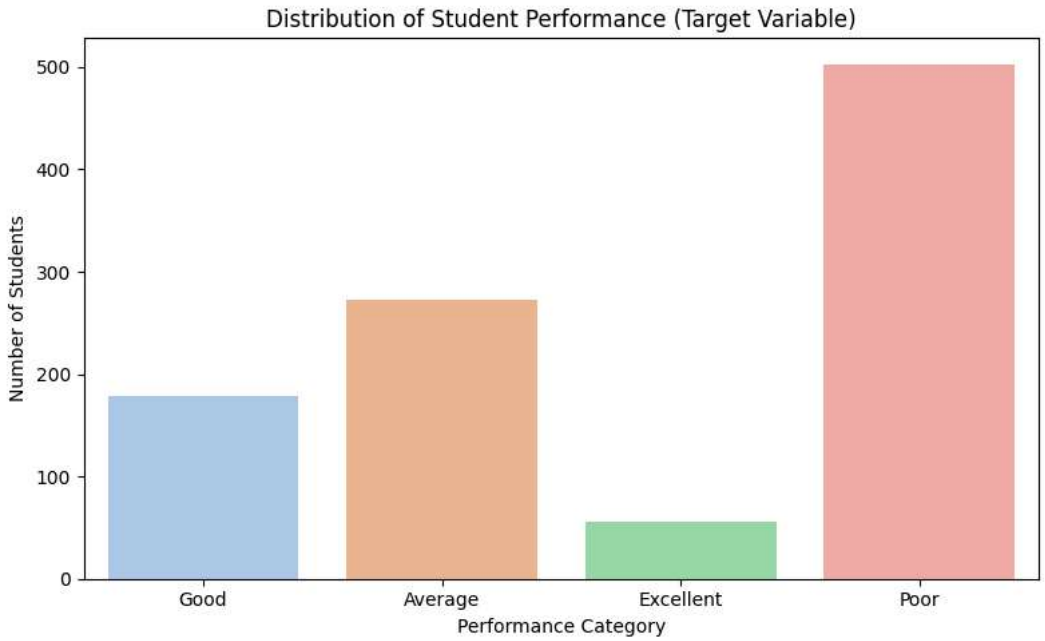
```
# <Student to fill this section>
target_name = 'target'


# Plot distribution of the target variable
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='target', palette='pastel')
plt.title("Distribution of Student Performance (Target Variable)")
plt.xlabel("Performance Category")
plt.ylabel("Number of Students")
plt.tight_layout()
plt.show()

# Display percentage distribution
target_dist = df['target'].value_counts(normalize=True) * 100
target_dist.round(2)
```



target	proportion
Poor	49.85
Average	26.96
Good	17.64
Excellent	5.55

dtype: float64

```
# Plot distribution of the target variable
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='target', palette='pastel')
plt.title("Distribution of Student Performance (Target Variable)")
plt.xlabel("Performance Category")
plt.ylabel("Number of Students")
plt.tight_layout()
plt.show()

# Percentage distribution of each class
target_dist = df['target'].value_counts(normalize=True) * 100
target_dist.round(2)
```



target	proportion
Poor	49.85
Average	26.96
Good	17.64
Excellent	5.55

dtype: float64

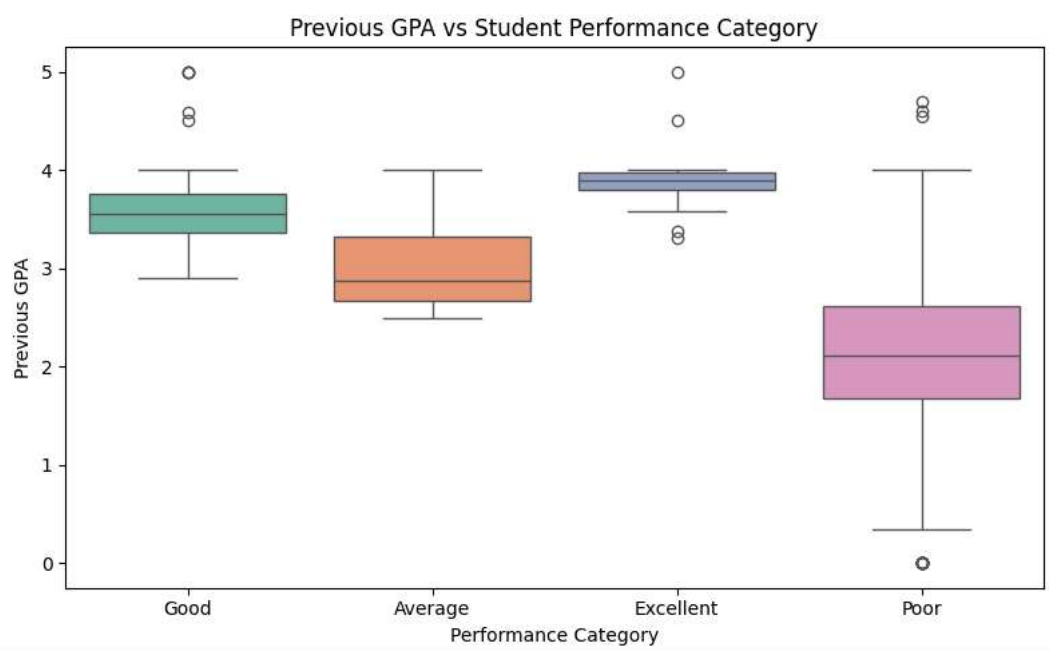
```
# <Student to fill this section>
target_insights = """
The target variable in this dataset reflects students' academic performance
categories and is composed of four distinct classes: 'Poor', 'Average', 'Good',
and 'Excellent'. The distribution is notably imbalanced — nearly half of the
students (49.85%) fall into the 'Poor' performance category, followed by
'Average' (26.96%) and 'Good' (17.64%). Only a small fraction (5.55%) belong to
the 'Excellent' group.

This imbalance may bias machine learning models toward the majority class
('Poor') unless mitigated through stratified sampling, resampling techniques
(like SMOTE), or use of metrics such as F1-score and balanced accuracy.
The limited representation of 'Excellent' students may also hinder the model's
ability to accurately detect high performers. This must be considered in both
model training and evaluation.

Despite the imbalance, the clear distinction between classes makes the target a
suitable categorical variable for a multi-class classification problem. It will
allow us to explore patterns in academic behavior and demographic factors that
correlate with different levels of student success.
"""


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

target_insights

The target variable in this dataset reflects students' academic performance categories and is composed of four distinct classes: 'Poor', 'Average', 'Good', and 'Excellent'. The distribution is notably imbalanced — nearly half of the students (49.85%) fall into the 'Poor' performance category, followed by 'Average' (26.96%) and 'Good' (17.64%). Only a small fraction (5.55%) belong to the 'Excellent' group. This imbalance may bias machine learning models toward the majority class ('Poor') unless mitigated through stratified sampling, resampling techniques (like SMOTE), or use of metrics such as F1-score and balanced accuracy. The limited representation of 'Excellent' students may also hinder the model's ability to accurately detect high performers. This must be considered in both model training and evaluation. Despite the imbalance, the clear distinction between classes makes the target a suitable categorical variable for a multi-class classification problem. It will allow us

C.3 Explore Feature of Interest `previous_gpa`

```
# <Student to fill this section>


plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='previous_gpa', palette='Set2')
plt.title("Previous GPA vs Student Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Previous GPA")
plt.tight_layout()
plt.show()
```




```
feature_3_insight = """
The feature 'previous_gpa' was chosen as a key feature of interest due to its
high importance across multiple feature selection methods. The boxplot comparing
'previous_gpa' across performance categories shows a clear positive relationship:
students with higher previous GPAs tend to fall into higher performance categories.

Students in the 'Poor' and 'Average' groups generally have lower previous GPAs,
while the 'Excellent' group shows consistently higher values with a tighter
distribution. This supports the hypothesis that academic history is a strong
indicator of future performance.

This feature is also interpretable and actionable — identifying students with
historically low GPAs can help target early interventions and academic support.
"""


# Do not modify this code
print_tile(size="h3", key='feature_3_insight', value=feature_3_insight) # Changed feature_3_insights to feature_3_insight
```

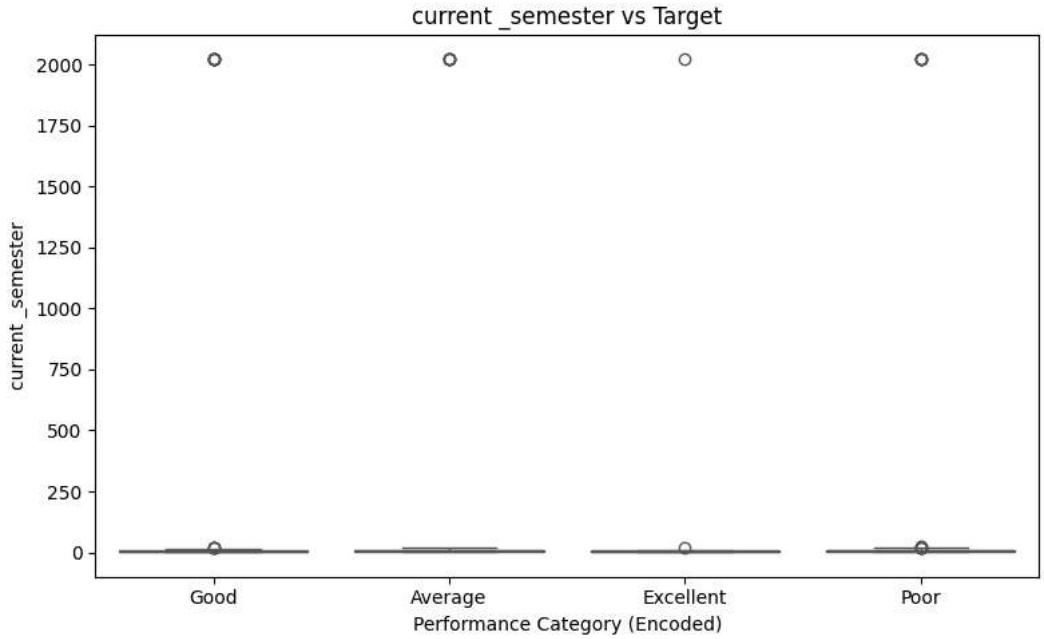
feature_3_insight

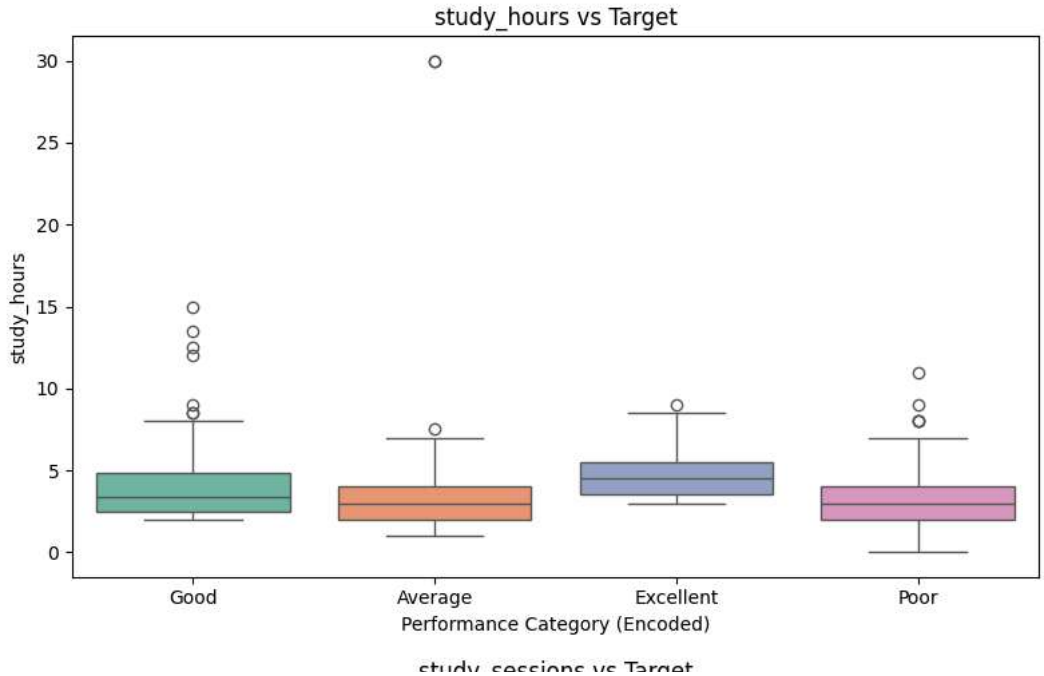
The feature 'previous_gpa' was chosen as a key feature of interest due to its high importance across multiple feature selection methods. The boxplot comparing 'previous_gpa' across performance categories shows a clear positive relationship: students with higher previous GPAs tend to fall into higher performance categories. Students in the 'Poor' and 'Average' groups generally have lower previous GPAs, while the 'Excellent' group shows consistently higher values with a tighter distribution. This supports the hypothesis that academic history is a strong indicator of future performance. This feature is also interpretable and actionable — identifying students with historically low GPAs can help

C.4 Explore Feature of Interest `column_to_check`

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Define the columns of interest
columns_to_check = [
    'current _semester', 'study_hours', 'study_sessions', 'social_media_hours',
    'average_attendance', 'skills_development_hours', 'previous_gpa',
    'current_gpa', 'completed_credits', 'house_income'
]

# Ensure 'target' exists in the dataset
df = df.dropna(subset=['target'])

# Generate boxplots for each numerical feature vs target
for col in columns_to_check:
    plt.figure(figsize=(8, 5))
    sns.boxplot(data=df, x='target', y=col, palette='Set2')
    plt.title(f"{col} vs Target")
    plt.xlabel("Performance Category (Encoded)")
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

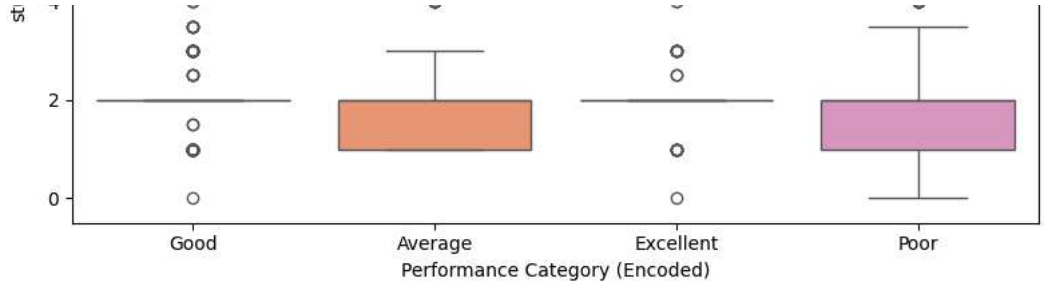


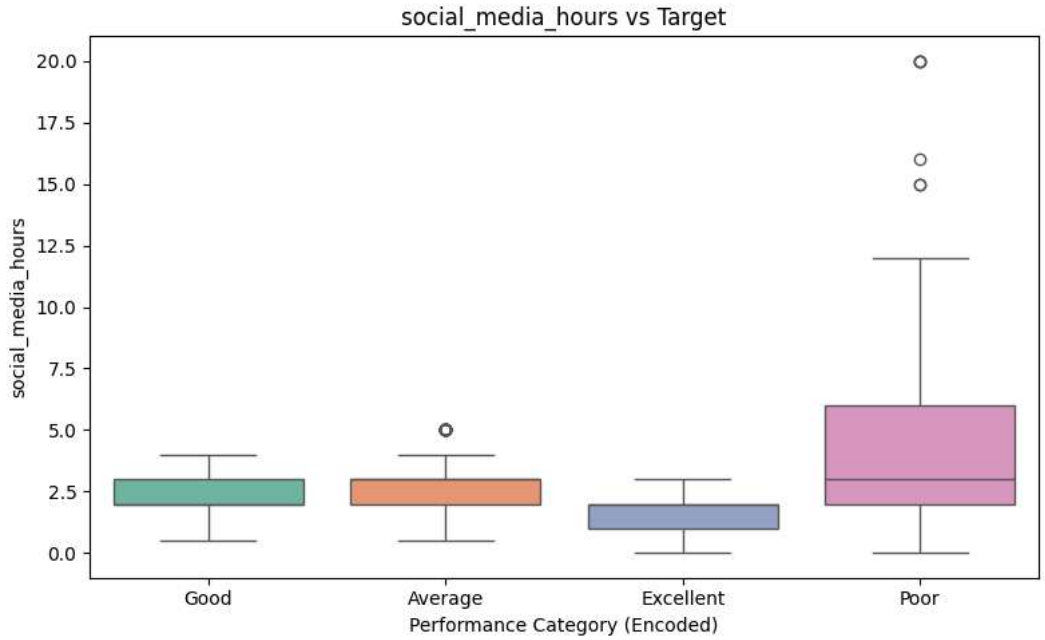


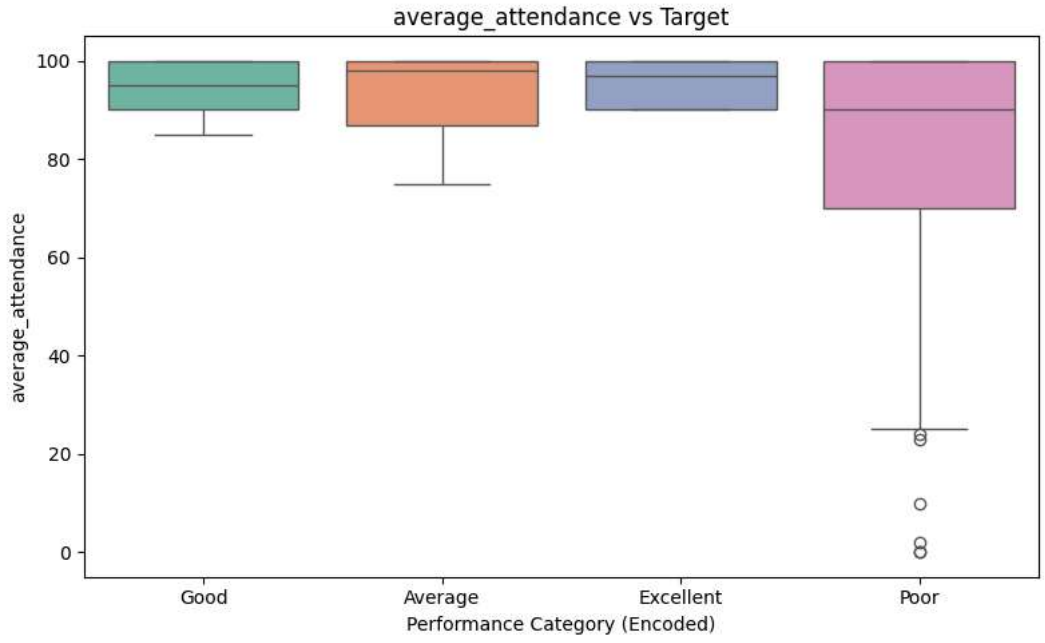
study_sessions vs Target

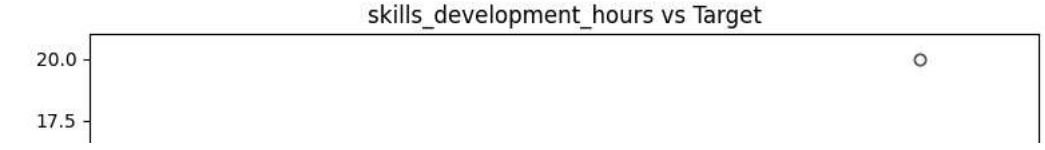
```
# Get the updated list of object-type columns (categorical features)
categorical_cols = df.select_dtypes(include='object').columns.tolist()

# Plot relationship between each categorical column and the target
```

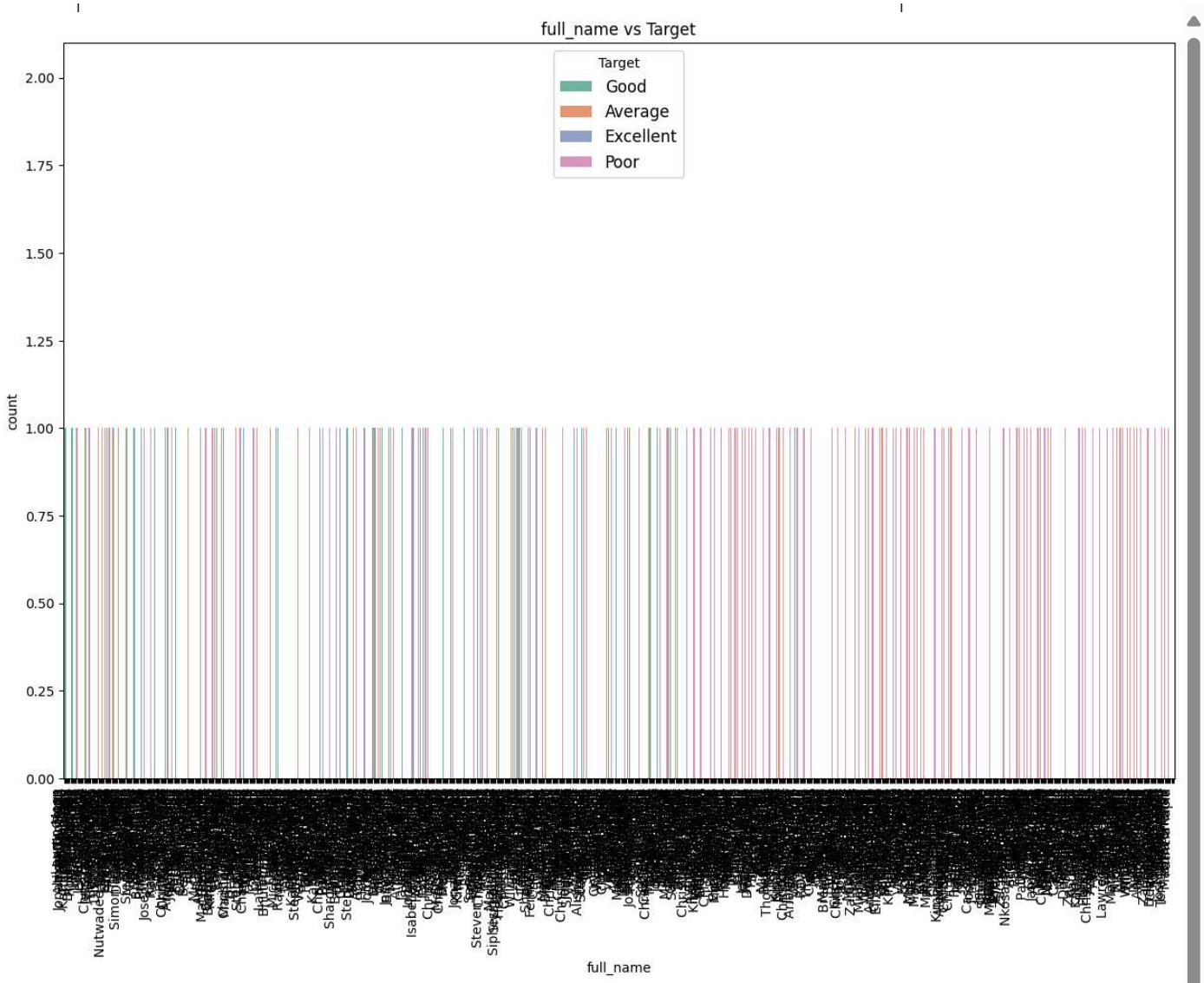
```
for col in categorical_cols:
    plt.figure(figsize=(15, 10))
    sns.countplot(data=df, x=col, hue='target', palette='Set2')
    plt.title(f"{col} vs Target")
    plt.xticks(rotation=90, ha='right', fontsize=10) # Increase x-tick fontsize and rotate for readability
    plt.yticks(fontsize=10)  # Increase y-tick fontsize
    plt.legend(title="Target", fontsize=12)  # Increase legend fontsize
    plt.show()
```

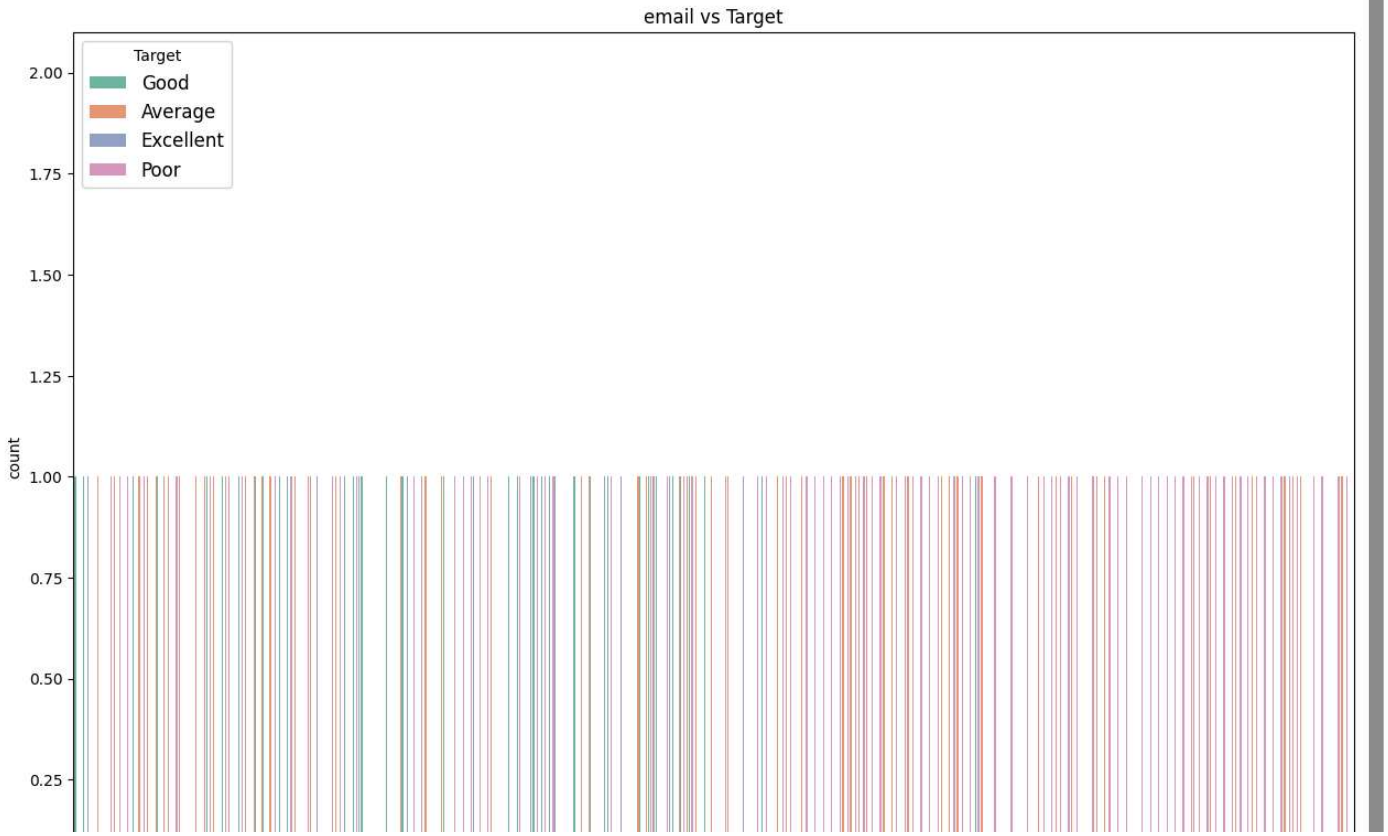


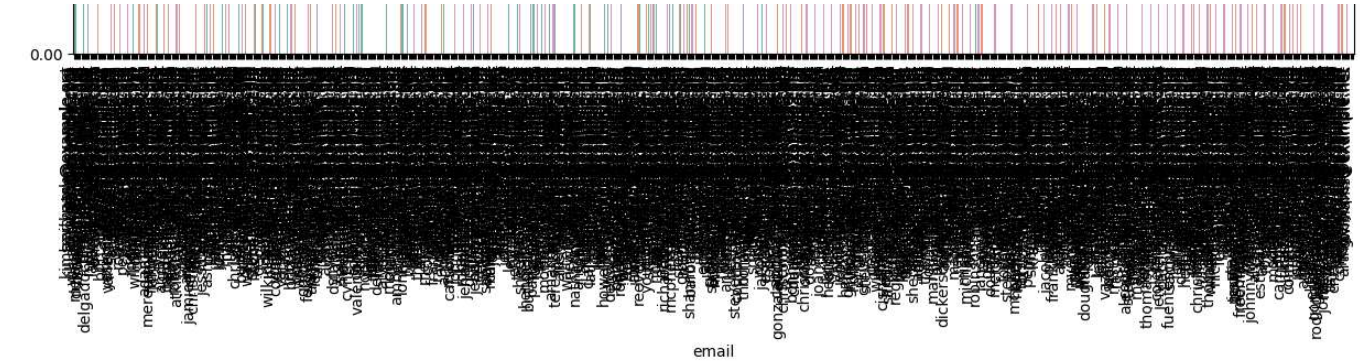


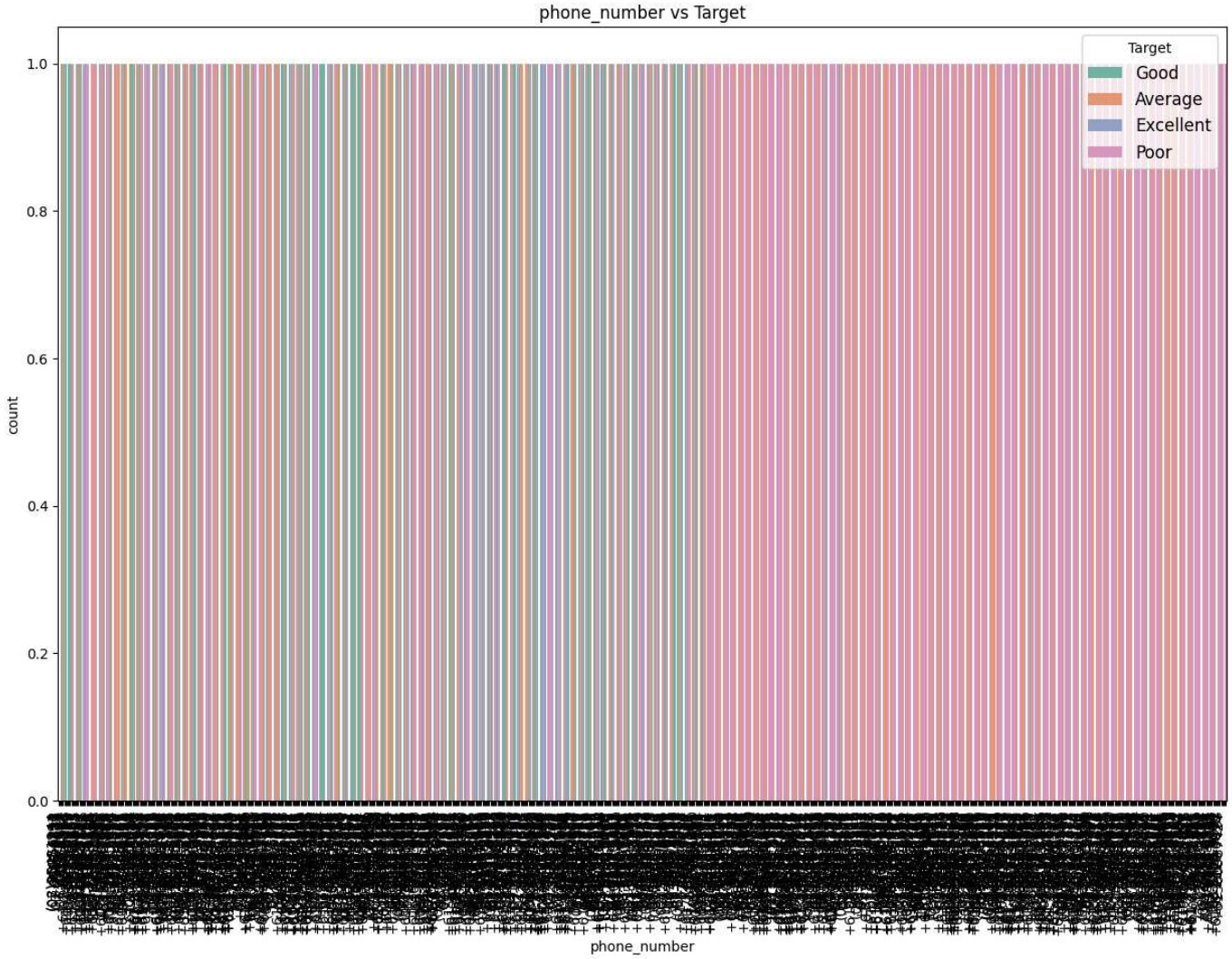




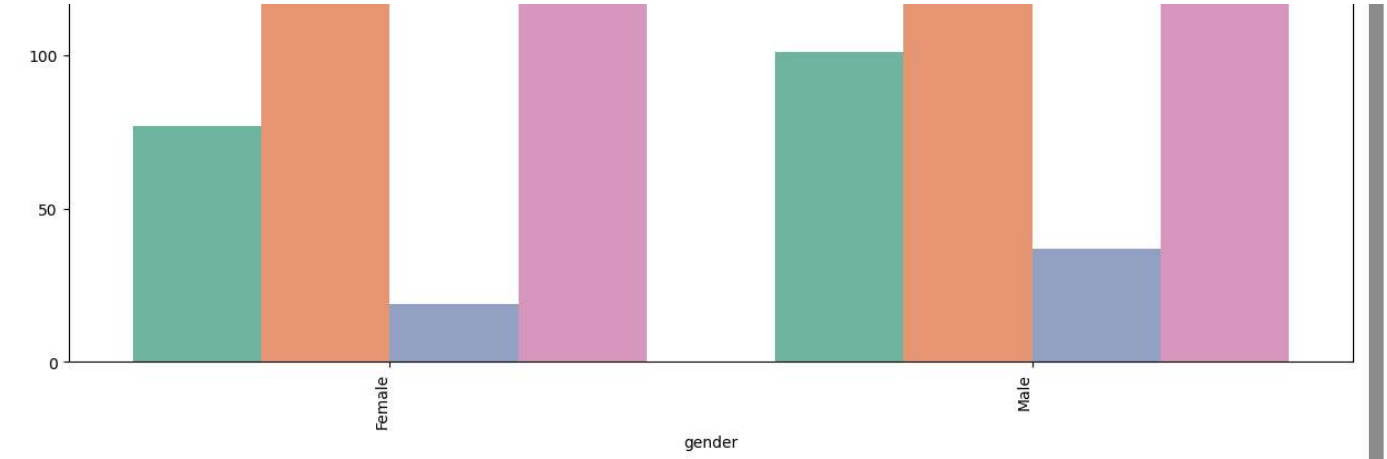

full_name vs Target
Target
Good
Average
Excellent
Poor
count
full_name

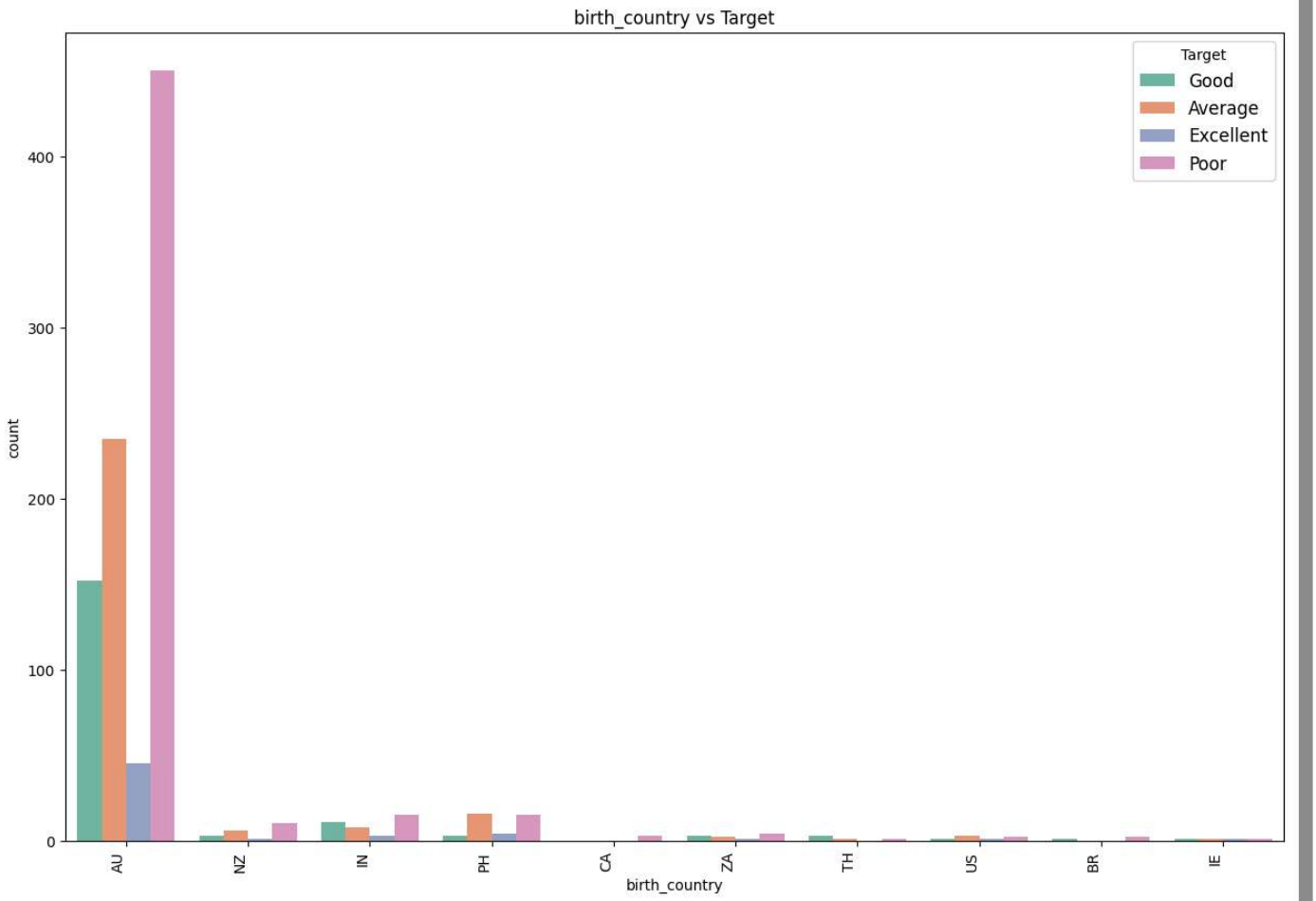

email vs Target
Target
Good
Average
Excellent
Poor
count

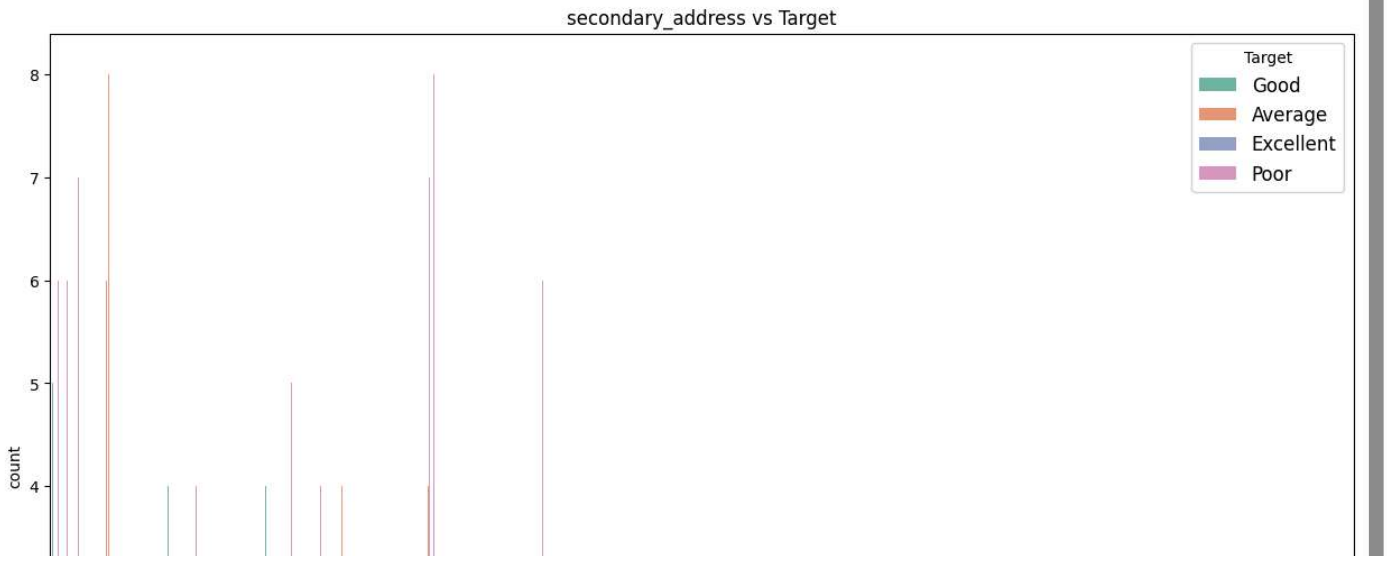

0.00
email

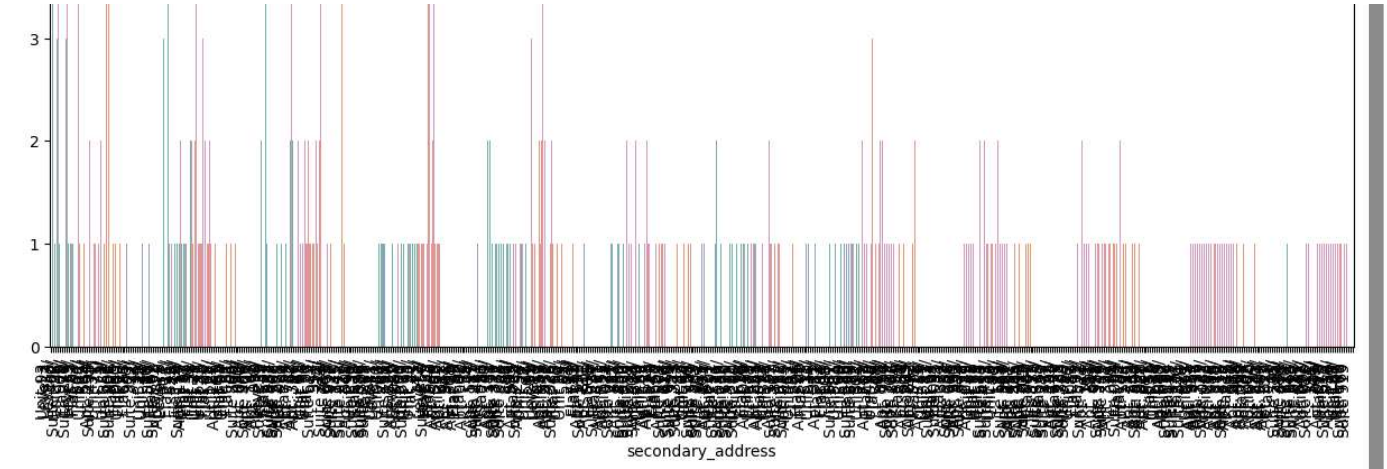

phone_number vs Target
Target
Good
Average
Excellent
Poor
1.0
0.8
0.6
0.4
0.2
0.0
count
phone_number

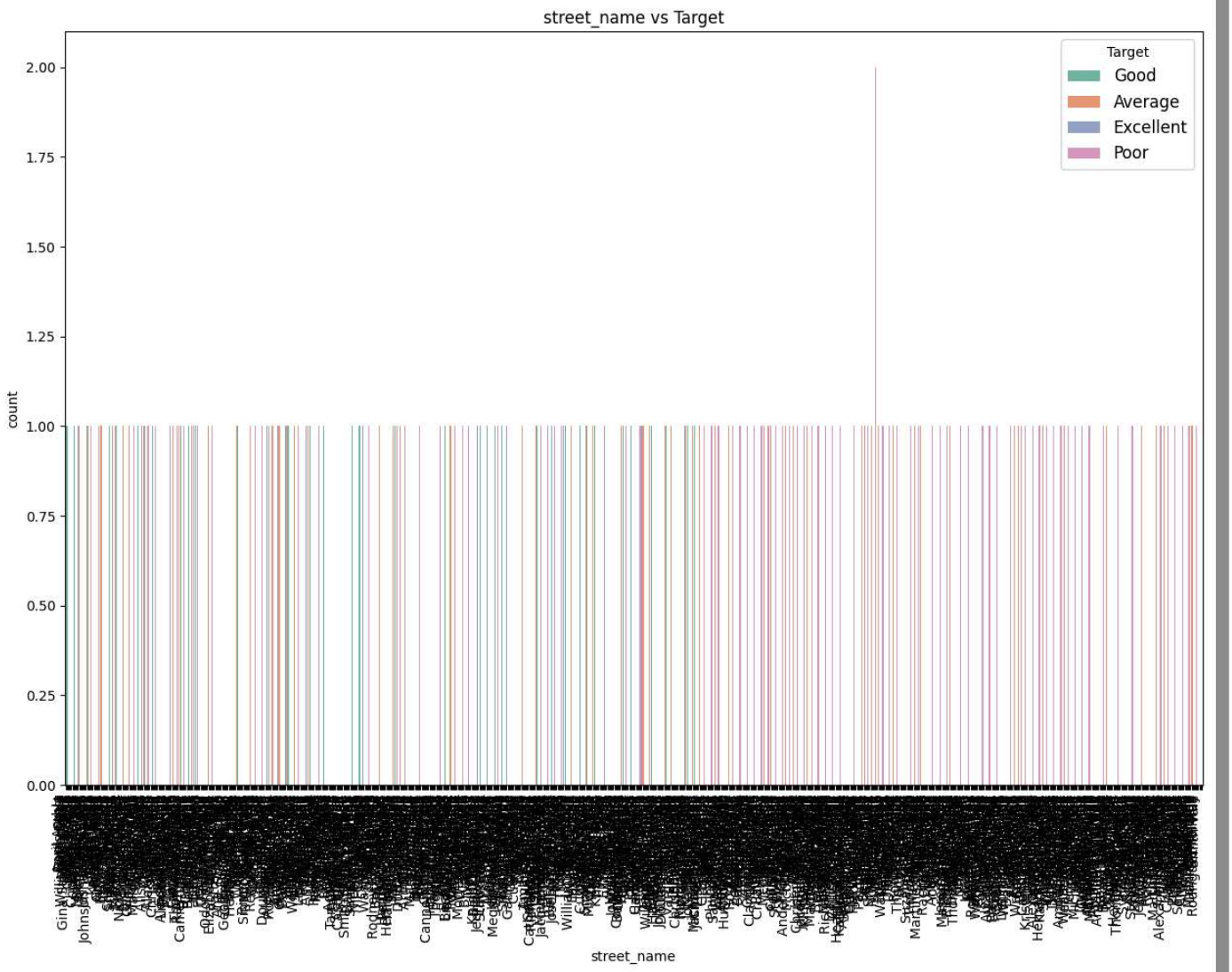

gender vs Target
Target
Good
Average
Excellent
Poor
250
200
150
count

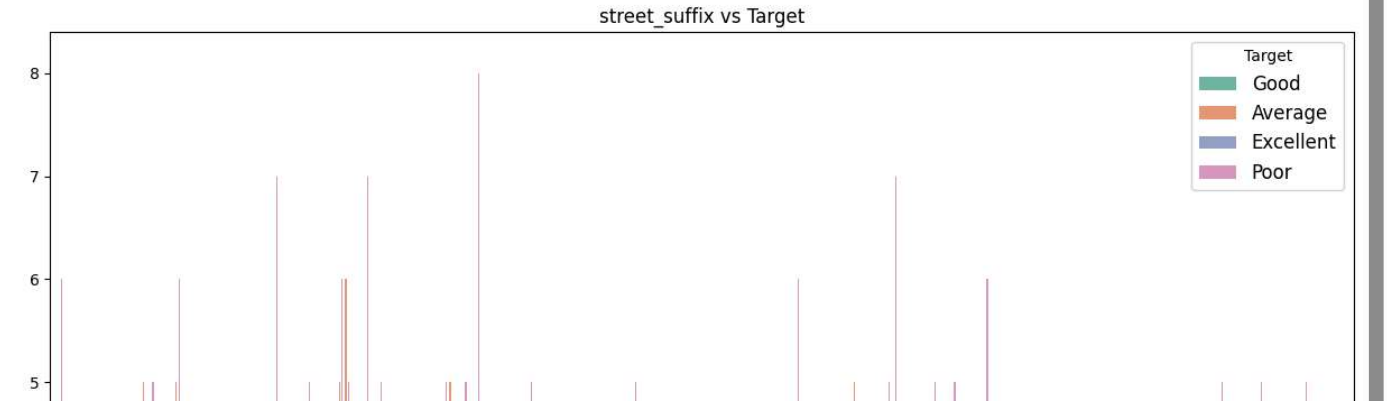

100
50
0
Female
Male
gender

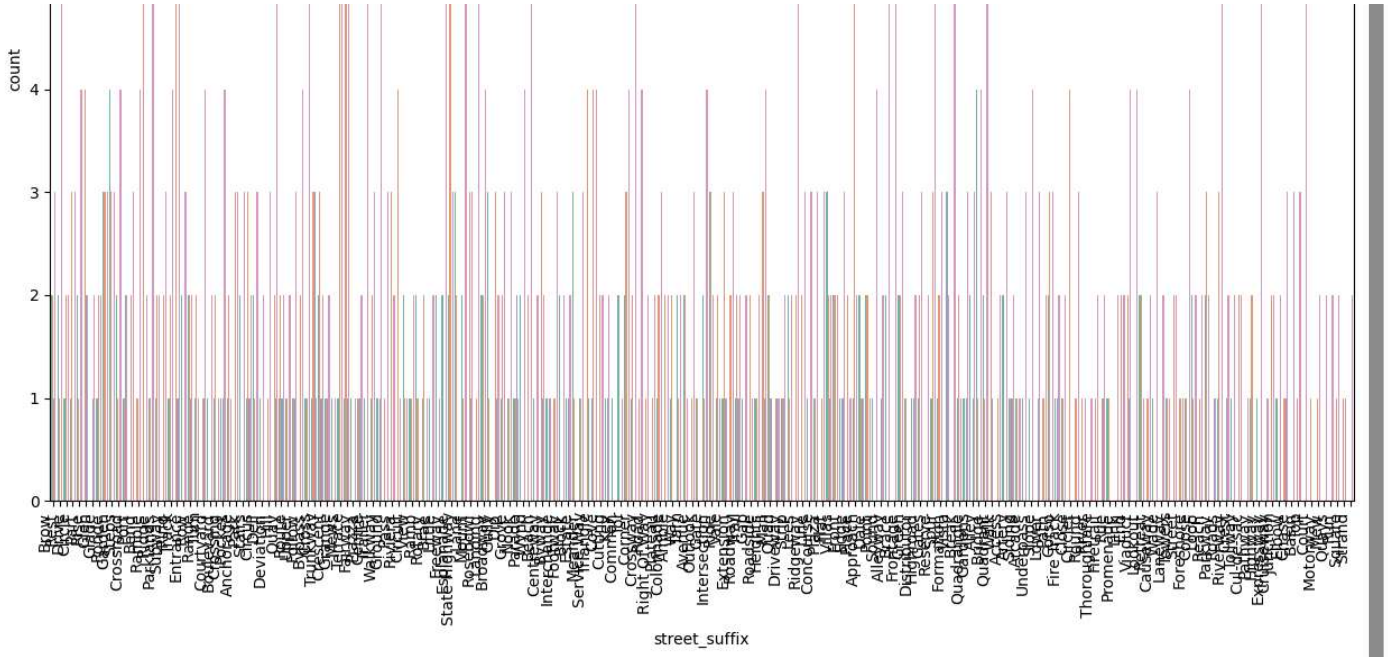

birth_country vs Target
Target
Good
Average
Excellent
Poor
400
300
200
100
0
count
AU
NZ
IN
PH
CA
ZA
TH
US
BR
IE
birth_country

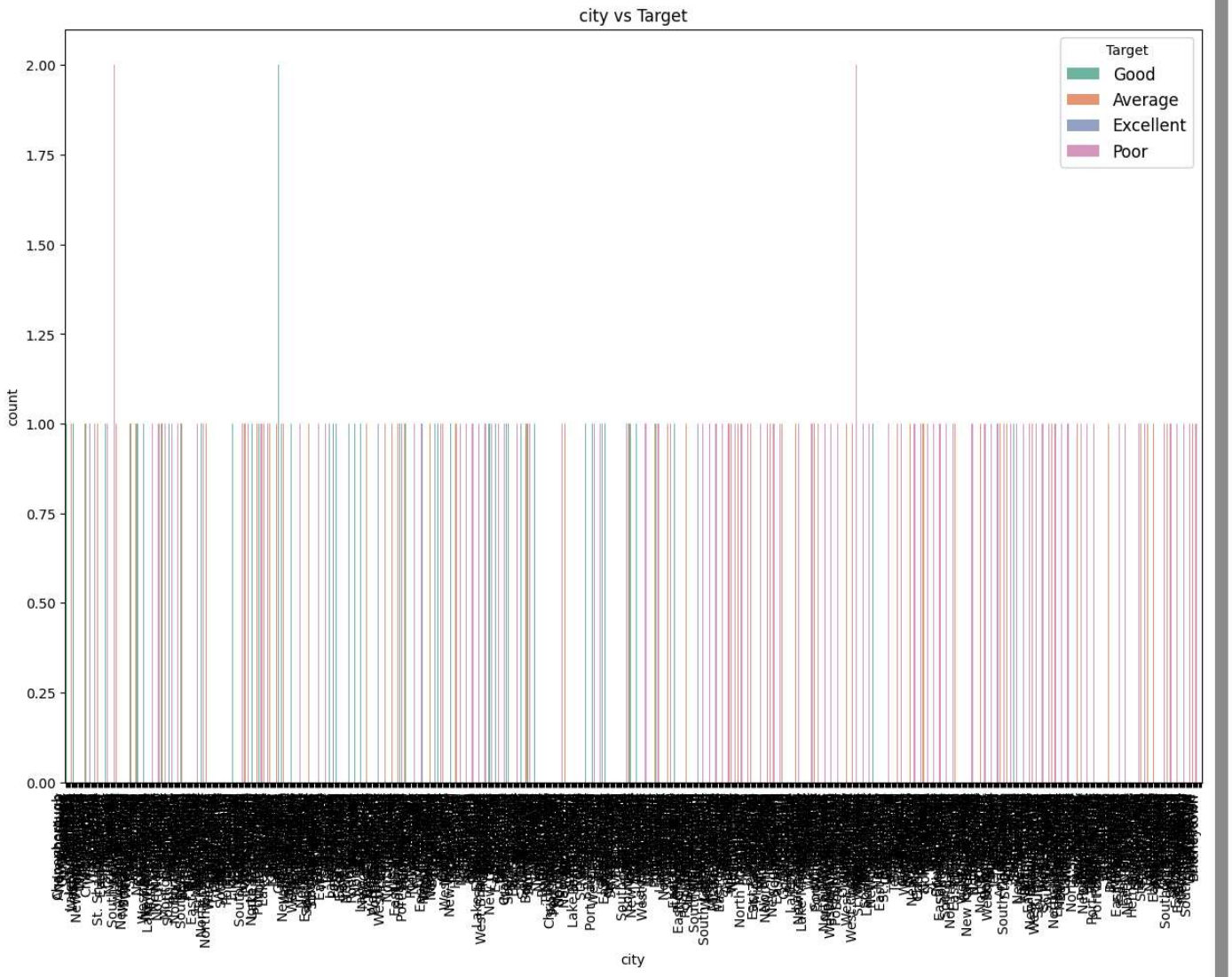

secondary_address vs Target
Target
Good
Average
Excellent
Poor
8
7
6
5
4
count

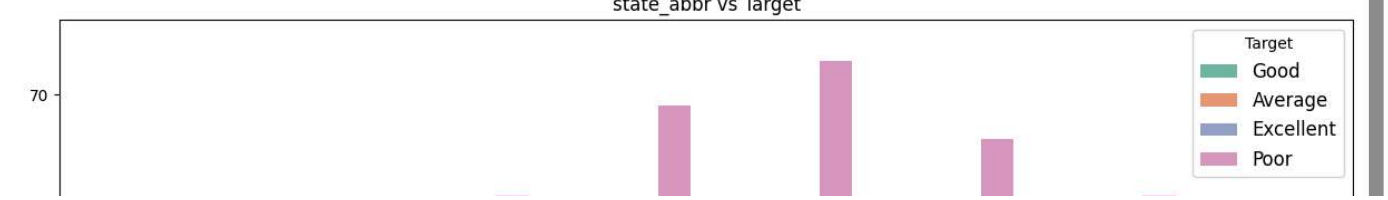

secondary_address

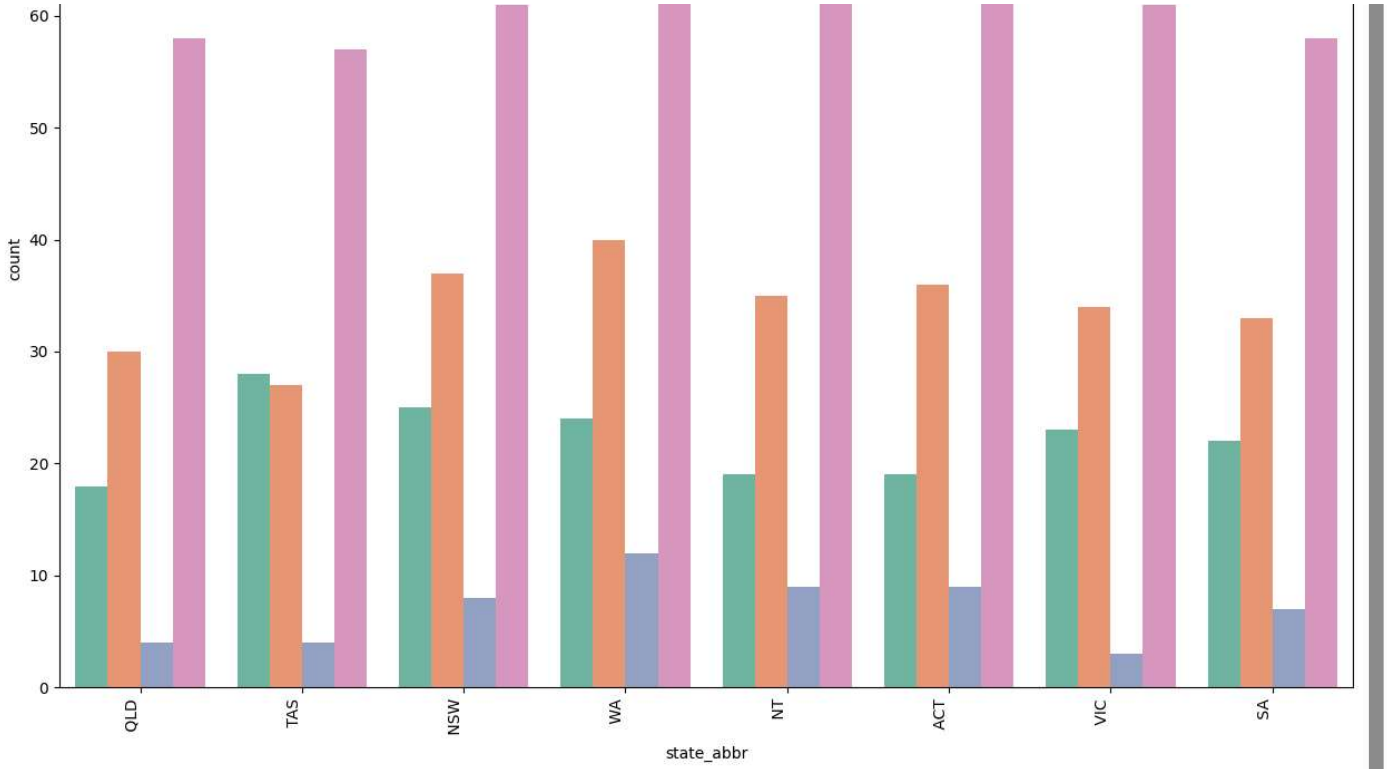

street_name vs Target
Target
Good
Average
Excellent
Poor
count
street_name

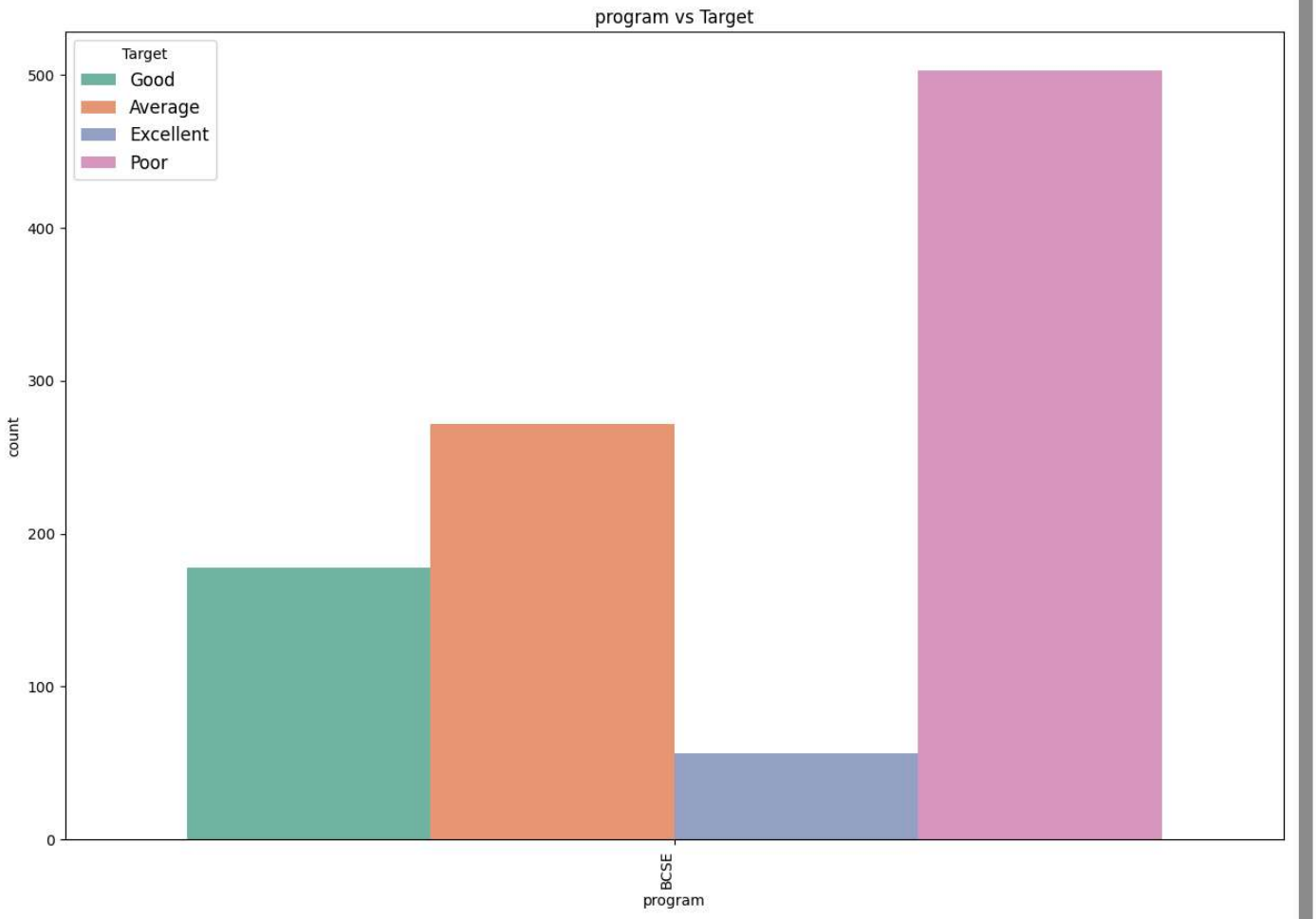

street_suffix vs Target
Target
Good
Average
Excellent
Poor

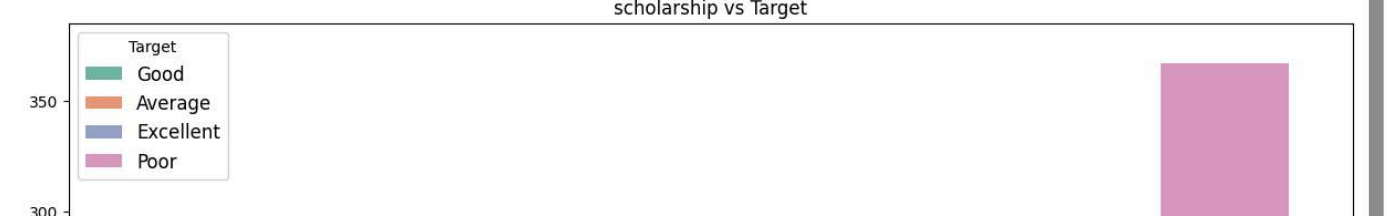

count
street_suffix

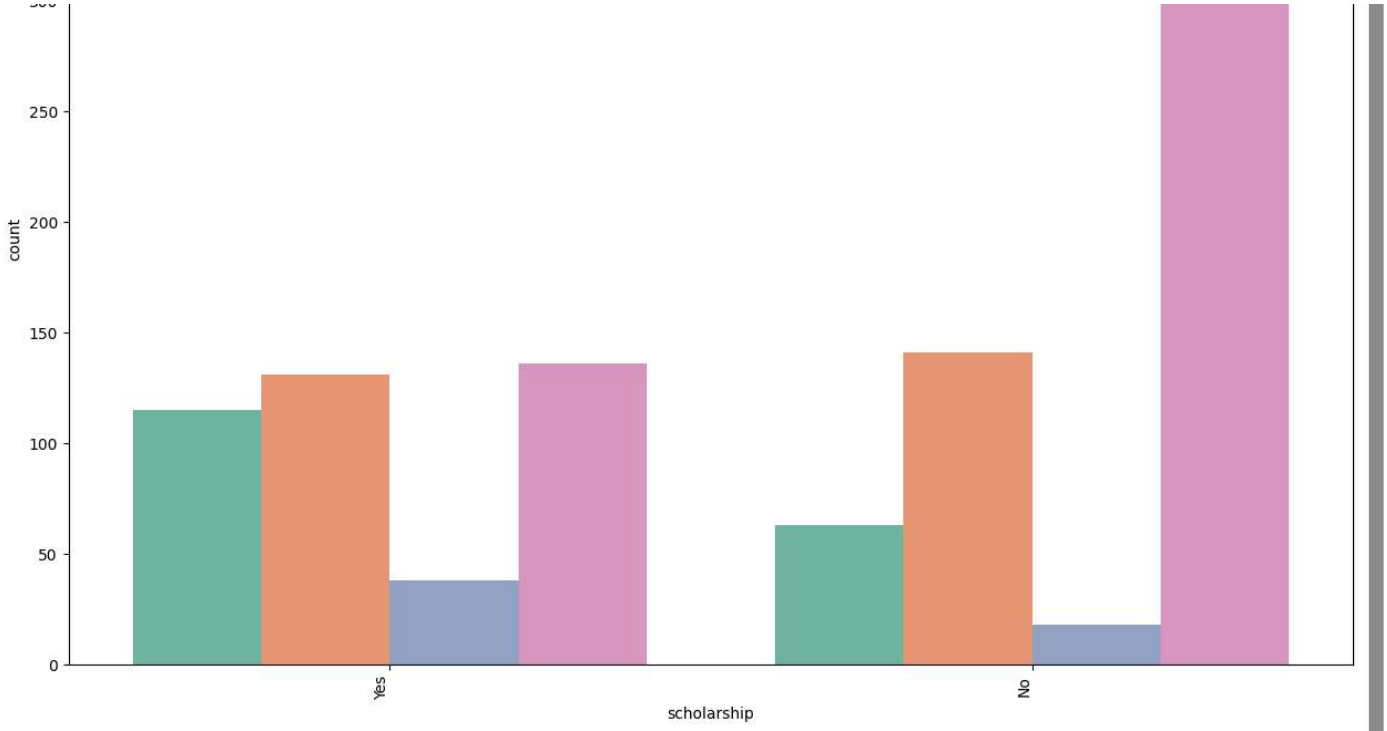

city vs Target
Target
Good
Average
Excellent
Poor
count
city

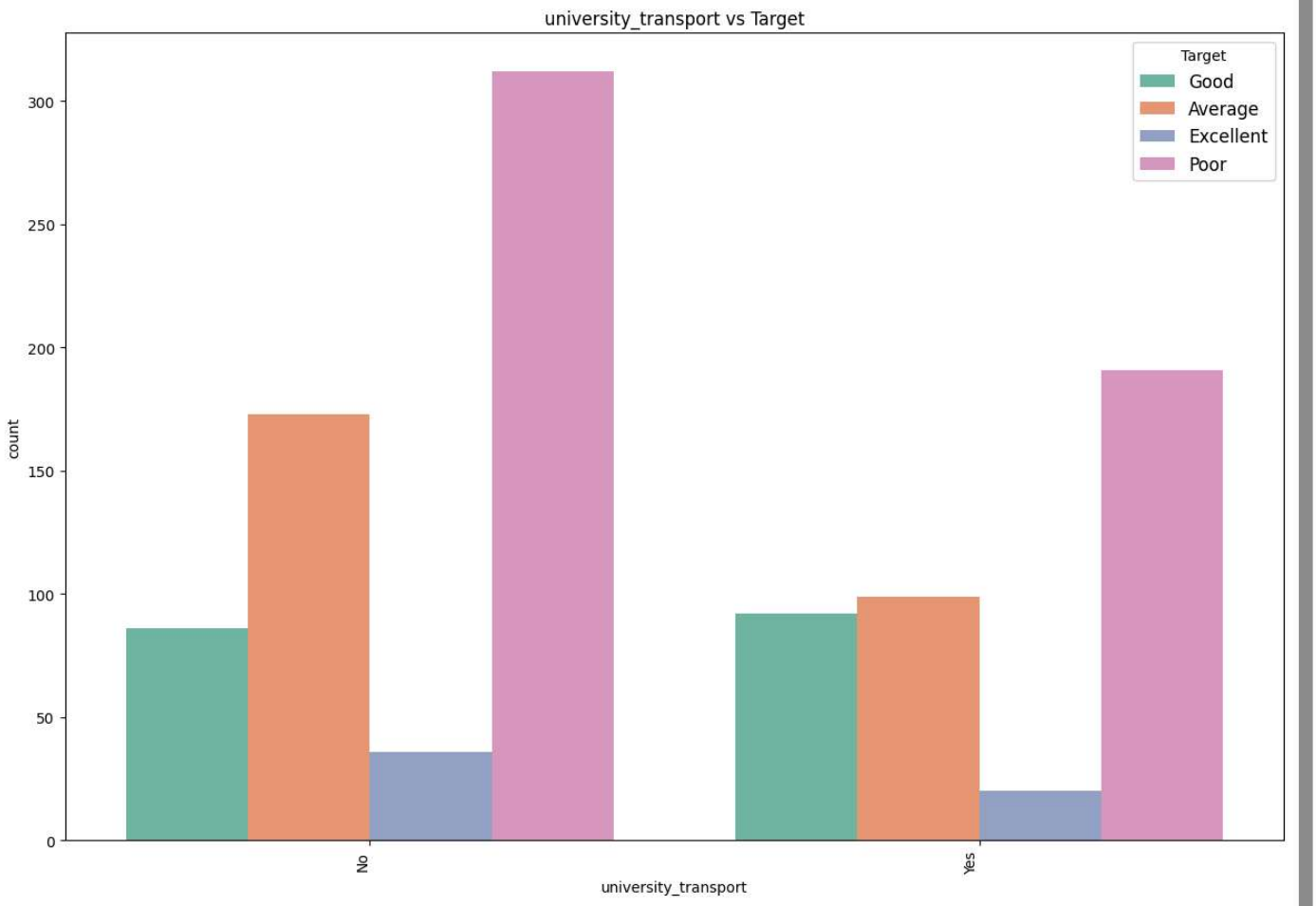

state_abbr vs Target
Target
Good
Average
Excellent
Poor

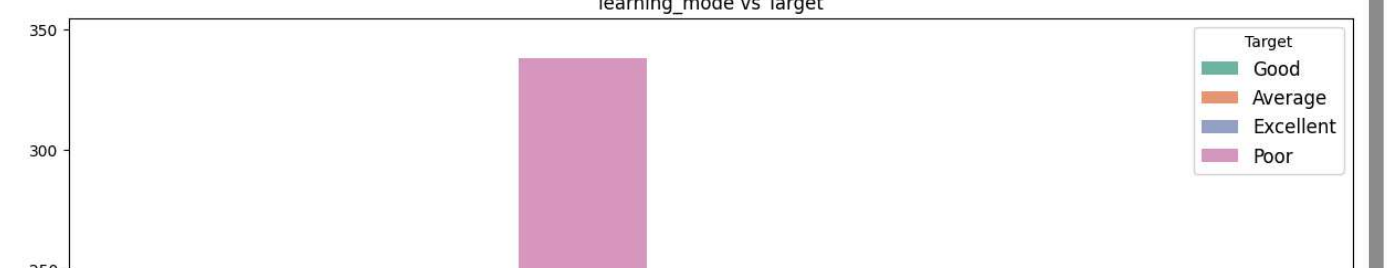

count
state_abbr
QLD
TAS
NSW
WA
NT
ACT
VIC
SA

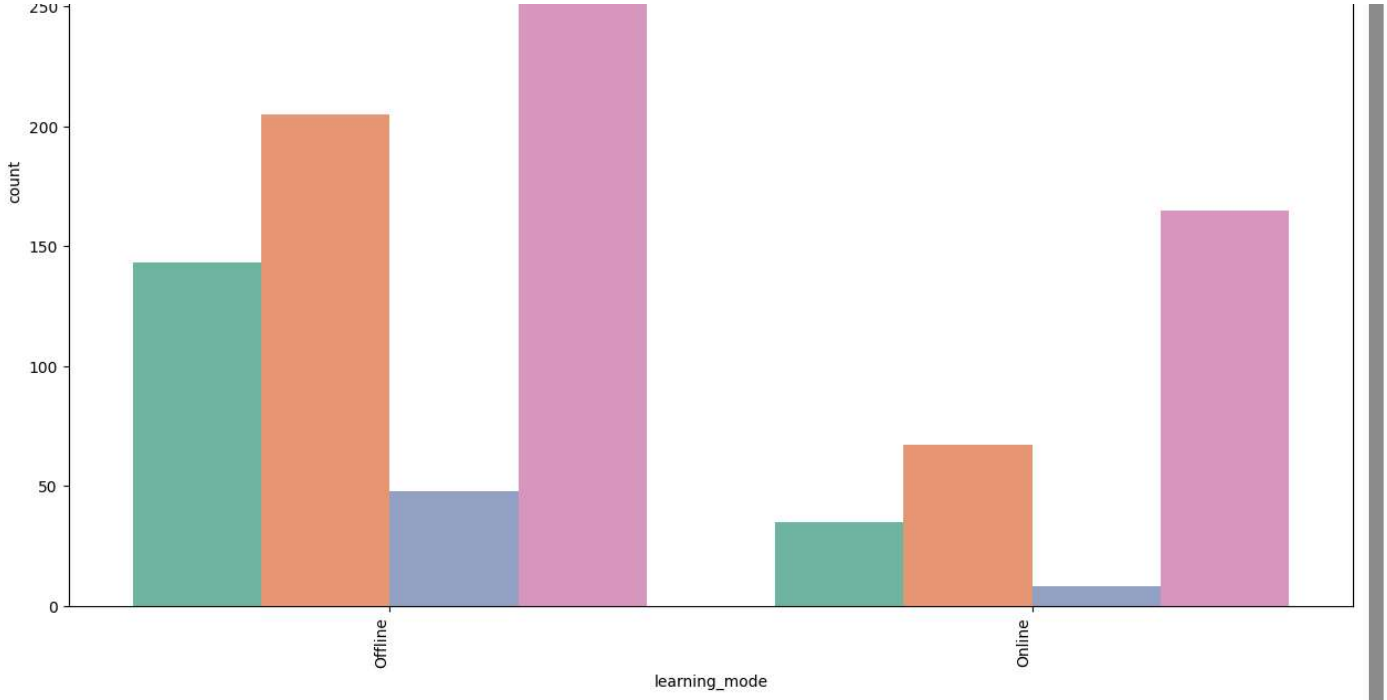

program vs Target
Target
Good
Average
Excellent
Poor
count
program
BCSE

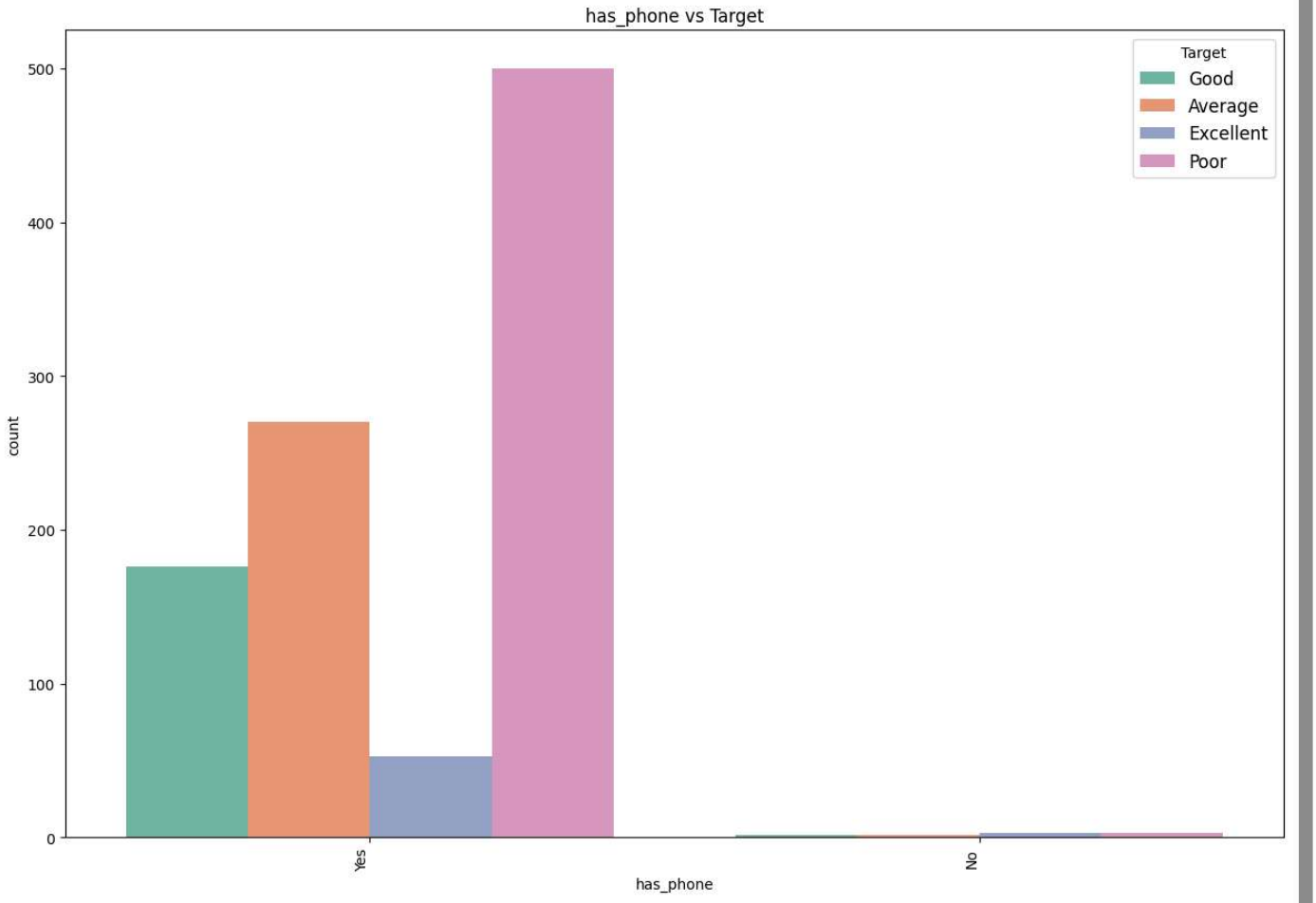

scholarship vs Target
Target
Good
Average
Excellent
Poor

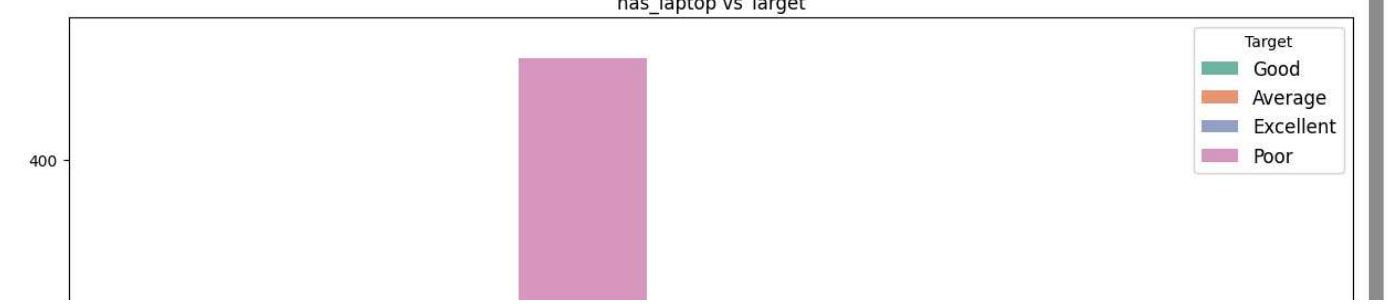

count
250
200
150
100
50
0
Yes
No
scholarship

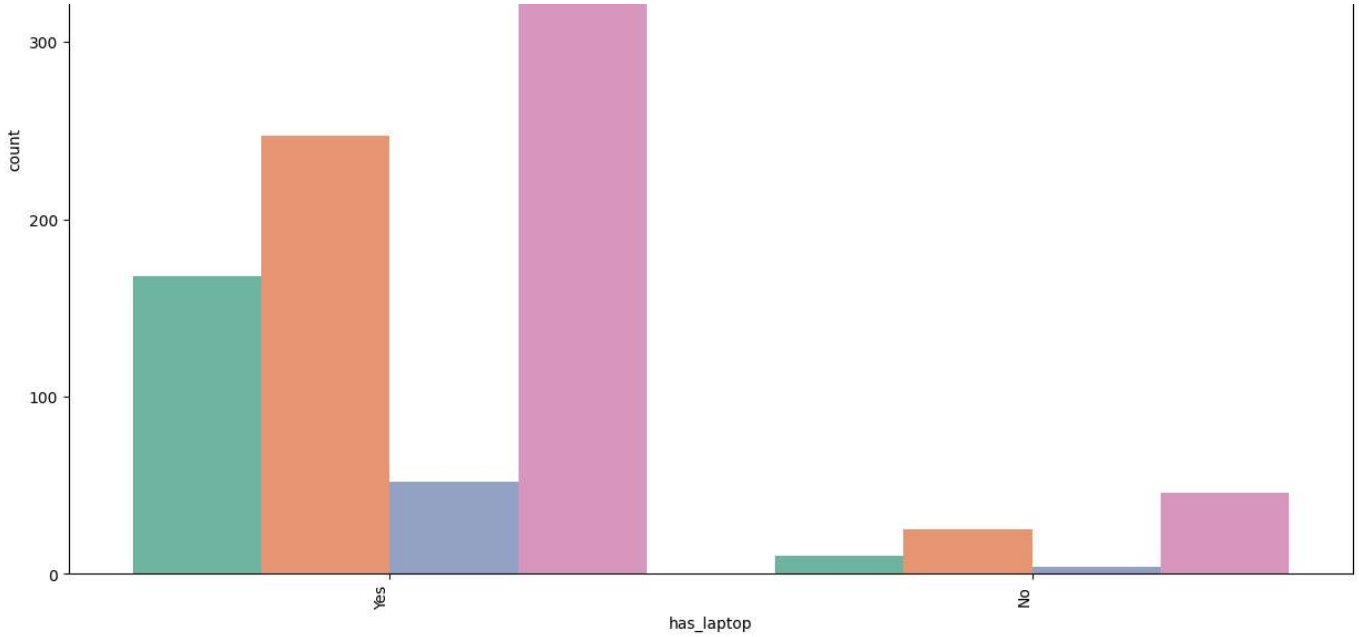

university_transport vs Target
Target
Good
Average
Excellent
Poor
count
300
250
200
150
100
50
0
No
Yes
university_transport

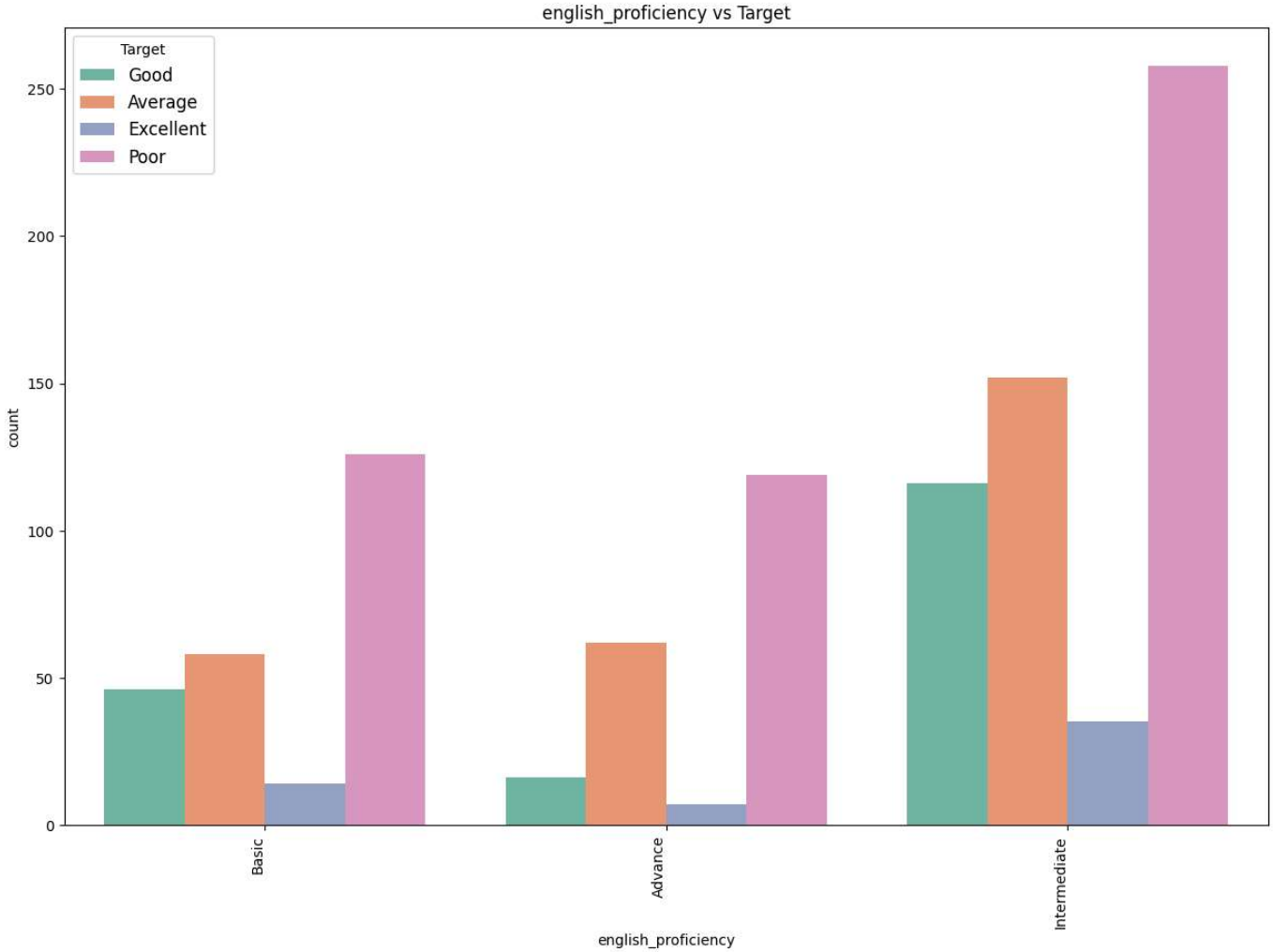

learning_mode vs Target
Target
Good
Average
Excellent
Poor
350
300

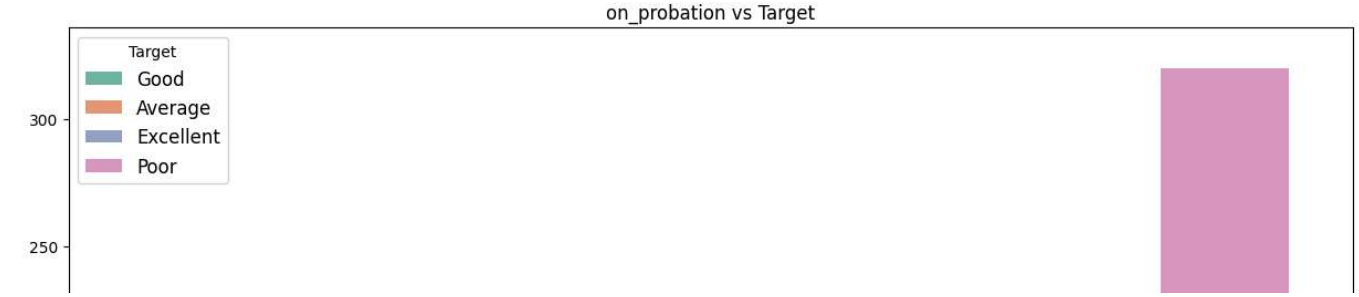

250
200
150
100
50
0
count
Offline
Online
learning_mode

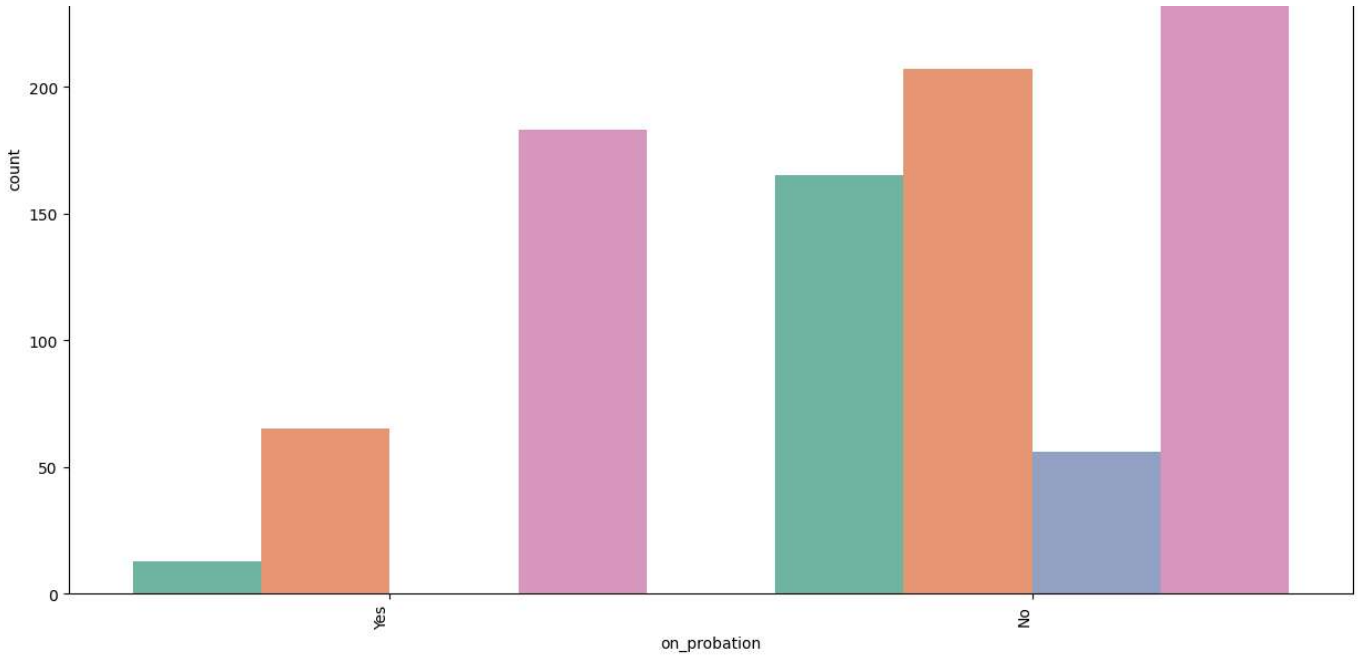

has_phone vs Target
Target
Good
Average
Excellent
Poor
500
400
300
200
100
0
count
Yes
No
has_phone

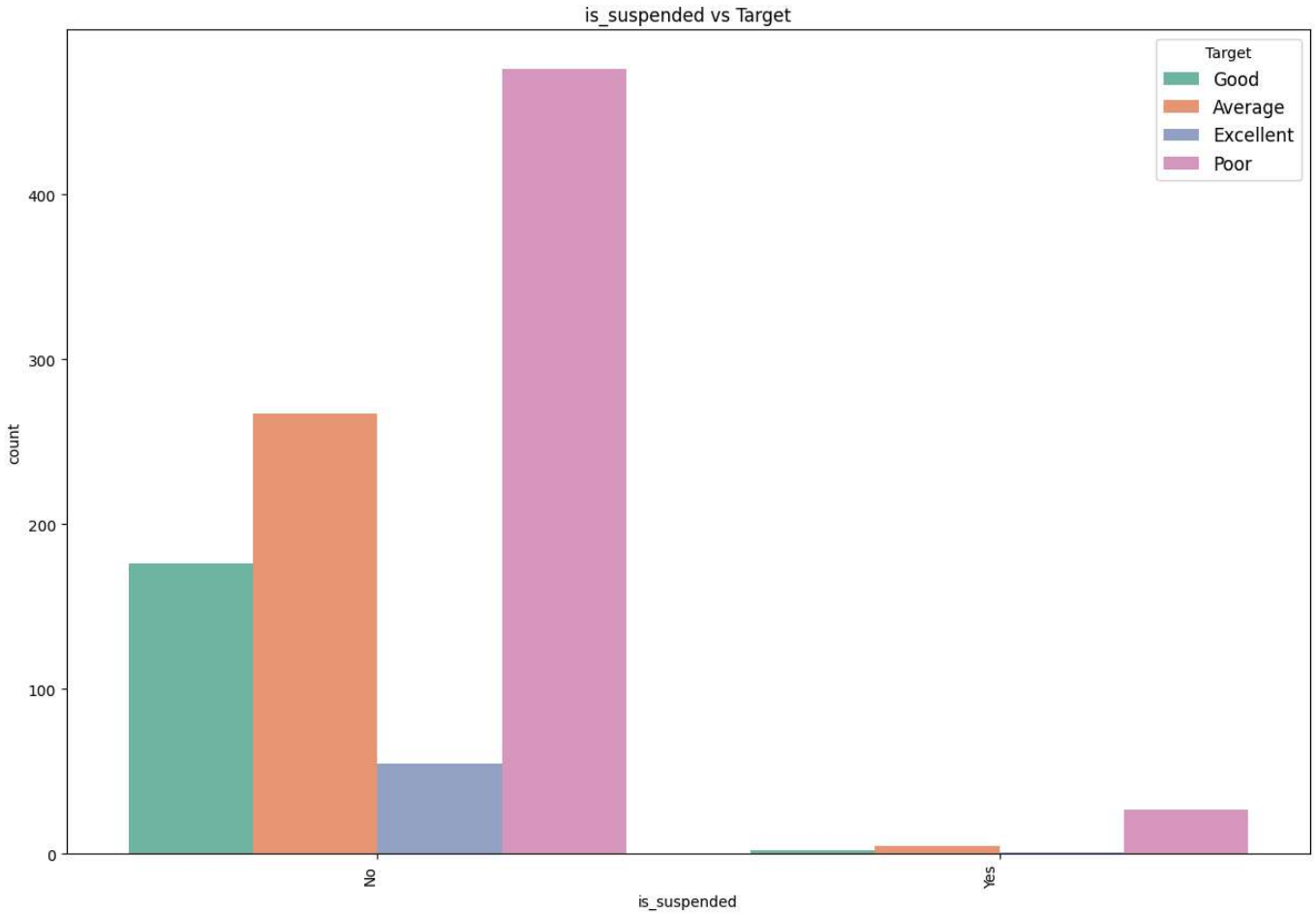

has_laptop vs Target
Target
Good
Average
Excellent
Poor
400

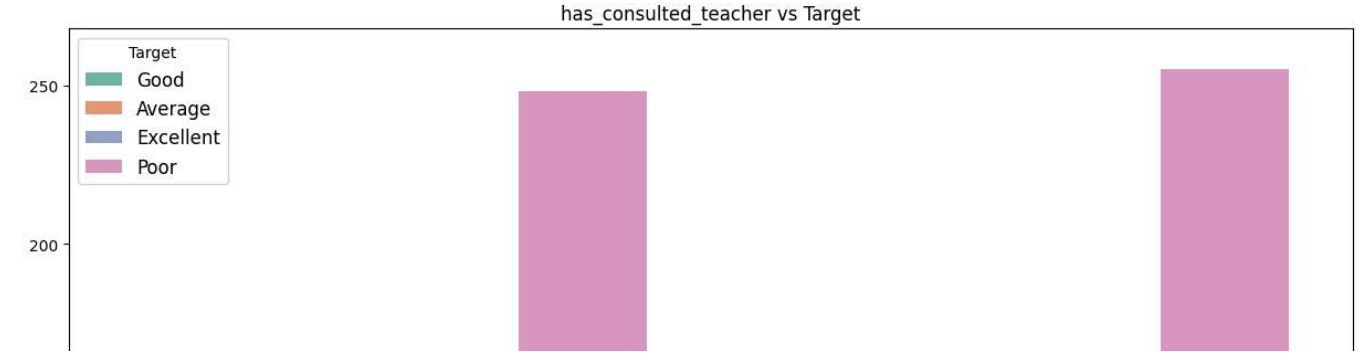

count
300
200
100
0
Yes
No
has_laptop

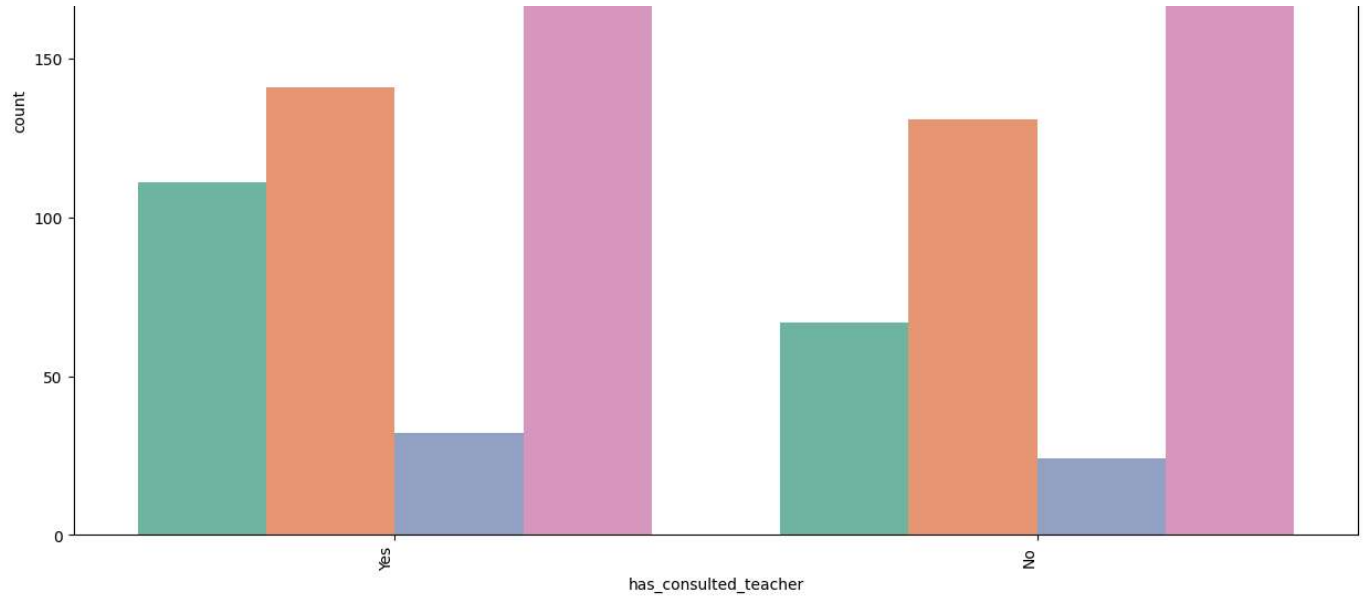

english_proficiency vs Target
Target
Good
Average
Excellent
Poor
count
250
200
150
100
50
0
Basic
Advance
Intermediate
english_proficiency

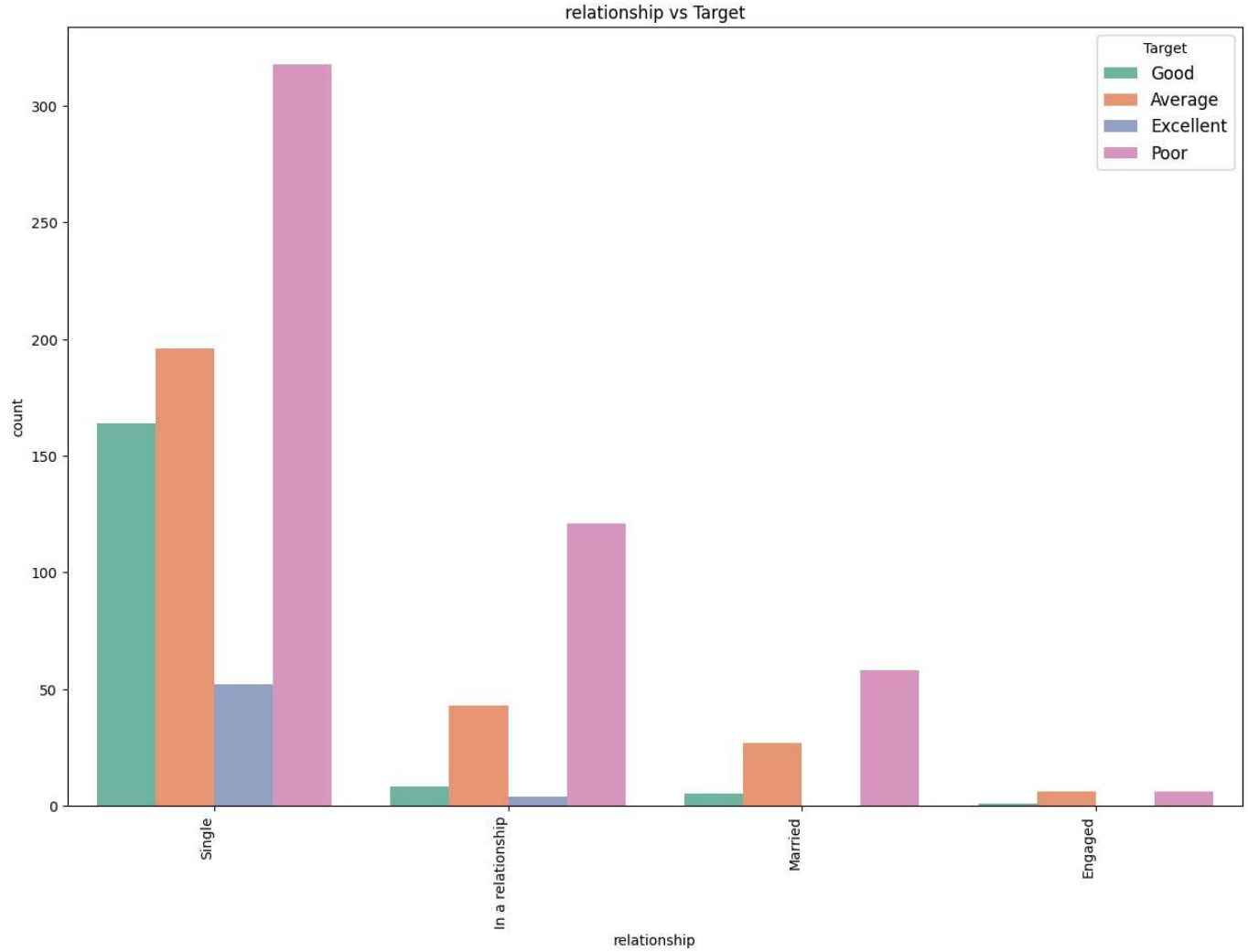

on_probation vs Target
Target
Good
Average
Excellent
Poor
300
250

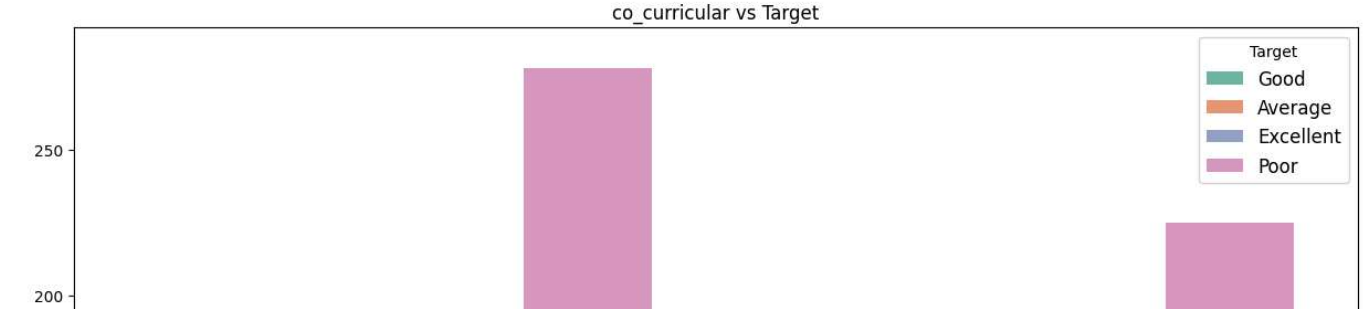

count
200
150
100
50
0
Yes
No
on_probation

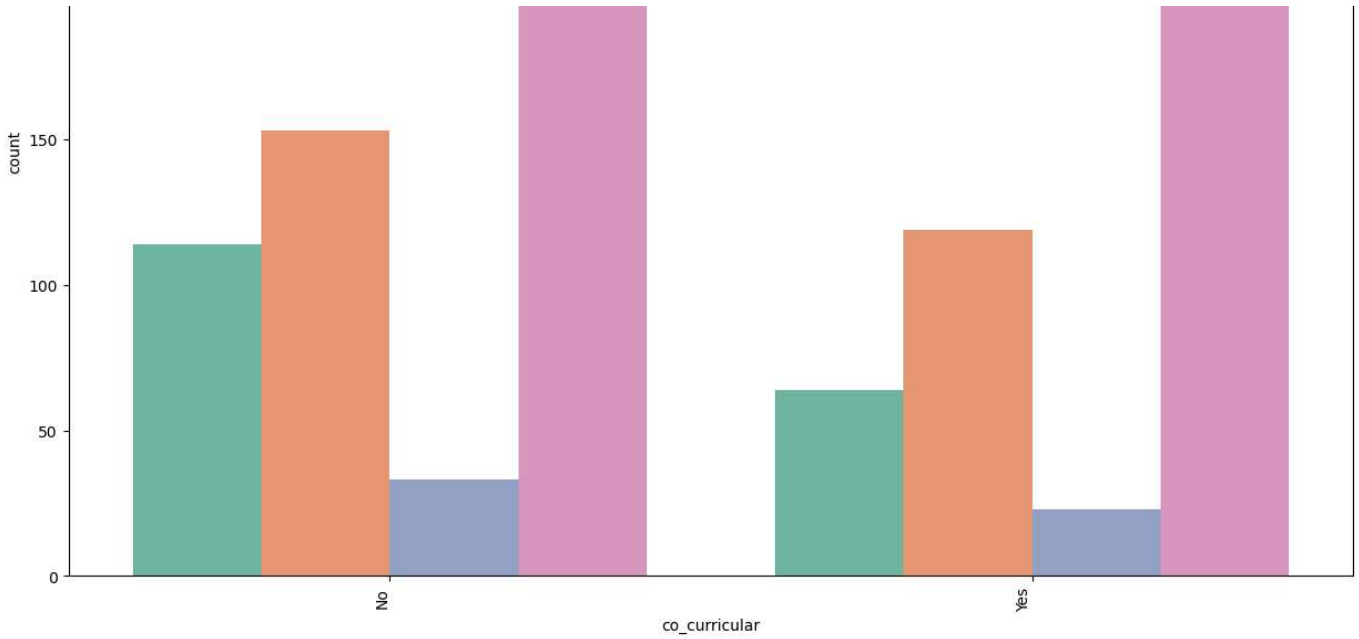

is_suspended vs Target
Target
Good
Average
Excellent
Poor
count
400
300
200
100
0
No
Yes
is_suspended

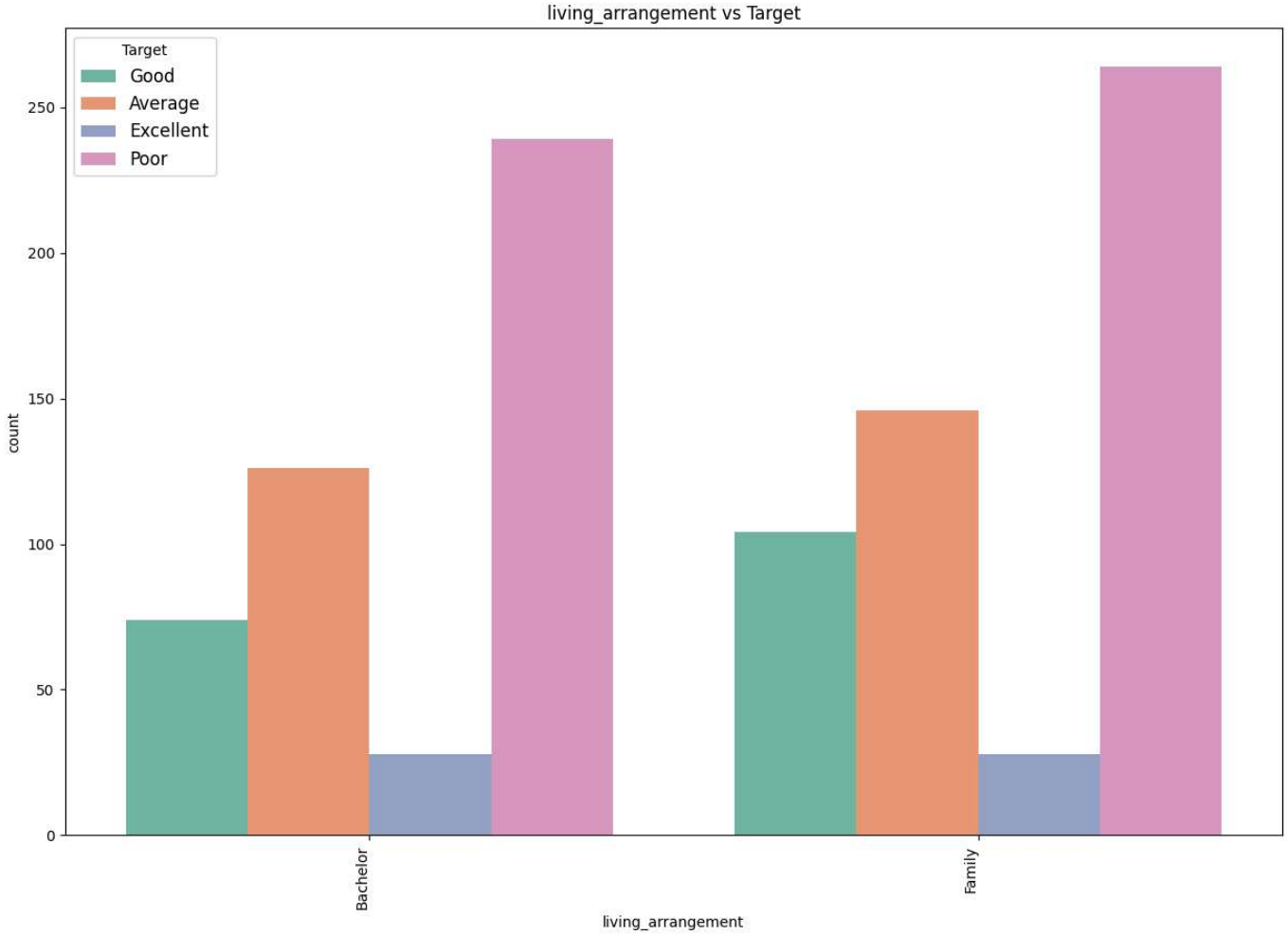

has_consulted_teacher vs Target
Target
Good
Average
Excellent
Poor
250
200

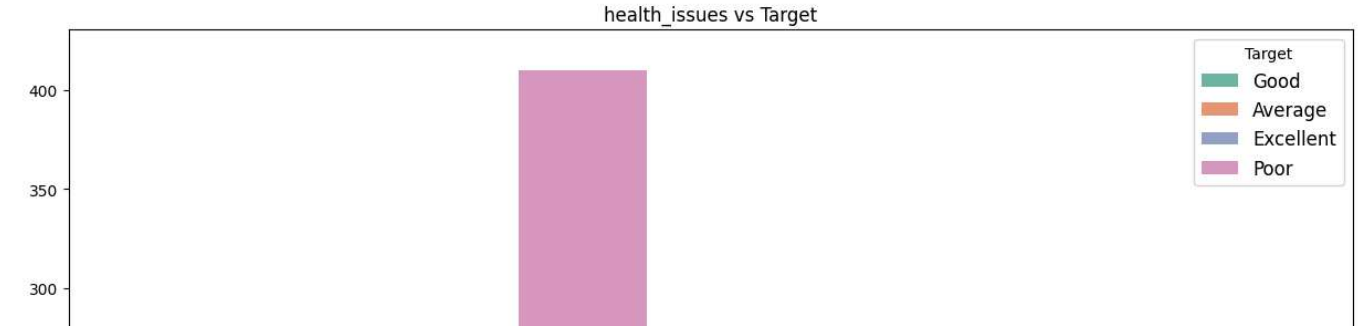

count
150
100
50
0
Yes
No
has_consulted_teacher

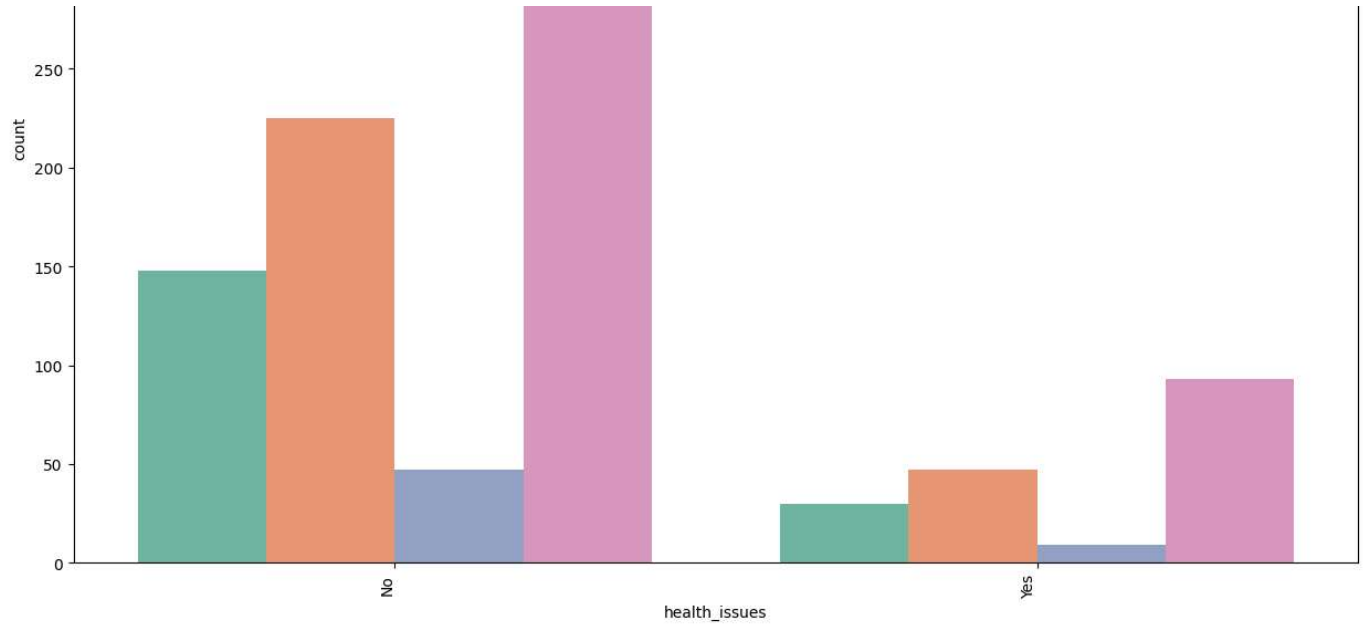

relationship vs Target
Target
Good
Average
Excellent
Poor
count
300
250
200
150
100
50
0
Single
In a relationship
Married
Engaged
relationship

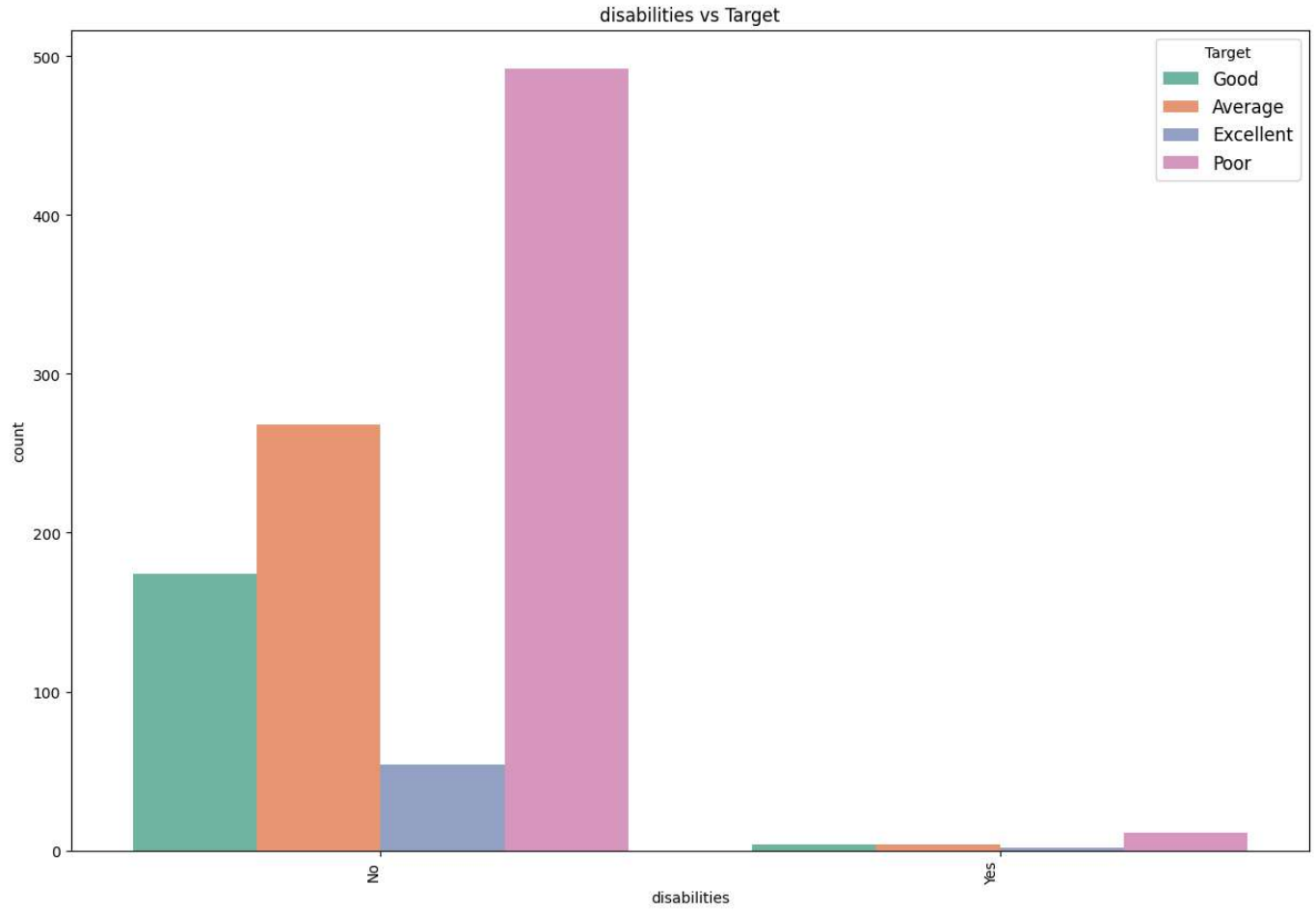

co_curricular vs Target
Target
Good
Average
Excellent
Poor
250
200


count
150
100
50
0
No
Yes
co_curricular

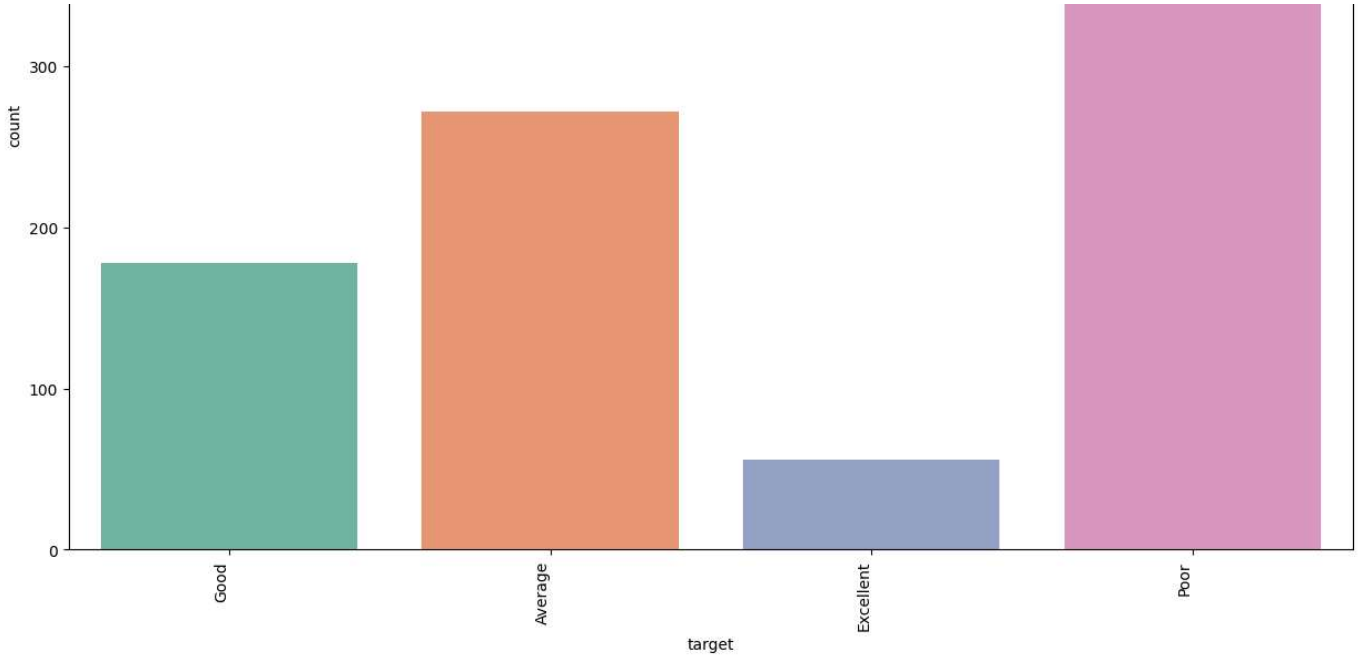

living_arrangement vs Target
Target
Good
Average
Excellent
Poor
count
250
200
150
100
50
0
Bachelor
Family
living_arrangement

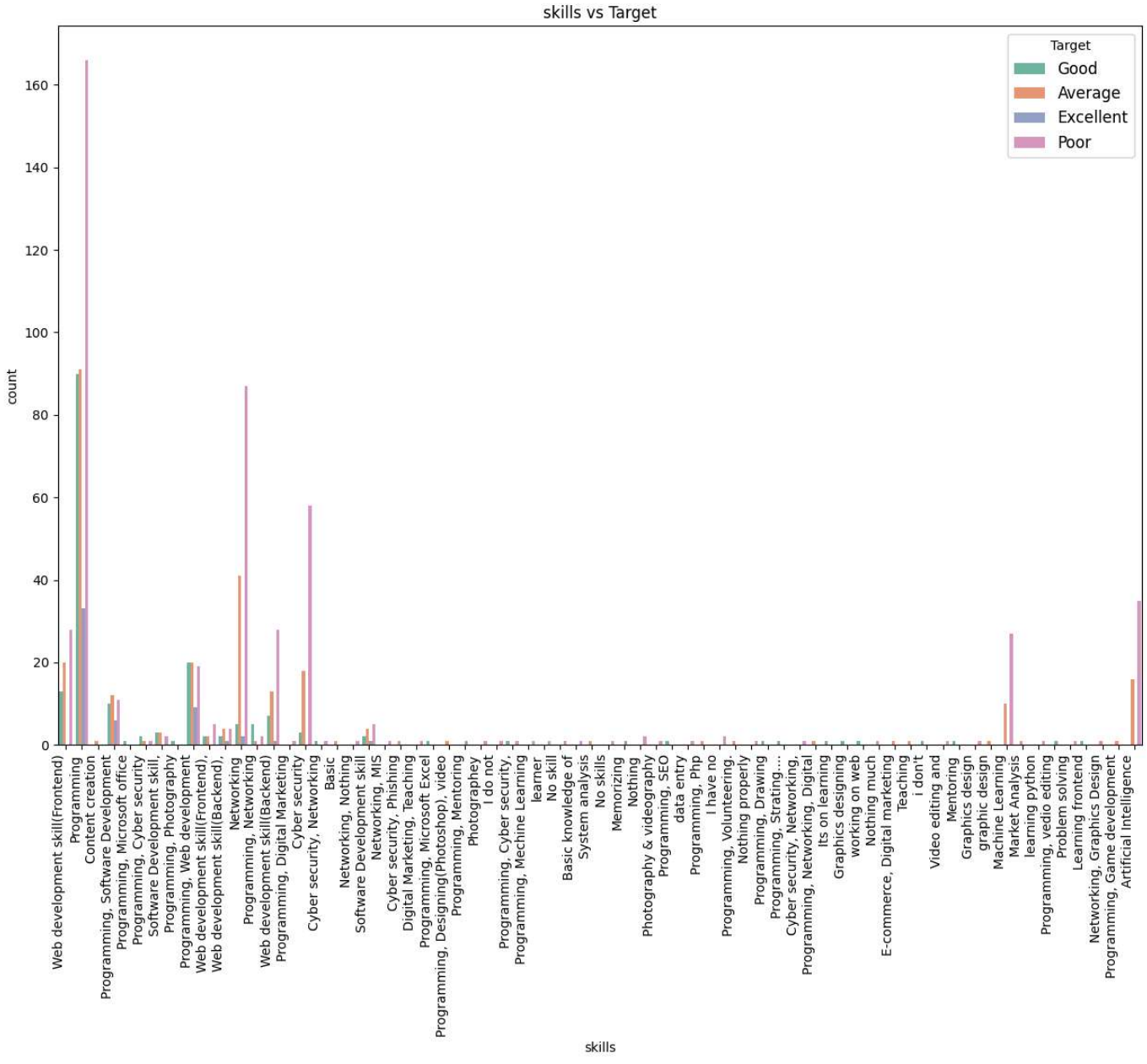

health_issues vs Target
Target
Good
Average
Excellent
Poor
400
350
300



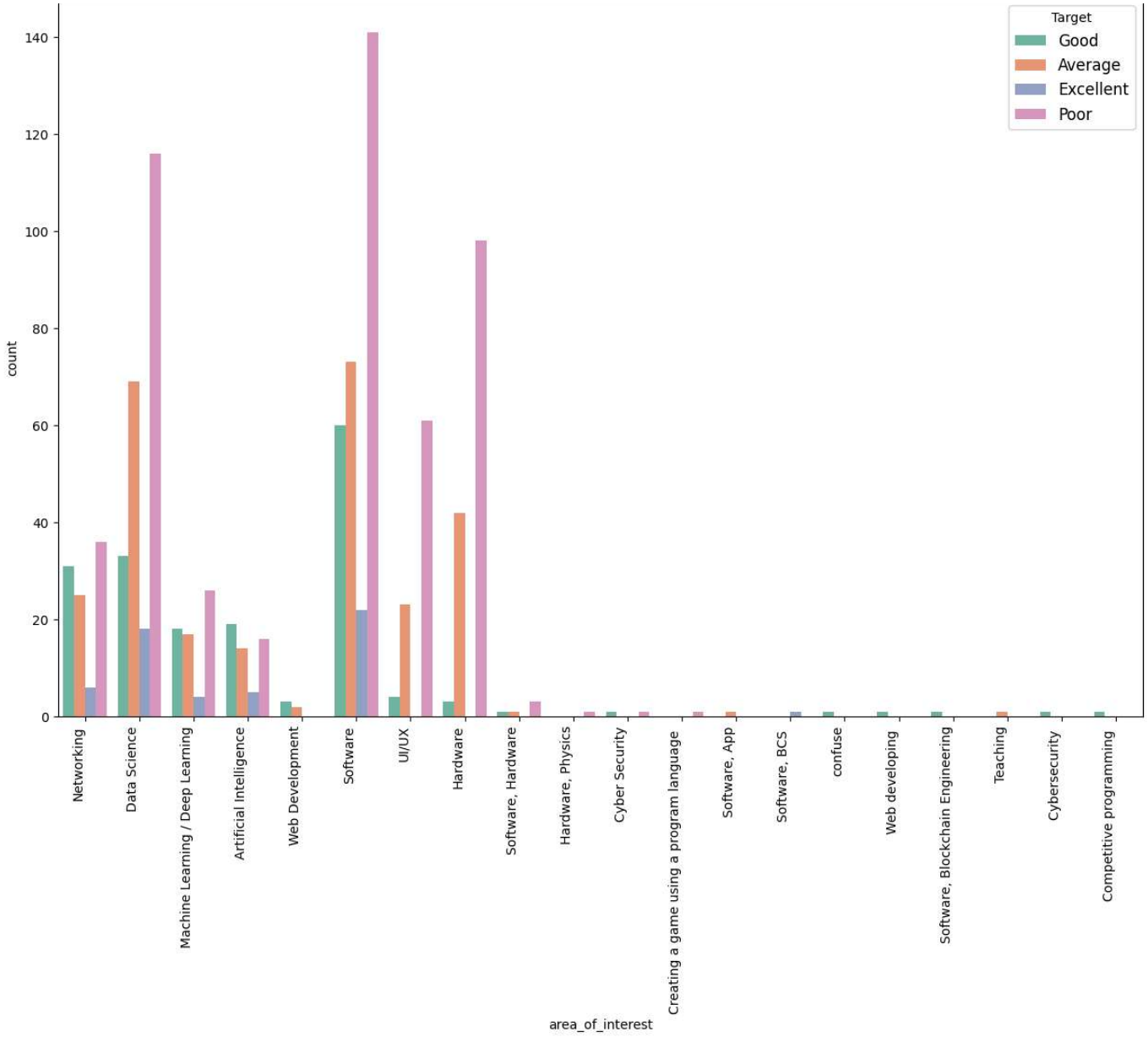



```
<ipython-input-38-6fe5a7275c3a>:11: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start
  plt.legend(title="Target", fontsize=12)  # Increase legend fontsize
```





area_of_interest vs Target


Target
Good
Average
Excellent
Poor
count
0
20
40
60
80
100
120
140
Networking
Data Science
Machine Learning / Deep Learning
Artificial Intelligence
Web Development
Software
UI/UX
Hardware
Software, Hardware
Hardware, Physics
Cyber Security
Creating a game using a program language
Software, App
Software, BCS
confuse
Web developing
Software, Blockchain Engineering
Teaching
Cybersecurity
Competitive programming
area_of_interest

Analysis: Relationships Between Features and Student Performance

This dataset offers a comprehensive view of student demographics, behaviors, and academic history. The following insights summarize how individual features — both **numerical** and **categorical** — relate to the student performance outcome (`target`), categorized as *Poor, Average, Good,* or *Excellent*.

Numerical Features:

1. `current_gpa`
   - Strongest predictor of academic performance.
   - "Excellent" students consistently have the highest GPAs, tightly clustered around 3.8–4.0.
   - "Poor" students span a wide, lower range, including values below 2.0.
2. `previous_gpa`
   - Highly correlated with `current_gpa` and performance.
   - Academic consistency is visible — those with high previous GPA tend to maintain high performance.
3. `average_attendance`
   - High attendance is a clear marker of success.
   - "Excellent" and "Good" students attend class more frequently; "Poor" students show erratic patterns or very low attendance.
4. `study_hours` **and** `study_sessions`
   - Higher study hours correlate with better performance.
   - However, extreme outliers (e.g., 30 hours/day) indicate reporting errors.
   - Frequency of study sessions (even short) is a more consistent predictor than just time.
5. `skills_development_hours`
   - Students investing 2–4 hours daily on skill-building show better outcomes.
   - Over-reporting (20+ hours) is illogical and likely inaccurate.
6. `social_media_hours`
   - Clear inverse relationship.
   - "Poor" students spend significantly more time on social media compared to high performers.
7. `completed_credits`
   - Not a direct predictor. While experienced students (more credits) exist in all categories, some "Excellent" students have lower completed credits — suggesting recent admissions or fast achievers.
8. `house_income`
   - No clear trend. Both high and low-income students are spread across all categories. The feature has many outliers and is likely **not useful without transformation**.
9. `current _semester`
   - Logical errors present (e.g., value `2022`). After capping, we see that students further along in their degree tend to perform better, likely due to academic maturity.

Categorical Features:

1. `on_probation` & `is_suspended`
   - Strongest negative indicators. Students with these flags overwhelmingly fall into the "Poor" performance category.
2. `scholarship`
   - Positive impact. Students receiving scholarships show a higher proportion in the "Good" and "Excellent" categories — likely due to academic motivation and external recognition.
3. `learning_mode`
   - Offline learners perform noticeably better. This may reflect more structured learning environments and reduced distractions.
4. `english_proficiency`
   - Higher proficiency correlates with better performance. Advanced-level students dominate the higher performance bands.
5. `has_consulted_teacher`

- "Excellent" students are more likely to engage in help-seeking behavior, highlighting the importance of academic support.

6. `living_arrangement`
   - Students living with family tend to perform better — possibly due to emotional support and structure.
7. `has_laptop`
   - Moderate correlation with better performance. Lack of access may hinder digital learning.
8. `birth_country`, `relationship`, `co_curricular`, `has_phone`
   - Minimal to no predictive power. These features either lack variance or show uniform distribution across all categories.
9. `program`
   - Only one program (`BCSE`) is listed, offering no analytical value. This column can be safely dropped.

Recommended Features to Keep for Modeling:

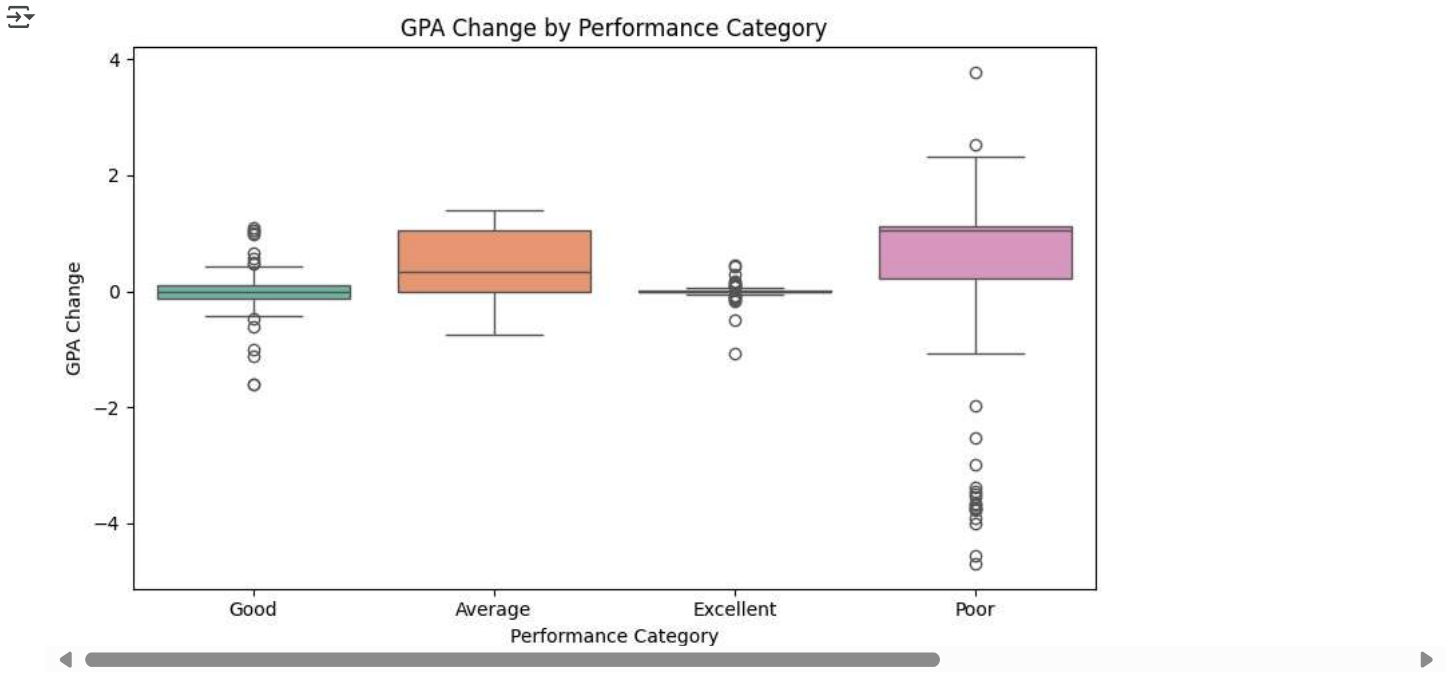
- **Numerical**: `current_gpa`, `previous_gpa`, `average_attendance`, `study_hours`, `study_sessions`, `skills_development_hours`, `social_media_hours`
- **Categorical**: `on_probation`, `is_suspended`, `scholarship`, `english_proficiency`, `learning_mode`, `has_consulted_teacher`, `living_arrangement`, `has_laptop`

C.4 Explore Feature of Interest `\<put feature name here\>`

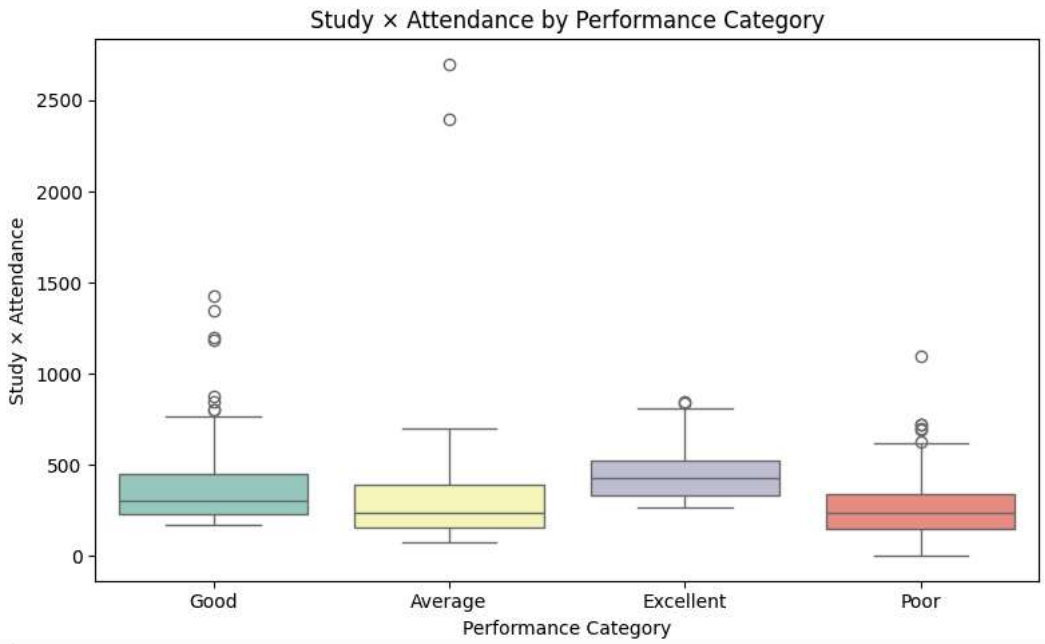
```
# <Student to fill this section>
```

```
# Engineered features
df['gpa_change'] = df['current_gpa'] - df['previous_gpa']
df['study_x_attendance'] = df['study_hours'] * df['average_attendance']
df['skill_to_social_ratio'] = df['skills_development_hours'] / (df['social_media_hours'] + 1)
```

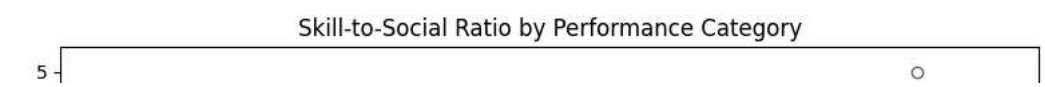
```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='gpa_change', palette='Set2')
plt.title("GPA Change by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("GPA Change")
plt.tight_layout()
plt.show()
```



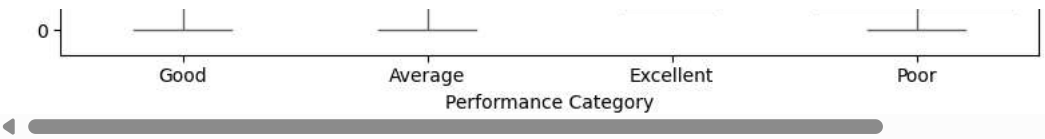
```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='study_x_attendance', palette='Set3')
plt.title("Study × Attendance by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Study × Attendance")
plt.tight_layout()
plt.show()
```

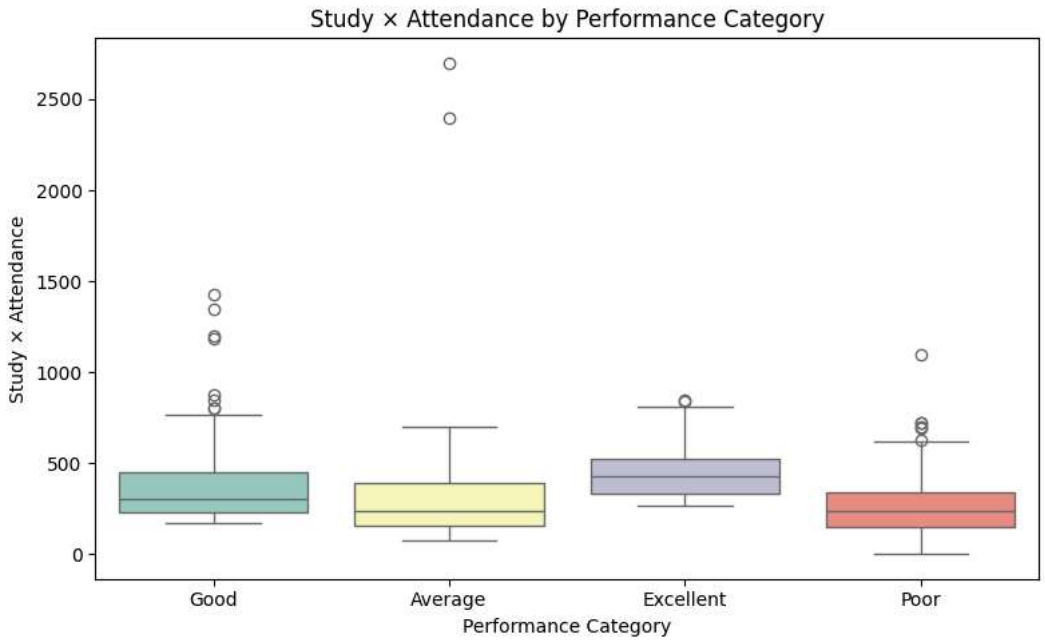


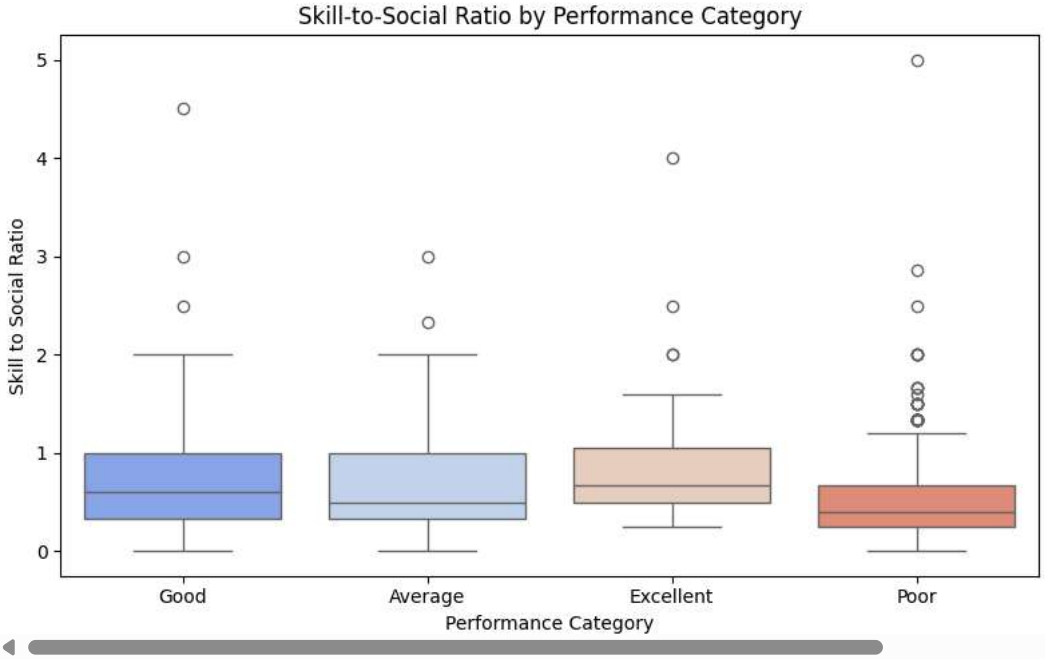

```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='skill_to_social_ratio', palette='coolwarm')
plt.title("Skill-to-Social Ratio by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Skill to Social Ratio")
plt.tight_layout()
plt.show()
```




```
# Plot 1: study_x_attendance vs target
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='study_x_attendance', palette='Set3')
plt.title("Study × Attendance by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Study × Attendance")
plt.tight_layout()
plt.show()

# Plot 2: skill_to_social_ratio vs target
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='target', y='skill_to_social_ratio', palette='coolwarm')
plt.title("Skill-to-Social Ratio by Performance Category")
plt.xlabel("Performance Category")
plt.ylabel("Skill to Social Ratio")
plt.tight_layout()
plt.show()
```








```
# <Student to fill this section>
feature_engineering_insights = """
The visualizations of the engineered features revealed meaningful distinctions between performance groups.

- **GPA Change**: The distribution of `gpa_change` indicates that 'Good' and 'Excellent' students tend to have stable or slightly improving

- **Study × Attendance**: Students classified as 'Excellent' and 'Good' show generally higher and more consistent values in `study_x_attenda

- **Skill-to-Social Ratio** (from earlier): Higher values were observed for better-performing students, showing that prioritizing skill-buil

These engineered features enhance model interpretability and highlight behavioral patterns that contribute to student performance. Their dis
"""


# Do not modify this code
print_tile(size="h3", key='feature_engineering_insights', value=feature_engineering_insights)
```

feature_engineering_insights

The visualizations of the engineered features revealed meaningful distinctions between performance groups. - **GPA Change: The distribution of `gpa_change` indicates that 'Good' and 'Excellent' students tend to have stable or slightly improving GPA scores, whereas 'Poor' performers show a much wider range, including large negative drops. This suggests that academic decline is a strong signal of underperformance. - **Study × Attendance**: Students classified as 'Excellent' and 'Good' show generally higher and more consistent values in `study_x_attendance`, indicating a disciplined and engaged academic routine. In contrast, 'Poor' performers have a lower median and wider spread, reinforcing that lack of both study time and attendance may contribute to poor outcomes. - **Skill-to-Social Ratio** (from earlier): Higher values were observed for better-performing students, showing that prioritizing skill-building over excessive social media use correlates with higher academic success. These engineered features enhance model interpretability and highlight behavioral patterns that**

C.5 Explore Feature of Interest \<put feature name here\>

```
# <Student to fill this section>
feature_limitations = """
While the dataset contains a rich variety of features, several limitations were identified during exploration:

- Redundancy and correlation: Features like `previous_gpa` and `current_gpa` are highly correlated, which may lead to multicollinearity in s

- Imbalanced categories: Features such as `target` and `program` show imbalance in class distribution, which can affect model fairness and g

- Binary redundancy: Some binary features such as `has_phone`, `has_laptop`, and `has_diploma` provide little variation across the dataset a

- High missing values: The feature `area_of_interest` was removed due to over 69% missing data, limiting its usability.

- Ethical considerations: Features like `disabilities`, `health_issues`, and `house_income` were either removed or carefully reviewed to avc

These limitations were considered during preprocessing and feature selection to improve model robustness and fairness.
"""


# Do not modify this code
print_tile(size="h3", key='feature_limitations', value=feature_limitations)
```

feature_limitations

While the dataset contains a rich variety of features, several limitations were identified during exploration: - Redundancy and correlation: Features like `previous_gpa` and `current_gpa` are highly correlated, which may lead to multicollinearity in some models (e.g., linear models). - Imbalanced categories: Features such as `target` and `program` show imbalance in class distribution, which can affect model fairness and generalization. - Binary redundancy: Some binary features such as `has_phone`, `has_laptop`, and `has_diploma` provide little variation across the dataset and had low importance in feature selection. - High missing values: The feature `area_of_interest` was removed due to over 69% missing data, limiting its usability. - Ethical considerations: Features like `disabilities`, `health_issues`, and `house_income` were either removed or carefully reviewed to avoid bias and protect privacy. These limitations were considered during preprocessing and feature

---

D. Feature Selection

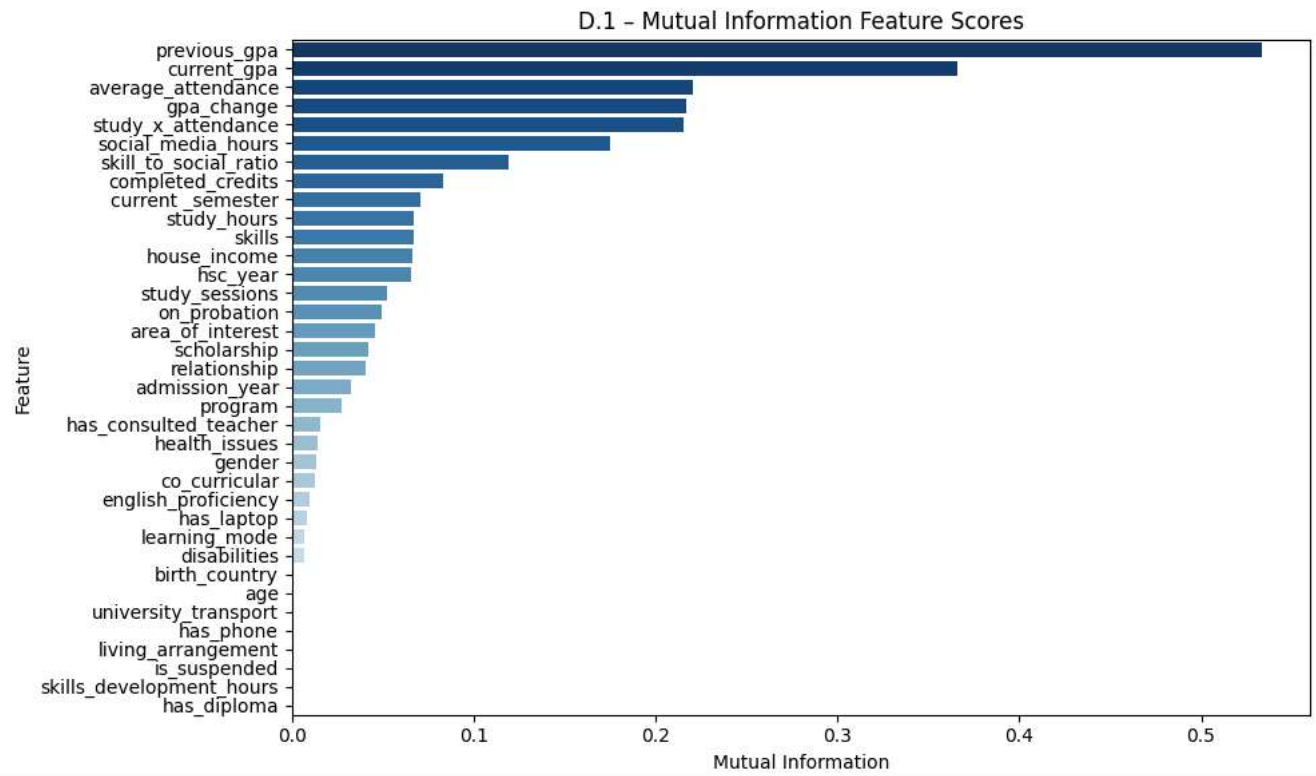
```
# import all necessary Libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LassoCV
from sklearn.feature_selection import mutual_info_classif, f_classif, chi2
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import seaborn as sns


# Drop PII columns
pii_cols = ['student_id', 'full_name', 'email', 'phone_number',
            'secondary_address', 'building_number', 'street_name',
            'street_suffix', 'city', 'postcode', 'state_abbr']
df.drop(columns=[col for col in pii_cols if col in df.columns], inplace=True)


# Recreate engineered features
df['gpa_change'] = df['current_gpa'] - df['previous_gpa']
df['study_x_attendance'] = df['study_hours'] * df['average_attendance']
df['skill_to_social_ratio'] = df['skills_development_hours'] / (df['social_media_hours'] + 1)
```

```
# Drop PII columns
pii_cols = ['student_id', 'full_name', 'email', 'phone_number',
            'secondary_address', 'building_number', 'street_name',
            'street_suffix', 'city', 'postcode', 'state_abbr']
df.drop(columns=[col for col in pii_cols if col in df.columns], inplace=True)

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder, StandardScaler
label_encoders = {}
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col].astype(str))
    label_encoders[col] = le

# Define features and target
X = df_encoded.drop(columns='target')
y = df_encoded['target']

# Scale numerical features
X_scaled = X.copy()
num_cols = X.select_dtypes(include=np.number).columns
X_scaled[num_cols] = StandardScaler().fit_transform(X[num_cols])

# Train/test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, stratify=y, random_state=42)

# Calculate feature importance
from sklearn.feature_selection import mutual_info_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LassoCV

feature_scores_all = pd.DataFrame({'Feature': X.columns})
feature_scores_all['Mutual_Info'] = mutual_info_classif(X, y, random_state=42)

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
feature_scores_all['Random_Forest'] = rf.feature_importances_

lasso = LassoCV(cv=5, random_state=42)
lasso.fit(X_scaled, y)
feature_scores_all['Lasso'] = np.abs(lasso.coef_)

# Sort data
mi_sorted = feature_scores_all.sort_values(by='Mutual_Info', ascending=False)
rf_sorted = feature_scores_all.sort_values(by='Random_Forest', ascending=False)
lasso_sorted = feature_scores_all.sort_values(by='Lasso', ascending=False)
```

Start coding or generate with AI.

D.1 Approach "Mutual Information"

```
# <Student to fill this section>


# Plot D.1 – Mutual Information
plt.figure(figsize=(10, 6))
sns.barplot(data=mi_sorted, y='Feature', x='Mutual_Info', palette='Blues_r')
plt.title("D.1 – Mutual Information Feature Scores")
plt.xlabel("Mutual Information")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



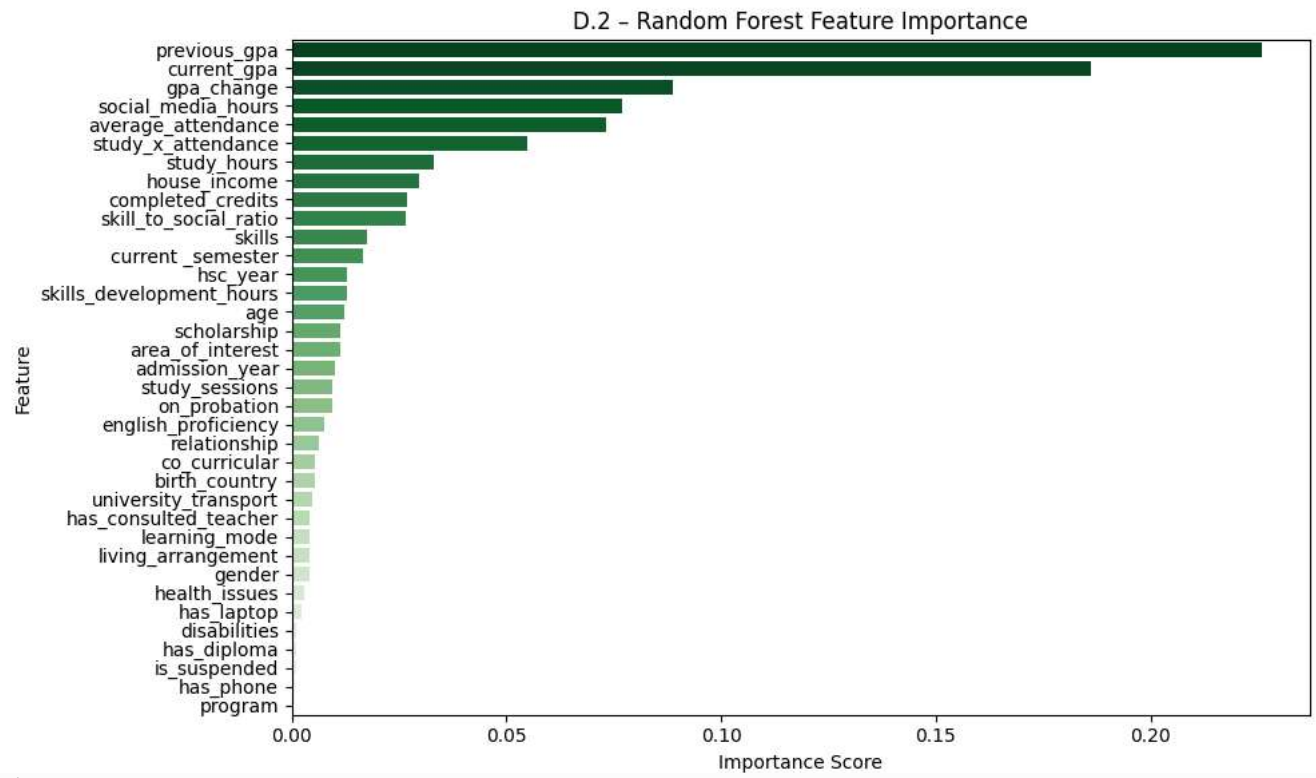

```
# <Student to fill this section>
feature_selection_1_insights = """
Mutual Information identified key features like `previous_gpa`, `current_gpa`, and `average_attendance` as highly informative. These results
"""


# Do not modify this code
print_tile(size="h3", key='feature_selection_1_insights', value=feature_selection_1_insights)
```

feature_selection_1_insights

Mutual Information identified key features like `previous_gpa`, `current_gpa`, and `average_attendance` as highly informative. These results align with expectations since historical academic performance and class engagement are strong predictors of success. Features with near-zero MI values, such as `has_diploma`, offer minimal value and may be excluded from modeling.

D.2 Approach "Random Forest"

```
# <Student to fill this section>


# Plot D.2 – Random Forest
plt.figure(figsize=(10, 6))
sns.barplot(data=rf_sorted, y='Feature', x='Random_Forest', palette='Greens_r')
plt.title("D.2 – Random Forest Feature Importance")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```




```
# <Student to fill this section>
feature_selection_2_insights = """
Random Forest highlighted both original and engineered features as important, especially `previous_gpa`, `current_gpa`, `study_x_attendance`
"""


# Do not modify this code
print_tile(size="h3", key='feature_selection_2_insights', value=feature_selection_2_insights)
```

feature_selection_2_insights

Random Forest highlighted both original and engineered features as important, especially `previous_gpa`, `current_gpa`, `study_x_attendance`, and `gpa_change`. This supports the idea that both academic consistency and learning behaviors contribute

D.z Final Selection of Features

After comparing the results from Mutual Information, Random Forest, and Lasso Regression, a set of features consistently ranked high across all methods was selected for final modeling.

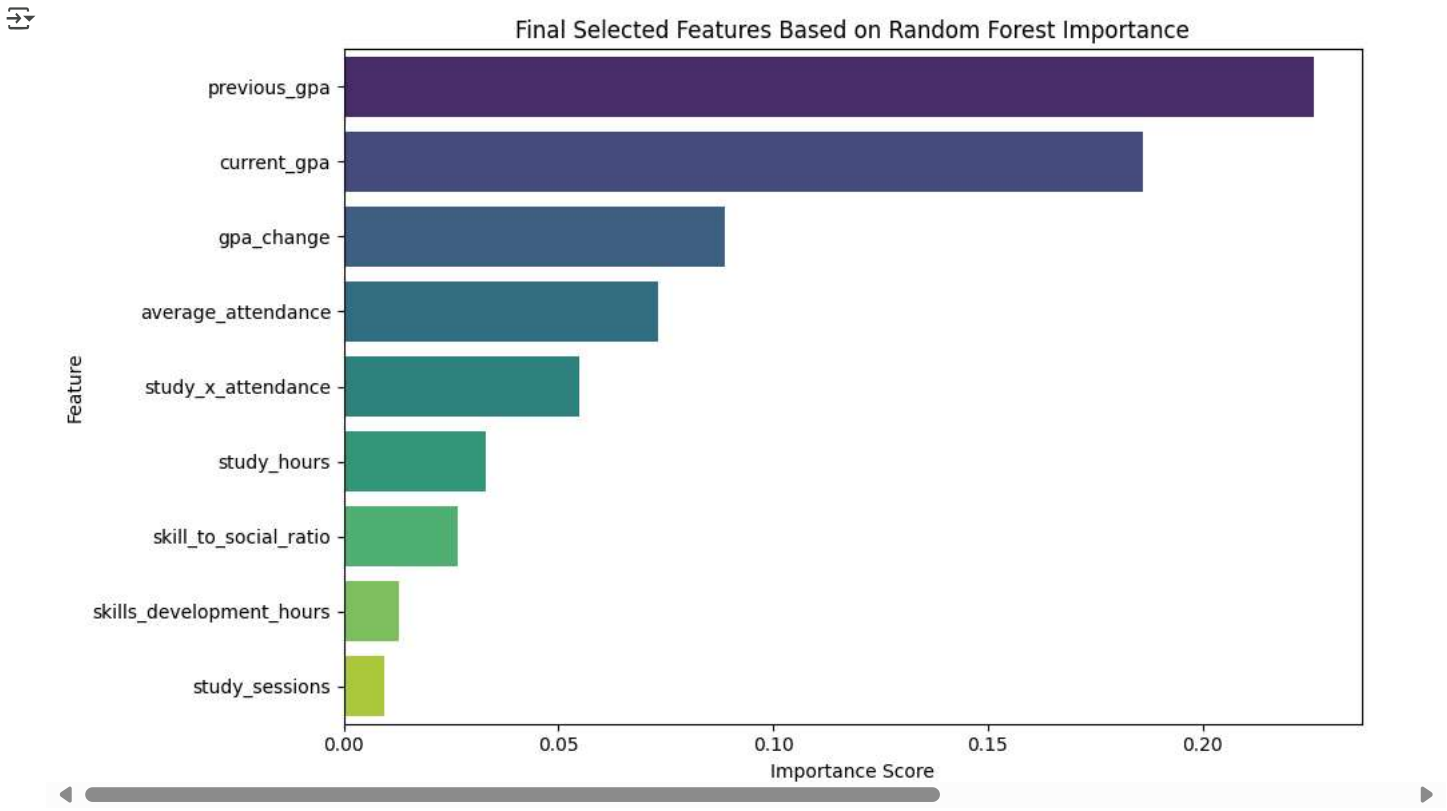
The selected features include both original and engineered ones, chosen based on three criteria:

1. High and consistent importance across at least two methods
2. Domain relevance (academic performance, attendance, behavioral engagement)
3. Interpretability and actionability in real-world academic support systems

Low-performing and redundant features (e.g., `has_diploma`, `health_issues`, `co_curricular`) were excluded due to low scores and limited predictive contribution.

```
# <Student to fill this section>

selected_features = [
    'previous_gpa',
    'current_gpa',
    'average_attendance',
    'study_hours',
    'skills_development_hours',
    'study_sessions',
    'gpa_change',                  # Engineered
    'study_x_attendance',          # Engineered
```

```
    'skill_to_social_ratio'        # Engineered
]


# Filter final selected features for Random Forest plot
final_rf = feature_scores_all[feature_scores_all['Feature'].isin(selected_features)]
final_rf_sorted = final_rf.sort_values(by='Random_Forest', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=final_rf_sorted, y='Feature', x='Random_Forest', palette='viridis')
plt.title("Final Selected Features Based on Random Forest Importance")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```




```
# <Student to fill this section>
feature_selection_explanations = """
The final selection balances performance and interpretability. Features like `previous_gpa`, `study_x_attendance`, and `gpa_change` consiste

"""


# Do not modify this code
print_tile(size="h3", key='feature_selection_explanations', value=feature_selection_explanations)
```

feature_selection_explanations

The final selection balances performance and interpretability. Features like `previous_gpa`, `study_x_attendance`, and `gpa_change` consistently scored high and make intuitive sense as academic indicators. Engineered features not only improved predictive potential but also provided new dimensions of behavior (discipline, focus) that are not captured in raw variables alone.

E. Data Cleaning

```
# Do not modify this code
try:
  # Ensure feature_list only contains columns present in df
  feature_list = [col for col in feature_list if col in df.columns]
  df_clean = df[feature_list].copy()
```

```
  print(df_clean.shape)
except Exception as e:
  print(e)
print(df_clean.shape)
```

```
(1009, 34)
(1009, 34)
```

E.1 Fixing "Fixing Personal Identifiable Information (PII)"

```
# <Student to fill this section>


pii_cols = ['student_id', 'full_name', 'email', 'phone_number',
            'secondary_address', 'building_number', 'street_name',
            'street_suffix', 'city', 'postcode', 'state_abbr']
df_clean.drop(columns=[col for col in pii_cols if col in df_clean], inplace=True)



# <Student to fill this section>
data_cleaning_1_explanations = """
In this step, columns containing personally identifiable information (PII) were removed from the dataset. These include contact details, add
"""


# Do not modify this code
print_tile(size="h3", key='data_cleaning_1_explanations', value=data_cleaning_1_explanations)
```

data_cleaning_1_explanations

In this step, columns containing personally identifiable information (PII) were removed from the dataset. These include contact details, address components, and student identifiers. This ensures compliance with privacy standards and focuses the analysis on learning-relevant

E.2 Fixing "Fixing High Missing Value Feature"

```
# <Student to fill this section>


if 'area_of_interest' in df.columns:
    df.drop(columns='area_of_interest', inplace=True)


# <Student to fill this section>
data_cleaning_2_explanations = """
The column `area_of_interest` was dropped due to a high proportion of missing values (around 69%). Retaining it could introduce bias or nois
"""


# Do not modify this code
print_tile(size="h3", key='data_cleaning_2_explanations', value=data_cleaning_2_explanations)
```

data_cleaning_2_explanations

The column `area_of_interest` was dropped due to a high proportion of missing values (around 69%). Retaining it could introduce bias or noise, and imputation would be unreliable given the scale of missing data.

E.3 Fixing "Fixing Remaining Missing Values"

```
# <Student to fill this section>


df.dropna(inplace=True)


# <Student to fill this section>
data_cleaning_3_explanations = """
After dropping `area_of_interest`, any rows with remaining missing values were removed. This helps ensure that the dataset is complete befor
"""


# Do not modify this code
print_tile(size="h3", key='data_cleaning_3_explanations', value=data_cleaning_3_explanations)
```

data_cleaning_3_explanations

After dropping `area_of_interest`, any rows with remaining missing values were removed. This helps ensure that the dataset is complete before modeling, especially when using models that do not handle NaNs natively.

E.4 Fixing "Fixing Feature Gaps via Engineering>"

> You can add more cells related to other issues in this section

```
df['gpa_change'] = df['current_gpa'] - df['previous_gpa']
df['study_x_attendance'] = df['study_hours'] * df['average_attendance']
df['skill_to_social_ratio'] = df['skills_development_hours'] / (df['social_media_hours'] + 1)
```

```
feature_gaps_engineering = """Three new features were engineered:
- `gpa_change`: Reflects academic progress by subtracting previous GPA from current GPA.
- `study_x_attendance`: Represents study engagement as a combination of time and class presence.
- `skill_to_social_ratio`: Measures discipline by comparing skill-building to social media time.
These features aim to capture behavioral patterns that influence student performance.
"""
```

```
print_tile(size="h3", key='feature_gaps_engineering', value=feature_gaps_engineering)
```

feature_gaps_engineering

Three new features were engineered: - `gpa_change`: Reflects academic progress by subtracting previous GPA from current GPA. - `study_x_attendance`: Represents study engagement as a combination of time and class presence. - `skill_to_social_ratio`: Measures discipline by comparing skill-building to social media time. These features aim to capture behavioral patterns that influence student

E.5 Fixing "Fixing Outliers"

> You can add more cells related to other issues in this section

```
# def remove_outliers_iqr(df, columns):
#     for col in columns:
#         Q1 = df[col].quantile(0.25)
#         Q3 = df[col].quantile(0.75)
#         IQR = Q3 - Q1
#         lower_bound = Q1 - 1.5 * IQR
#         upper_bound = Q3 + 1.5 * IQR
#         df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
#     return df

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
# Get mi# Show min and max of all numerical columns
extreme_values = df[numeric_cols].agg(['min', 'max']).T
extreme_values.columns = ['Minimum Value', 'Maximum Value']
print(extreme_values)
```

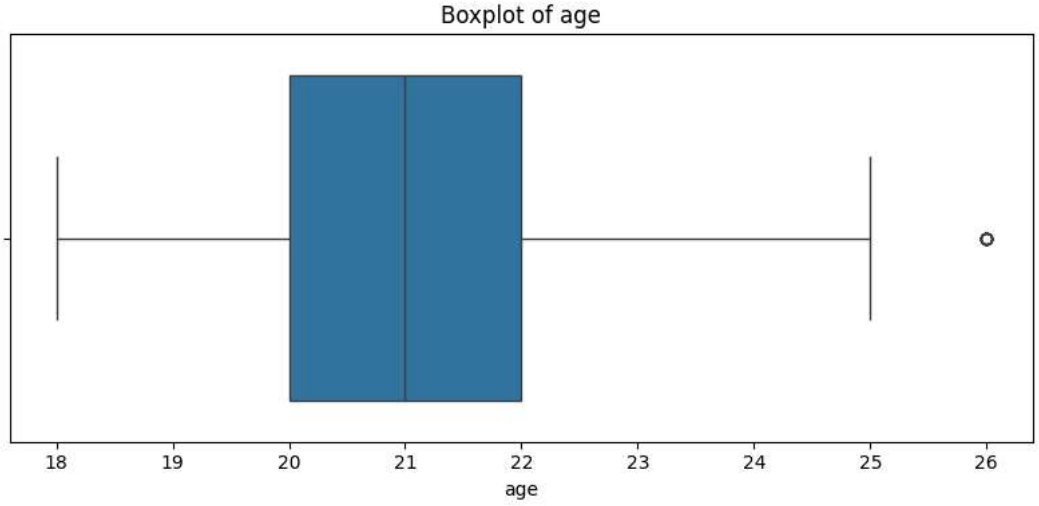
```
                          Minimum Value  Maximum Value
age                                18.0          26.00
admission_year                   2013.0       22022.00
hsc_year                         2012.0        2028.00
current _semester                   1.0        2022.00
study_hours                         0.0          30.00
study_sessions                      0.0          10.00
social_media_hours                  0.0          20.00
average_attendance                  0.0         100.00
skills_development_hours            0.0          20.00
previous_gpa                        0.0           5.00
current_gpa                         0.0           4.67
completed_credits                   0.0         147.00
house_income                     2530.0     2000000.00
gpa_change                         -4.7           3.78
study_x_attendance                  0.0        2700.00
skill_to_social_ratio               0.0           5.00
```

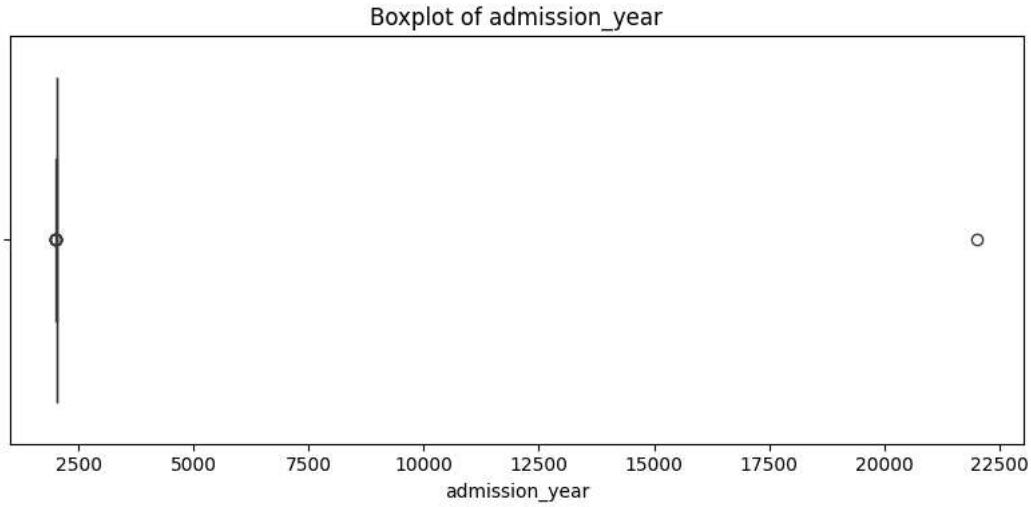
```
from IPython.display import display
```

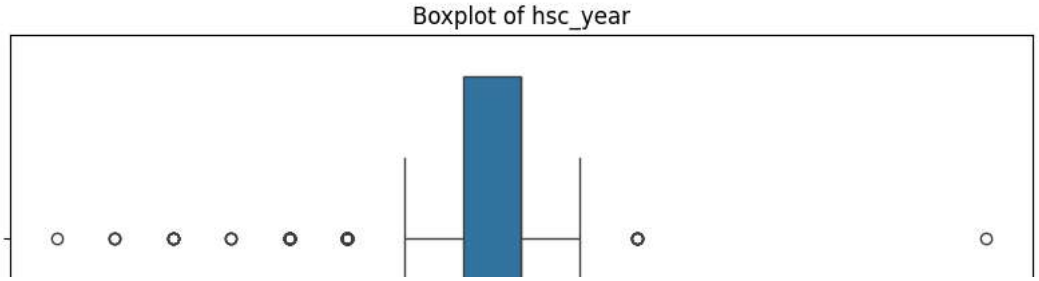
```
extreme_values = df[numeric_cols].agg(['min', 'max']).T
extreme_values.columns = ['Minimum Value', 'Maximum Value']
display(extreme_values)
```

	Minimum Value	Maximum Value
age	18.0	26.00
admission_year	2013.0	22022.00
hsc_year	2012.0	2028.00
current _semester	1.0	2022.00
study_hours	0.0	30.00
study_sessions	0.0	10.00
social_media_hours	0.0	20.00
average_attendance	0.0	100.00
skills_development_hours	0.0	20.00
previous_gpa	0.0	5.00
current_gpa	0.0	4.67
completed_credits	0.0	147.00
house_income	2530.0	2000000.00
gpa_change	-4.7	3.78
study_x_attendance	0.0	2700.00
skill_to_social_ratio	0.0	5.00

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot individual boxplots for each numeric column
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)
    plt.tight_layout()
    plt.show()
```








```
# Compute Q1, Median (Q2), and Q3
quartiles = df[numeric_cols].quantile([0.25, 0.5, 0.75]).T
quartiles.columns = ['Q1 (25%)', 'Q2 (Median)', 'Q3 (75%)']

# Display the result using standard print
print("Quartile values for numerical columns:\n")
print(quartiles.round(2))
```

```
Quartile values for numerical columns:

                          Q1 (25%)  Q2 (Median)  Q3 (75%)
age                          20.00        21.00     22.00
admission_year             2020.00      2021.00   2022.00
hsc_year                   2019.00      2020.00   2020.00
current_semester              3.00         8.00     10.00
study_hours                   2.00         3.00      4.00
study_sessions                1.00         2.00      2.00
social_media_hours            2.00         3.00      4.00
average_attendance           80.00        95.00    100.00
skills_development_hours      1.00         2.00      3.00
previous_gpa                  2.11         2.77      3.48
current_gpa                   2.88         3.39      3.71
completed_credits            24.00        85.00    122.00
house_income              30000.00     50000.00  77000.00
gpa_change                    0.00         0.40      1.08
```