Data Mining and Informatics

By

IFEANYI NNAMDI OKOLI
2348748

Department of Computer Science
with Artificial Intelligence
In partial fulfillment for the award of
Master's Degree (M.Sc.) in
Computer Science

March 12, 2025

Contents

Chapter ONE

Project Tools Used

The project just like any other project require some certain tools for its proper execution. Having carefully examined the tasks ahead, some certain tools of the trade were chosen. Below is a few of the chosen tools (or technologies) and their description.

1. **Data Manipulation Tools**

   (a) Numpy - The full meaning of numpy is numerical python. As such, it is a python package used for numerical computation and manipulation of data. Numpy is free and open source, hence one can use it as wished and can also contribute to it code base, is it is maintained by the open source community. For more, check out the documentation **?**

   (b) Pandas - Pandas, a table or spreadsheet-like data manipulation tool or framework is used to handle and crunch. Pandas effectively handles excel, csv, tsv and others. Pandas looks much like excel but much more than excel. **?**

2. **Visualization Tools**

   (a) Matplotlib - Matplotlib is a visualization library commonly used in python data analytics. It is very easy to learn and use and it is very new-user friendly, being considered the most common very first visualization library for python users. To see the documentation and tutorial on what pandas is and what it is used for, [see][]**?**

   (b) Seaborn - Seaborn is a more sophisticated and advanced visualization library using python. Seaborn was written on top of the matplotlib library. See **?**

3. **Algorithm and Methodology**Givenn that the outcome that will be predicted is a categorical data, the algorithm methodology to be employed is going to be classification algorithm. This implies that the data set would be label encoded to transform categorical features to numeric features, standardized to ensure that all the data points are in the same scale.

Chapter TWO

Dataset

TWO.1 Data Collection & Understanding

The dataset, titled **Student Performance & Behaviour Dataset is a dataset of 5000 records collected from a private learning provider. It contains 5000 records and 23 features. ? of different and diverse students' behaviors and environmental factors that go a long way, directly and or indirectly affecting a students academic performance. And it has key student attributes necessary for exploring patterns, correlations and insights related to academic performance. Below is the data dictionary table**

1. **Data Dictionary**
   (a) **Student_ID: Unique identifier for each student**
   (b) **First_Name: Student's first name**
   (c) **Last_Name: Student's last name**
   (d) **Email: Contact email (can be anonymized)**
   (e) **Gender: Male, Female and Other**
   (f) **Age: The age of the student**
   (g) **Department: Student's department (e.g., CS, Engineering, Business).**
   (h) **Attendance (%): Attendance percentage (0-100**
   (i) **Midterm_Score: Midterm exam score (out of 100).**
   (j) **Final_Score: Final exam score (out of 100).**
   (k) **Assignments_Avg: Average score of all assignments (out of 100).**
   (l) **Quizzes_Avg: Average quiz scores (out of 100).**
   (m) **Participation_Score: Score based on class participation (0-10).**
   (n) **Projects_Score: Project evaluation score (out of 100).**
   (o) **Total_Score: Weighted sum of all grades.**
   (p) **Grade: Letter grade (A, B, C, D, F).**
   (q) **Study_Hours_per_Week: Average study hours per week.**

(r) Extracurricular_Activities: Whether the student participates in extracurriculars (Yes/No).

(s) Internet_Access_at_Home: Does the student have access to the internet at home? (Yes/No).

(t) Parent_Education_Level: Highest education level of parents (None, High School, Bachelor's, Master's, PhD).

(u) Family_Income_Level: Low, Medium, High.

(v) Stress_Level (1-10): Self-reported stress level (1: Low, 10: High).

(w) Sleep_Hours_per_Night: Average hours of sleep per night.

Chapter THREE

Data Preprocessing

In this project, the very first task was to import the libraries that will be used in this project. I prefer to import all my libraries at the beginning and this enables me to debug faster given that I have a single library import cell. Below is the code snippet

THREE .1 Import Libraries

import libraries needed for the task

```
# import needed libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Markdown as mkd
```

After libraries importation (which is iterative until the project is completed) the next step in data analytics is to import and pre process the data. Data pre-processing is a collection of investigative actions performed on data prior to the actual analysis being performed. The first step is to import the dataset into the working environment, mostly but not limited to jupyter notebook. This was done with the python code snippet

THREE .2 Data Preprocessing

load the dataset

```
# load the dataset in the df variable
df = pd.read_csv("datasets/Students_Grading_Dataset.csv")

# convert all features to lowercase
df.columns = [col.lower() for col in df.columns]
```

THREE .3 Data Shape

Data shape is a term used too describe how many rows and columns are in the dataset. Here, I used markdown imported as 'mkd' to print out a statement that shows we have 5000 rows (entries) and 23 columns, features or attributes of each entity in the dataset.See Figure Many.1

THREE .4 Data Head and Tail

Data head an tail, were respectively used to glimpse at the first and last five entries of the dataset. We these, we got our first knowledge of the content of the dataset from the top and the bottom. See data head Figure Many.2 and see data tail Figure Many.3

THREE .5 Null Values

Data, being sourced from many possible sources is mainly always noisy containing both empty values, unwanted and unexpected values introduced by both human errors in the case of human data entry and system glitches when the data are sourced from sensors and other automated devices. For this reason, I checked to see if there was any null value or invalid entry. Total number of null values Figure Many.4 and duplicates. Null values distribution Figure Many.5

THREE .6 Basic Descriptive Statistics

This is what I use to first check if there is an outlier in the dataset. This works only for numeric feature and does not work for categorical features. Here is the the formulat 1.5 * IQR. Any number outside of this range, in both positive and negative directions (+ or - signed) is considered an outlier. This is followed by a boxplot to confirm the outlier or disprove it. This is as shown in igure ??

THREE .7 Inconsistent Grading System

From the results below, grading inconsistency is discovered A 78.89 final_score and 66.13 total_score were graded F on line 5 while on line 4997, a 644.21 final_score and 54.25 total_score were graded as A. No attem is made yet to investigate why it existed given the task at hand and time. A fix is provided, using a function to calculate grades based on final_score and created a new feature called grade_corrected. Inconsistent grading Figure ??

```
# These are the functions used in the task

# function to fix grading inconsistent issue.
"""
Using;
↪  https://www.mastersportal.com/articles/2290/how-to-convert-uk-grades-f

Converting British grades into American grades
In the US, the grading system uses letters A-F (without E) to
↪  evaluate students. D is the minimum passing grade.

+70% = A
60-69% = B
50-59% = C
40-49% = D
Below 40% = F (fail)
"""
def grader(total_score):
"""
This function is used to compute grades for the students.
This also applies a known, global and generally accepted
↪  grading system.
The input is the total score.
"""
"""Grades : A, B, C, D, E, F"""
if isinstance(total_score, int) or isinstance(total_score,
↪  float):
if total_score < 40:
return 'F'
# elif total_score <= 50:
#     return 'E'
elif total_score <= 49:
return 'D'
elif total_score <= 59:
return 'C'
elif total_score <= 69:
return 'B'
else:
return 'A'
else:
raise "Wrong value type. Int or float required"



# apply the the to create the grade_corrected feature
stud_record['grade_corrected'] =
↪  stud_record['total_score'].apply(grader)
```

```
# show grading inconsistency once more,
# investigate accurate fix
stud_record[['final_score', 'total_score', 'grade',
 ↪  'grade_corrected']].head()
```

The Figure ?? is the sample of the grade_corrected feature just created

Chapter FOUR

Exploratory Data Analysis (EDA)

Exploratory Data Analysis, EDA is the exploration of the data to discover patterns and trends prior to training a model.

FOUR.1 Univariate Data Analysis

. Univariate analysis is the simplest form of analyzing data. “Uni” means “one”, so in other words, your data has just one variable. It does not deal with causes or relationshiops (unlike what regression analysis does) and its major purpose is to describe, it takes data and summarizes that data and finds trends and patterns in them.

FOUR.1.1 Gender Distribution

Checking the dataset, running the code to check for number of each gender

```
# students gender distribution
gender_distr = stud_record.groupby(by='gender').size()

gender_distr.plot(kind='bar')
plt.title('Student Gender Distribution', fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel('Student Gender', fontdict={'fontweight':'bold'})
plt.ylabel('Gender Count', fontdict={'fontweight':'bold'})
plt.ylim(2400, 2600)
plt.grid()

# show gender distribution figures
gender_distr
```

This generated figure Many.7

FOUR.1.2 Student Age Distribution

Here student age distribution was investigate. This was done to ascertain that the student population is of the expected student age and not of elderly people or underaged. From the result got, it is proved that the student are of student ages ranging from 18 years old to 24 years old.

It is also observed that the ages are closely distributed within the range of 682 for 18 years of age (the minimum) to 24 years old (the maximum age), the highest age. The investigation code is as shown below:

```
# student age distribution
age_dist = stud_record.groupby(by = 'age').size()
age_dist.plot(kind='bar')
plt.ylim(600, 800)
plt.grid()
plt.title("Student Age Distribution", fontdict={'size':14,
 ↪  'fontweight':'bold'})
plt.xlabel('Student Age', fontdict={'fontweight':'bold'})
plt.ylabel("Student Age Count", fontdict={'fontweight':'bold'})


# pd.DataFrame([age_dist.values, age_dist.index], columns=['Numbers',
 ↪  'values'])
age_dist.index
pd.DataFrame({'Age Group': age_dist.index, 'Age Group Count':
 ↪  age_dist.values})
```

As pictorized in Figure Many.11

FOUR.1.3 Department Distribution

Let find out what departments / fields of study that are represented in our student population. Not just what departments or fields that are represented but also to find what population of our students are in these departments. The code snippet below was used for the investigation

```
# group dataset by department and get size of each department
dept_dist = stud_record.groupby(by='department').size()
dept_dist.plot(kind='bar')
plt.title('Department Distribution', fontdict={'size':14,
 ↪  'fontweight':'bold'})
plt.xlabel('Department', fontdict={'fontweight':'bold'})
plt.ylabel('Department Count', fontdict={'fontweight':'bold'})
plt.grid()
# plt.show()

pd.DataFrame({'Department':dept_dist.index, "No. of Students":
 ↪  dept_dist.values})
```

In figure, Figure Many.9 shows the department and number of students in them.

FOUR.1.4 Attendance Distribution [First 100]

The rate of attendance of students, in an attempt to identify what could be related to student performance / grade. The code below was used. As can be seen in the chart, the output is much like a timeseries result. Given that this is outside the scope of this task, further investigation was not carried.

```
# plot student attendance
stud_record['attendance (%)'].plot()
plt.title("Attendance Plot [First 100]", fontdict={'size':14, 'fontweight':'bold'}
plt.xlabel("Student Attendance", fontdict={'fontweight':'bold'})
plt.ylabel("Student Attendance (%)", fontdict={'fontweight':'bold'})
plt.xlim(1, 100)
plt.show()
```

Assignment avg, Quizzes avg and study hour per week shared the same pattern and the same thing could be said about them. Moving on

FOUR.1.5 Internet Home Acess Distribution

The internet has had a profound and far-reaching effect on students' studies, revolutionising the way they access information, interact with educational content, and collaborate with others. ?. The use of internet, in short while has affected all aspects of human with human education at the center of it all. Hence, the search to see patterns how internet access at home could affect or influence student performance. The code snippet is as below:

```
# dataset grouping by internet_access_at_home and getting the sizes
internet_access_at_home =
↪  stud_record.groupby(by='internet_access_at_home').size()

internet_access_at_home.plot(kind='bar')
plt.title('Student Internet Home Access Distribution',
↪  fontdict={'size':14, 'fontweight':'bold'})
plt.xlabel('Home Internet Accesss', fontdict={'fontweight':'bold'})
plt.ylabel('Home Internet Access Count',
↪  fontdict={'fontweight':'bold'})
# plt.ylim(2400, 2600)
plt.grid()
```

According to Figure Many.10, most of the students have internet access at home as suspected.

FOUR.1.6 Student Family Income Level Distribution

Let find out the income level of the parents of the students in the dataset. From Figure ??, it is evedent that most, almost of the students are from Low to Medium Income Earners, maybe workers and not business owners.

FOUR.1.7 Student Stress Level Distribution

Stress, personally seen as a by-product of hardwork affects humans in all endeavour. Students are not going to be of any exception. Hence let check / investigate the student calibrated stress level from the dataset. The code snippet is :

```
# group dataset by stress_level(1-10) and get size of each stress
↪  level
```

```
stress_level = stud_record.groupby(by='stress_level (1-10)').size()
stress_level = pd.DataFrame(stress_level)
stress_level.plot(kind='bar')
plt.title('Student Stress Level Distribution', fontdict={'size':14,
↪   'fontweight':'bold'})
plt.xlabel('Student Stress Level', fontdict={'fontweight':'bold'})
plt.ylabel('Student Stress Level Count',
↪   fontdict={'fontweight':'bold'})
plt.ylim(400)
plt.grid()
plt.show();
```

This is brought to live by Figure ??

FOUR.1.8 Parent Education Level

This is yet another factor that is expected to have great effect on students academic work success rate. Among those salient factors are parent's occupation, educational attainment, socioeconomic status, family composition, parental involvement, peer and teacher influence, and adolescent self-efficacy ?. I looked at the parent education level of students using the python code snippet below:

```
# dataset grouped by paretn_education_level and the different sizes
↪   obtained
parent_education =
↪   stud_record.groupby(by='parent_education_level').size()
parent_education.plot(kind='bar')
plt.title("Parent Education Level", fontdict={'fontweight':'bold',
↪   'fontsize':14})
plt.xlabel('Parent Education Level', fontdict = {'fontweight':'bold',
↪   'fontsize':10})
plt.ylabel('Count', fontdict={'fontweight':'bold', 'fontsize':10})
plt.grid()

plt.show();
```

And this generated the Figure Many.12

FOUR.1.9 Extra Curricular Activities Dist

It is a popular saying that all work and no play makes Johny a dull boy. Students are expected to engage in other extra curricular activites like games and sports to name just a few. Unfortunately, it is clear that more than half of the students do not engage in extra curricular activities. This could point to something. Who knows? The snippet for the data investigation is below :

```
# data grouped by extra curricular activities of student. Size of yes
↪   / no extracted
extracurricular =
↪   stud_record.groupby(by='extracurricular_activities').size()
```

```
extracurricular.plot(kind='bar')
plt.title('Extrac Curricular Activities Dist',
↪  fontdict={'fontweight':'bold', 'size':14})
plt.xlabel('Extrac Curricular Activities',
↪  fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Extra Curricular Count', fontdict={'fontweight':'bold',
↪  'size':10})
plt.grid()
plt.show()
```

And the accompanying figure is Figure Many.13

FOUR.1.10 Student Grade Distribution

Here, let see what grades are available. It is clear that students are graded by A, B, C, D and F. But from the chart, students only scored A, B, and C. No D or F. Impressive! The code snippet used is below:

```
# group dataset by grade_corrected and get size of each grade
grade = stud_record.groupby(by='grade_corrected').size()
grade.plot(kind='bar')
plt.title("Student Grades", fontdict={'fontweight':'bold',
↪  'size':14})
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Grade Count', fontdict={'fontweight':'bold', 'size':10})
plt.grid()

plt.show()
```

The code snippet above generated the visualisation : Figure Many.14

FOUR.2 Bivariate Data Analysis

This is the analysis of features in association with other features and the essence is to find how features relate to each other. This helps in predictability of one feature based on another feature. involves looking at two variables at a time. Bivariate EDA can help you understand the relationship between two variables and identify any patterns that might exist ?

FOUR.2.1 Gender Grade Distribution

Wondering if the ability if the ability to score high or low can be gender based? I assure you that you are not alone in that. Let see what our data has to say if that ability is or can be gender based. The investigation was done with the python code snippet below

```
# group dataset by grade_corrected and gender
gender_grade = stud_record.groupby(by=['grade_corrected',
↪  'gender']).count()['student_id'].reset_index()
```

```
gender_grade

valuez = np.array(gender_grade['student_id'])
rows = gender_grade['grade_corrected'].unique() # rows to be plotted
num_rows = int(len(gender_grade['grade_corrected'].unique())) # num
↪  of rows for reshaping
num_cols = int(len(gender_grade) / len(rows))   # num of cols for
↪  reshaping

data = pd.DataFrame(valuez.reshape((num_rows, num_cols)),
index=pd.Index(list(rows), name='grade'),
columns=pd.Index(['female','male'], name=''))
data = data.reset_index()
data
```

The gender grade distribution figure / number is as in Figure Many.15 and show in bar chart in Figure Many.16 From the results got from gender grade distribution, good grades or academic success might not depend on gender. As any one, female or male has the potential to put in good work and succeed. Other things would really affect students's success rate but not their gender. The different grades, A, B and C has balanced distribution of the number of female and male who obtained such grades.

FOUR.2.2 Grade By Department Distribution

Now considering grade by department distribution, one could ask if it is possible that the course of study could help boost students' success. Well, we have the and just have to ask the data to have, on the least an insight. The python code snippet :

```
# investigation of grade-department relationship
department_grade = stud_record.groupby(by = ['grade_corrected',
↪  'department']).size().reset_index()
department_grade

rows = department_grade['grade_corrected'].unique() # rows to be
↪  plotted
num_rows = int(len(department_grade['grade_corrected'].unique())) #
↪  num of rows for reshaping
num_cols = int(len(department_grade) / len(rows))   # num of cols for
↪  reshaping

data = pd.DataFrame(department_grade[0].values.reshape(num_rows,
↪  num_cols),
index=pd.Index(list(rows), name='grade'),
columns = department_grade['department'].unique()
)
data = data.reset_index()

grades = tuple(data['grade'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper right', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Department Grade Distrution',
↪  fontdict={'fontweight':'bold', 'size':10})
plt.title('Department Grade Distribution',
↪  fontdict={'fontweight':'bold', 'size':14})

plt.show()
```

is telling us that department / course of study could help student's performance. Yes I have that believe too. But then it would not depend on the course of study only as other factors would come in to play. Factors like:

- **Passion : As passion is a huge factor in success rate, students studying programs they are passionate about can, affect their rate of success in those fields. This could be encouraged by the trending computing fields in the social media and everyday living.**

- **Resource Availability: This is how easy it is for students to access materials relating to their course of study. It could be from the physical or digital library or the internet and even from peers.**

- **Social Trend : What is trending at the time. It is common knowledge that everyone at the moment is talking about Machine Learning, Data Science, AI and Deep Learning. All these fields are the children of computer. Hence their trending could make computer science, the leading department is in good grades much more interesting to study.**

According to the data, Computer Science (CS) is a leading programme in academic performance with Computer Science outperforming others in A, B or C grades with greater margin in A grade.

FOUR.2.3 Attendance Grade Distribution

Now moving forward investigating into whether attendance to classes has positive effect on academic performance. Checking if, as is popularly said, that punctuality is the soul of business. According to Figure Many.17 minimum attendance to classes is required for academic excellence

FOUR.2.4 Assignment Score - Grade Distribution

Investigating the relationship between Assignment score with final grade obtained, there seem to be an inverse relationship. Most peaple who scored 'A' got low in assignment while most people who got 'C' scored high in assignment. Further investigation into this matter is really needed to draw out more meaning insight. These, are outside the scope of this task. The code snippet used is :

```
# Assignments Avg Score Distribution was classified as Very low, Low,
↪   Medium, Good, Very good
stud_record_updated['assignment_classified'] =
↪   pd.cut(np.array(list(stud_record_updated['assignments_avg'])),
↪   len(labels), labels = labels)
assignment_avg = stud_record_updated.groupby(by =
↪   ['assignment_classified', 'grade_corrected'],
↪   observed=False).size().reset_index()


# stud_record_updated['assignment_avg_classified'] =
↪   pd.cut(np.array(list(stud_record_updated['assignments_avg'])),
↪   len(labels), labels = labels)
# assignment_avg = stud_record_updated.groupby(by =
↪   ['assignment_avg_classified', 'grade_corrected'],
↪   observed=False).size().reset_index()

rows = assignment_avg['grade_corrected'].unique()
num_rows = int(len(assignment_avg['grade_corrected'].unique())) # num
↪   of rows for reshaping
num_cols = int(len(assignment_avg) / len(rows))    # num of cols for
↪   reshaping

data = pd.DataFrame(assignment_avg[0].values.reshape(num_rows,
↪   num_cols),
index=pd.Index(list(rows), name='grade_corrected'),
columns = assignment_avg['assignment_classified'].unique()
)

data = data.reset_index()
grades = tuple(data['grade_corrected'].values)
```

```
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper left', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Assignment Score Count', fontdict={'fontweight':'bold',
↪   'size':10})
plt.title('Assignment Score Distribution',
↪   fontdict={'fontweight':'bold', 'size':14})

plt.show();
```

Chapter FIVE

Modeling

Modelling is where models are trained to using existing data to be able to predict outcomes given unseen data. The steps are as follows:

FIVE.1 Pre-Modeling

First it is not all the features of a dataset that would be used. Redundant dataset are removed or dropped. the include student_id, first_name, email and grade (the previous faulty grade). Below is the data used to achieve this:

```
# list of unwanted and redundant features
unwanted = ['student_id', 'first_name', 'last_name', 'email',
↪  'grade']

# drop unwanted features and create new variable -
↪  stud_recode_choice_features
stud_record_choice_features = stud_record.drop(unwanted, axis=1)
stud_record_choice_features.columns
```

FIVE.2 Modeling

For the modeling, the first thing that was done is the creation of two functions for categorical data encoding and data scaling.

FIVE.2.1 Methodology

The methodology used for the modeling is classification. Classification (multilabel classification) was used because the value to be predicted are not numbers rather categories like A, B and C as the case is.

FIVE.2.2 Justification of Methodology

Classification was used because as mentioned above output feature is of categorical nature and that classes

FIVE.2.3 Categorical Data Encoding

Given that the computer understands only digits, all categorical data (data in classes like female and male, young and old to name a few) has to be coded and represented as numbers to enable the computer understaand them better and to also work faster.

FIVE.2.4 Data Scaling

Also, this is reduction of numerical values to be between 0 and 1. It helps to control bias and maybe overfitting.

FIVE.2.5 Dependent and Independent Features

The data was separated into dependent and independent feature(s). Dependent feature is the feature that would be predicted. It is also called the outcome and is conventionally represented as "y". The independent features are the other features used to predict the outcome. To understand better in a simple way, see it as dependent feature depends on the other features as they are used to predict the dependent feature based on mathematical relationship(s) seen in the data. While independent features are not predicted or dependent on any feature. They are like the base.

FIVE.2.6 Data Split

Here data is spit into train, test dataset. And also from test dataset I split it further to have validation dataset. This is to have data, 70% of the entire dataset to train the model with, and the remaining dataset being 30% was further dividing into 80% for train and the last 20% for validation. These was done using the python code snippet below:

```
# split data into train test set
X_features_train, X_features_test, y_feature_train, y_feature_test =
↪  train_test_split(X_features, y_feature, train_size=0.7,
↪  random_state=42)

# create validation test set
X_features_test, X_features_valid, y_feature_test, y_feature_valid =
↪  train_test_split(X_features_test, y_feature_test, train_size=0.8,
↪  random_state=42)
```

FIVE.2.7 Data Encoding And Data Scaling

The functions earlier created are used in the code snippet below to label encode (using LabelEncoder class from sklearn.preprocessing) and encoding was also done using StandardScaler (from sklearn.preprocessing)

```
# encode and scale parameters
X_train =
↪  scaler(encoder(X_features_train.select_dtypes(include='object')))
```

```
X_test =
↪  scaler(encoder(X_features_test.select_dtypes(include='object')))
X_valid =
↪  scaler(encoder(X_features_valid.select_dtypes(include='object')))

le = LabelEncoder()
y_train = le.fit_transform(y_feature_train)
y_test = le.fit_transform(y_feature_test)
y_valid = le.fit_transform(y_feature_valid)
```

FIVE.2.8 Model Instantiation

First, sklearn packages for modeling are imported here as can be see from the code snippet:

```
# import modelling packages
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
```

and then a dictionary of different classification models were created as in the code snippet below. Next, an iteraction is made through the elements of the dictionary to instantiate each of the models, train the model, and make predictions with the models in turn and compute the accuracy metrics.

```
# building baseline model
models = {
        'xgb' : XGBClassifier,
        'rdf' : RandomForestClassifier,
        'dtc' : DecisionTreeClassifier,
}

for model in models:
        print(f"Model: {model}")
        clf = models[model]()
        clf.fit(X_train, y_feature_train)
        y_pred = clf.predict(X_test)
        print(f"Accuracy: {accuracy_score(y_feature_test, y_pred)}")
        print(classification_report(y_test, y_pred))
        cm = confusion_matrix(y_test, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cm,
        ↪  display_labels=clf.classes_)
        disp.plot()
        plt.show()
```

The best peforming model was found to be the decision tree model and no need for hyper parameter tuning as it was good enough for the task at ahand.

Chapter Many

Recommendation and Conclusion

It has been quite an enjoyed data mining journey from the beginning up to this point. Haven taken this data mining journing thus far, the following the following recommendation(s) and conclusion are here by made:

Recommendation I recommend that :

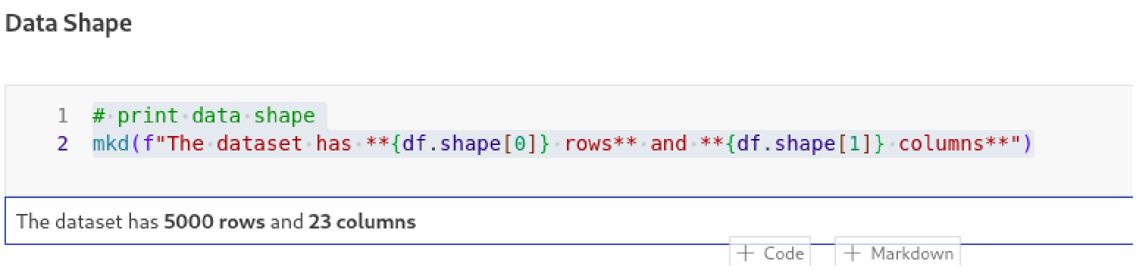
- **Data should be collect about how passionat the students are about their programme / course of study.**
- **Data should be collected about how easy the students feel their courses.**
- **Data should be collect about how much the students love their teacher and how much they feel the teacher loves and cares for their well-being in and outside of the classroom**
- **Data about how much the academic works related to students' inherent talents should also be collected and wrangled.**

Conclusion

- **From the trends and or patterns in the data, student grade or performance does not depend on his or her gender.**
- **Computer Science (CS) is best performming department possibly because for more than two decades, Data Science, Machine Learning, Artificial Intelligence have been trending and getting people's attension. This and availability of learning resourse on youtube, stackoverflow and others are some of the reasons the departent of Computer Science is the best performing department.**
- **Students whose parents are Bachelor's Degree holder performed best followed by the students whose parents are Ph.D holders.**
- **In all grades, students whose parents are high income earners took the lead.**

Chapter Many

Visualizations

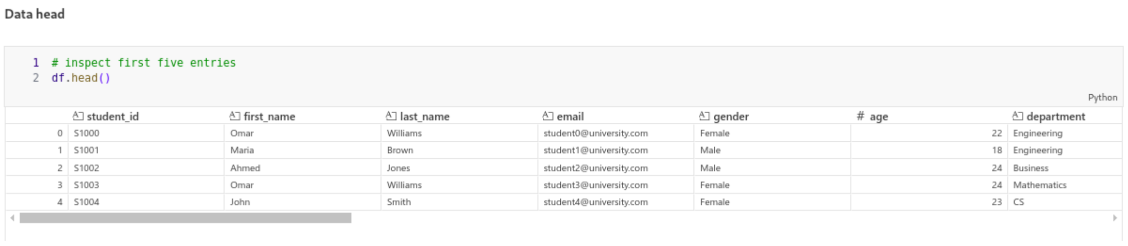


Data Shape

```
# print data shape
mkd(f"The dataset has **{df.shape[0]} rows** and **{df.shape[1]} columns**")
```

The dataset has **5000 rows** and **23 columns**

Figure Many.1: Dataset shape

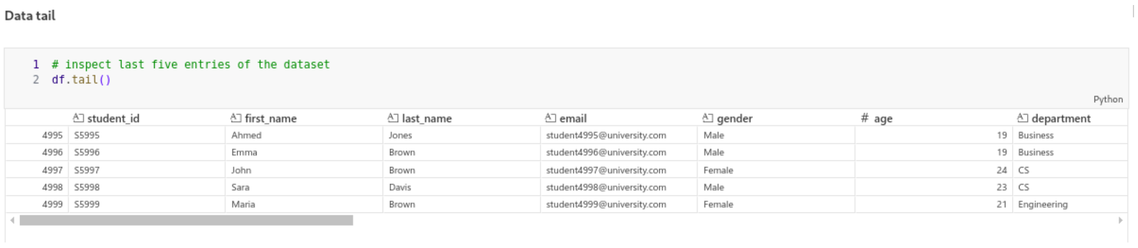


Data head

```
# inspect first five entries
df.head()
```

	student_id	first_name	last_name	email	gender	age	department
0	S1000	Omar	Williams	student0@university.com	Female	22	Engineering
1	S1001	Maria	Brown	student1@university.com	Male	18	Engineering
2	S1002	Ahmed	Jones	student2@university.com	Male	24	Business
3	S1003	Omar	Williams	student3@university.com	Female	24	Mathematics
4	S1004	John	Smith	student4@university.com	Female	23	CS

Figure Many.2: Dataset Head

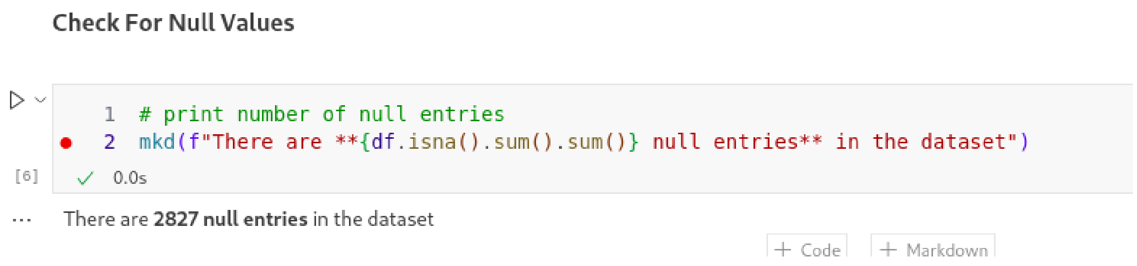


Data tail

```
# inspect last five entries of the dataset
df.tail()
```

	student_id	first_name	last_name	email	gender	age	department
4995	S5995	Ahmed	Jones	student4995@university.com	Male	19	Business
4996	S5996	Emma	Brown	student4996@university.com	Male	19	Business
4997	S5997	John	Brown	student4997@university.com	Female	24	CS
4998	S5998	Sara	Davis	student4998@university.com	Male	23	CS
4999	S5999	Maria	Brown	student4999@university.com	Female	21	Engineering

Figure Many.3: Dataset Tail

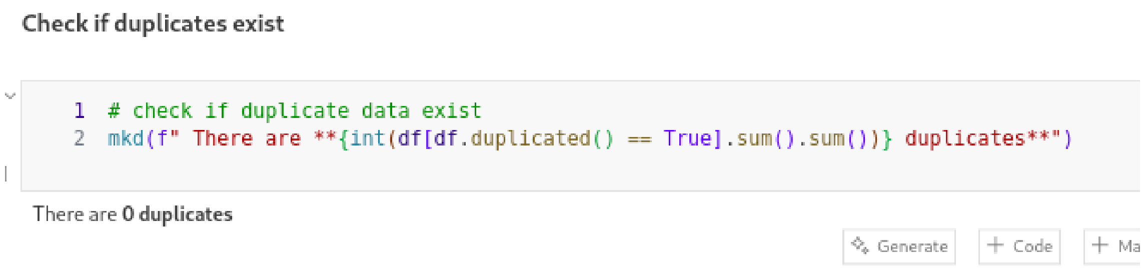


Check For Null Values

```
# print number of null entries
mkd(f"There are **{df.isna().sum().sum()} null entries** in the dataset")
```

There are **2827 null entries** in the dataset

Figure Many.4: Total number of null values



Check if duplicates exist

```
# check if duplicate data exist
mkd(f" There are **{int(df[df.duplicated() == True].sum().sum())} duplicates**")
```

There are **0 duplicates**

Figure Many.6: The number of duplicate values

```
gender
Female    2449
Male      2551
dtype: int64
```

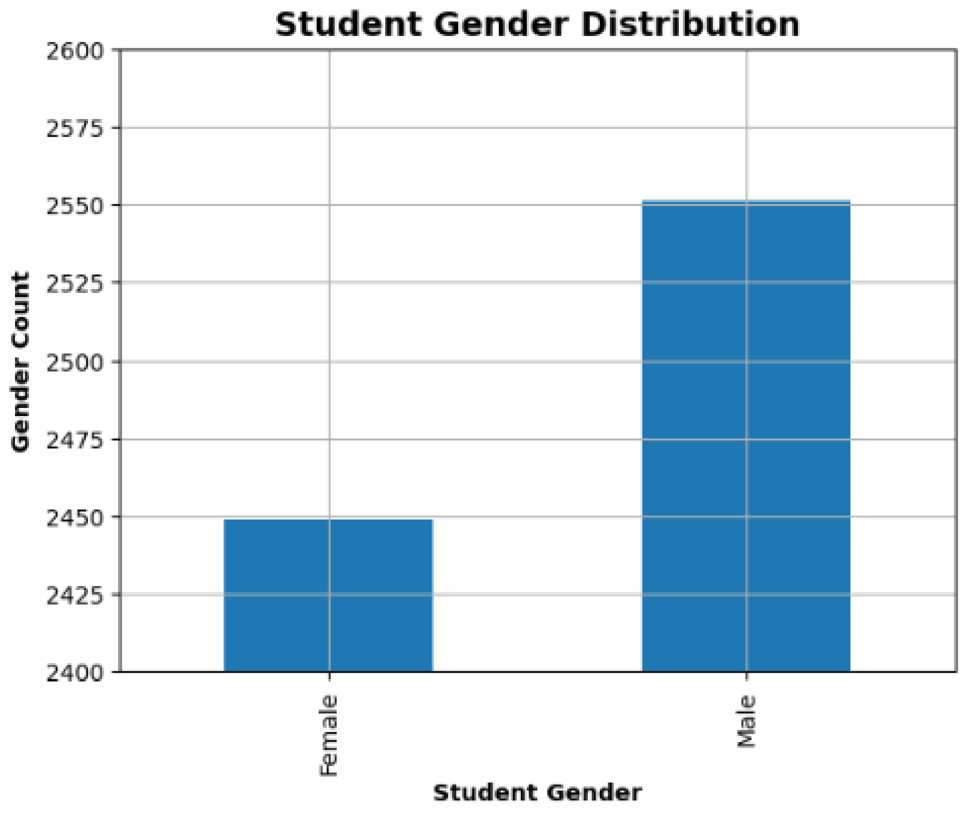
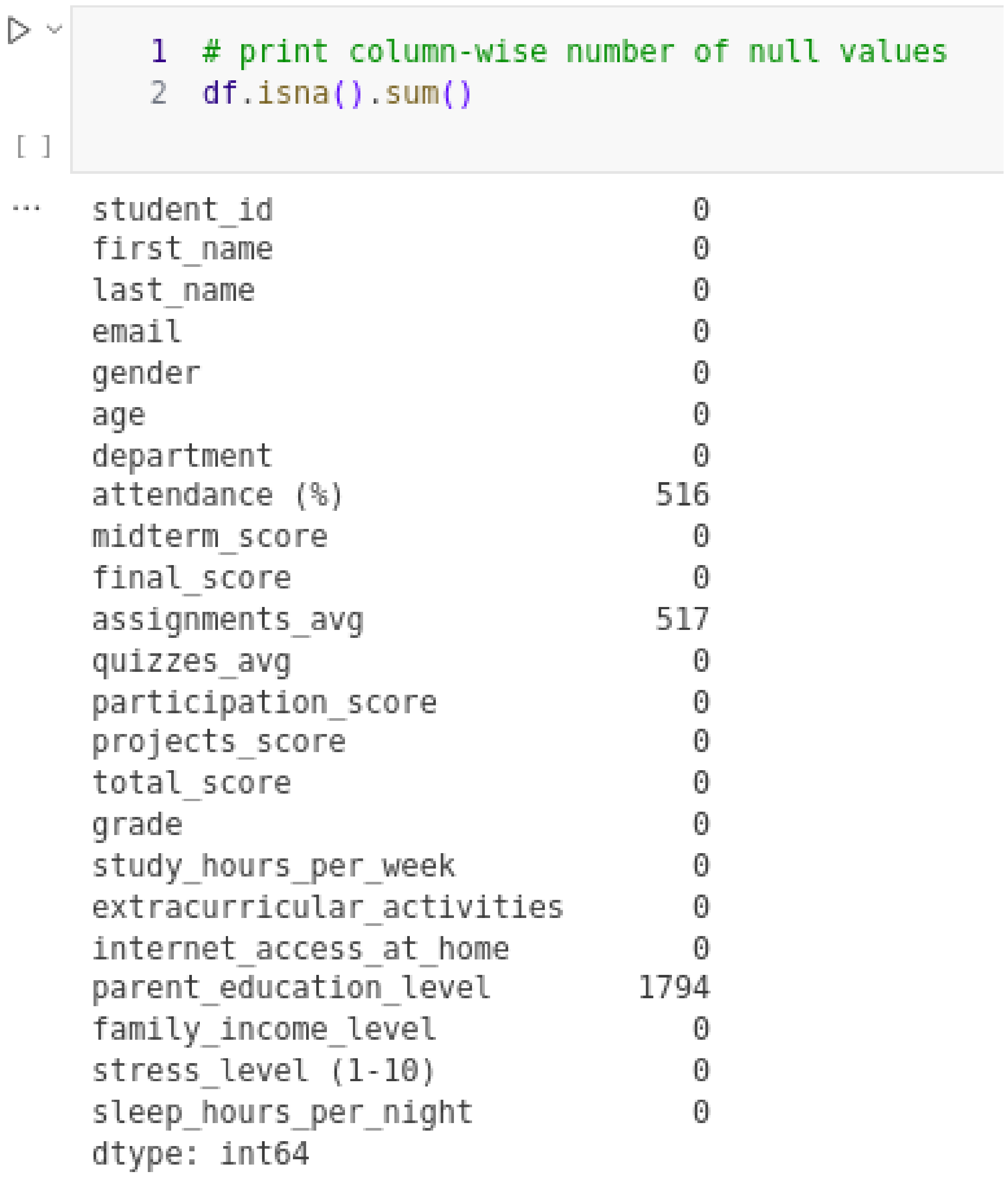


Figure Many.7: Inconsistent Grades

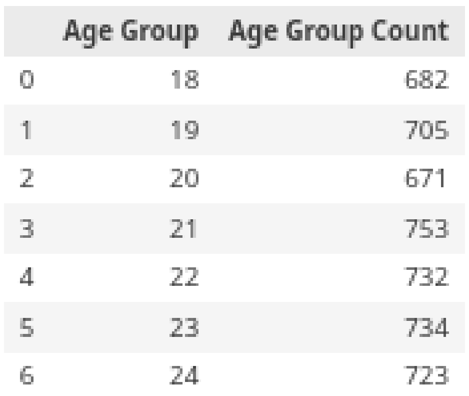
Show Null Value Features



```
# print column-wise number of null values
df.isna().sum()
```

```
student_id                    0
first_name                    0
last_name                     0
email                         0
gender                        0
age                           0
department                    0
attendance (%)              516
midterm_score                 0
final_score                   0
assignments_avg             517
quizzes_avg                   0
participation_score           0
projects_score                0
total_score                   0
grade                         0
study_hours_per_week          0
extracurricular_activities    0
internet_access_at_home       0
parent_education_level     1794
family_income_level           0
stress_level (1-10)           0
sleep_hours_per_night         0
dtype: int64
```

Figure Many.5: Null values distribution by features



	Age Group	Age Group Count
0	18	682
1	19	705
2	20	671
3	21	753
4	22	732
5	23	734
6	24	723

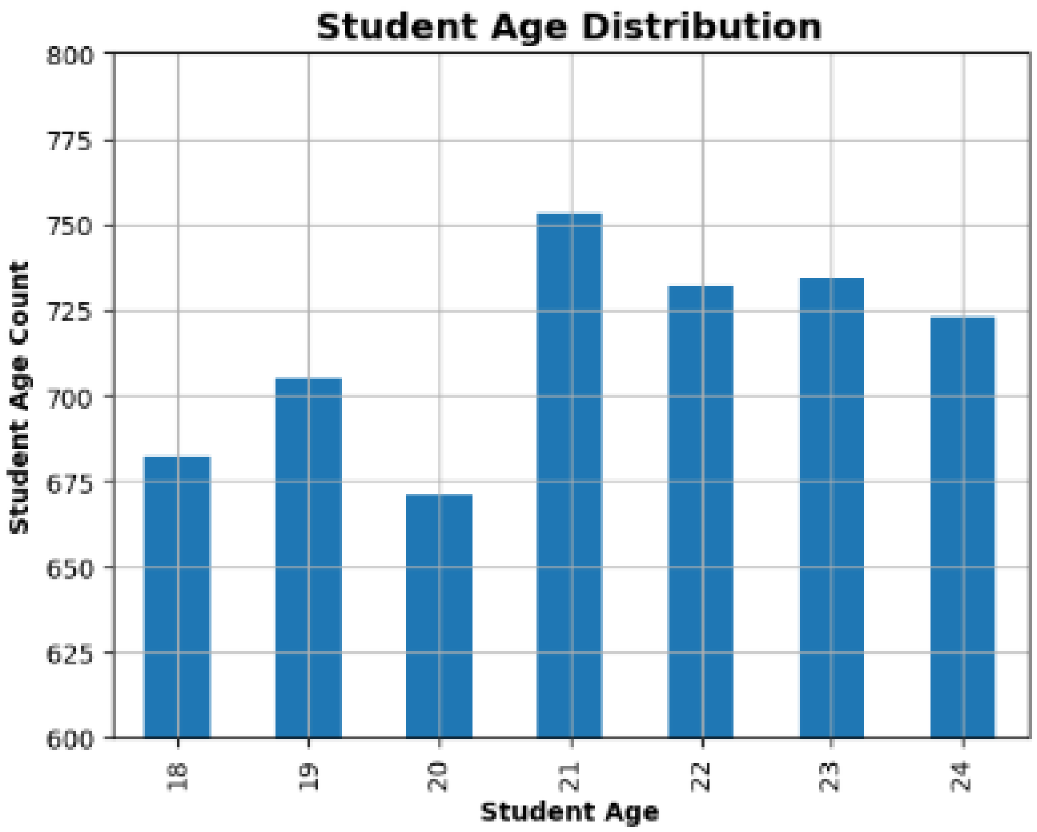
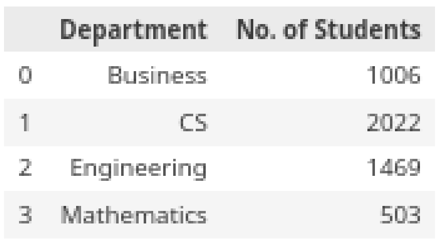


Figure Many.8: Student Age Distribution



	Department	No. of Students
0	Business	1006
1	CS	2022
2	Engineering	1469
3	Mathematics	503

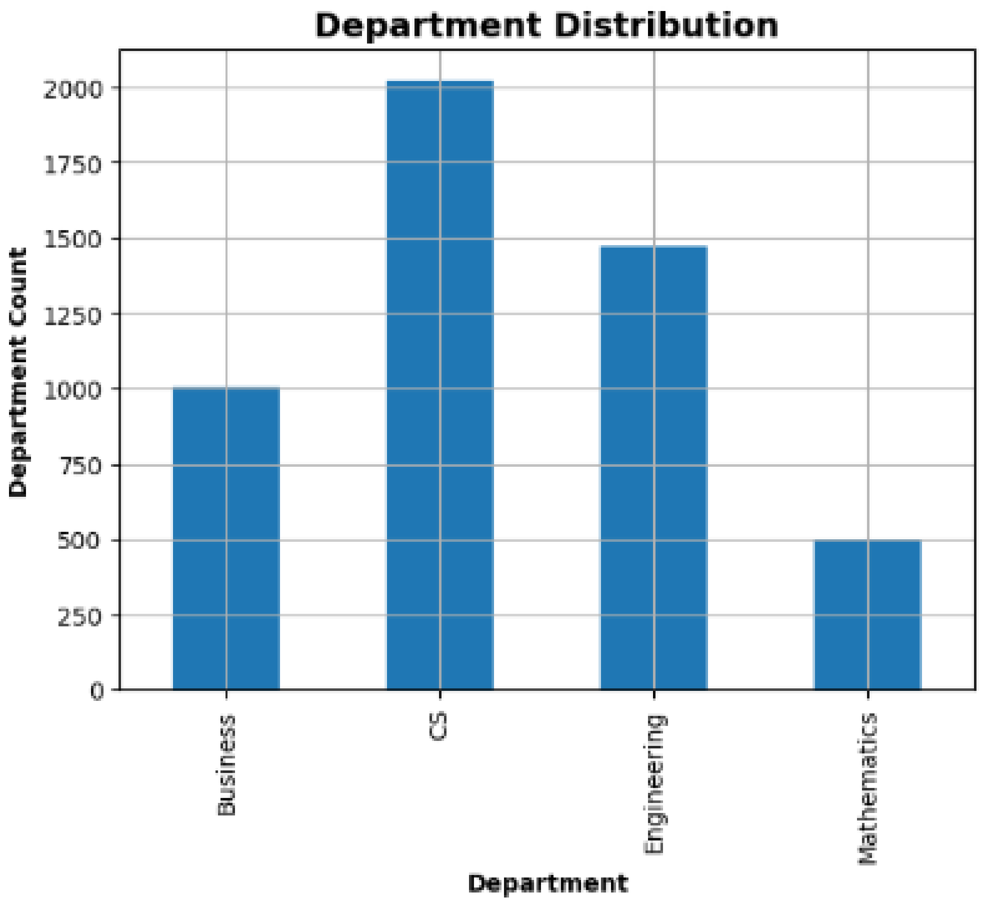


Figure Many.9: Student Department Distribution

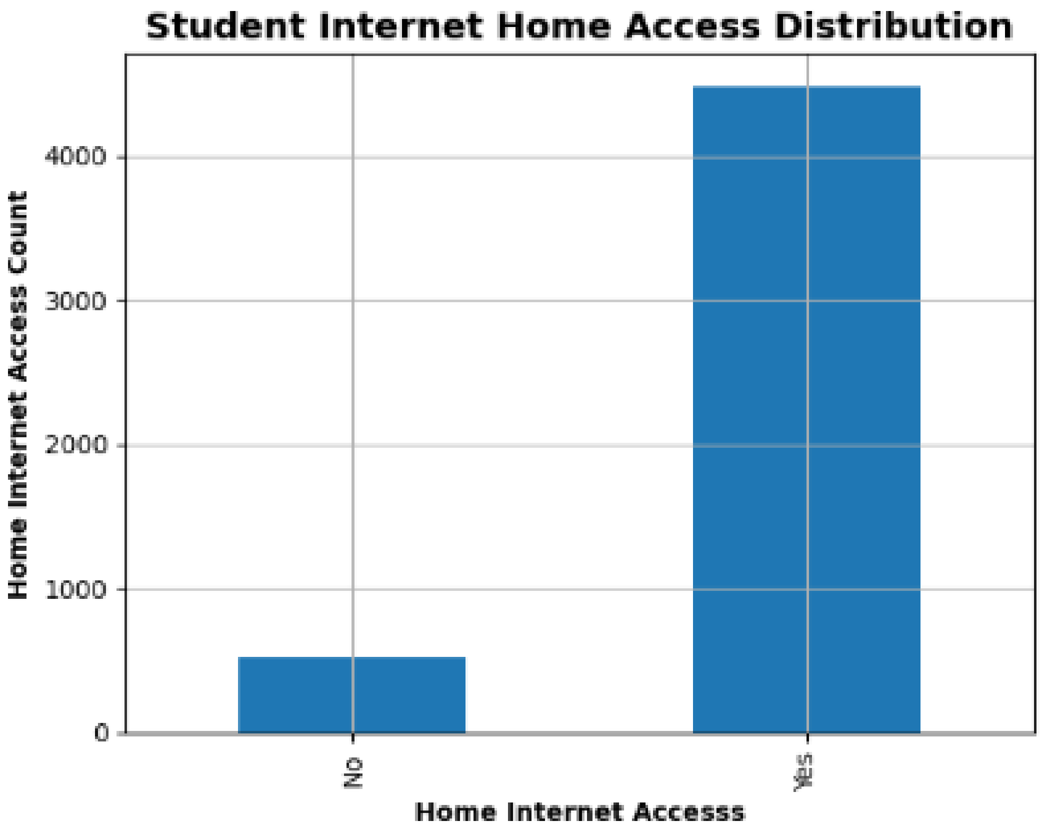


Figure Many.10: Student Home Internet Access Distribution

	Age Group	Age Group Count
0	18	682
1	19	705
2	20	671
3	21	753
4	22	732
5	23	734
6	24	723

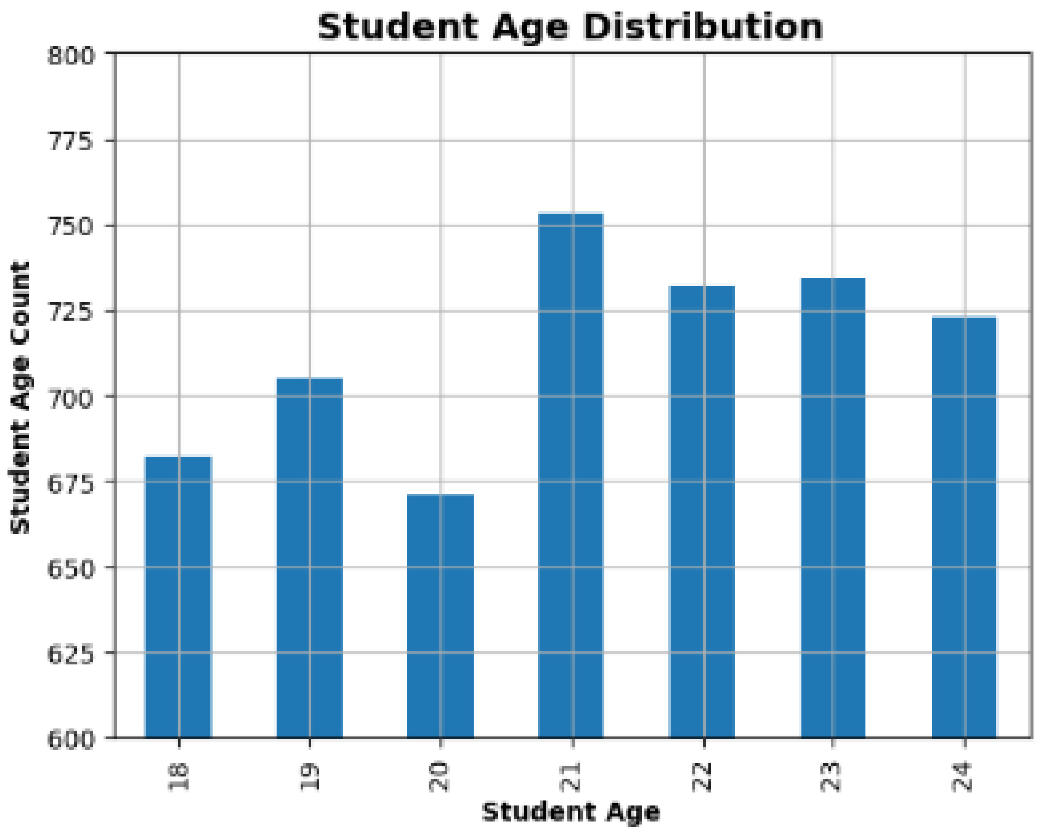


Figure Many.11: Student Age Distribution

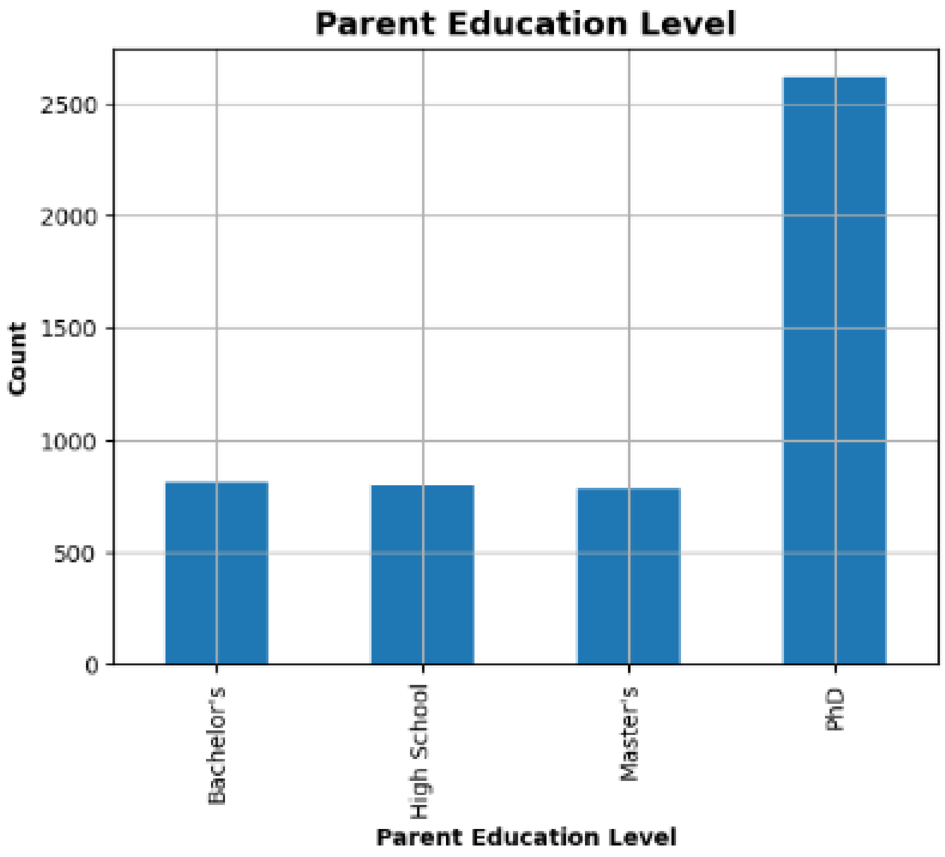


Figure Many.12: Student Parent Education Level Distribution

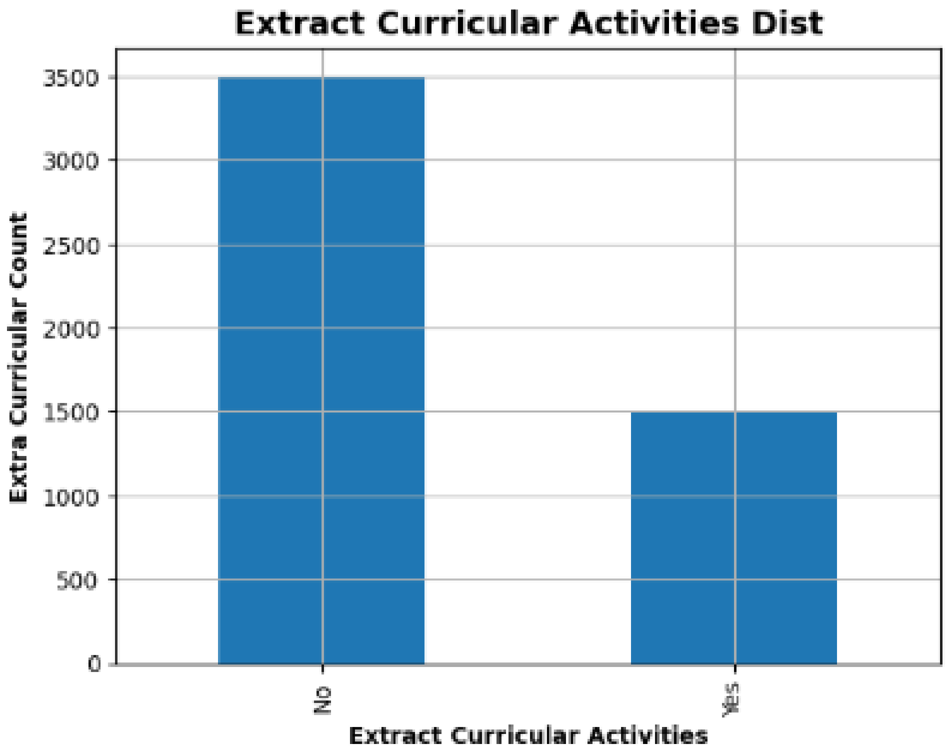


Figure Many.13: Student Extra Curricular Distribution

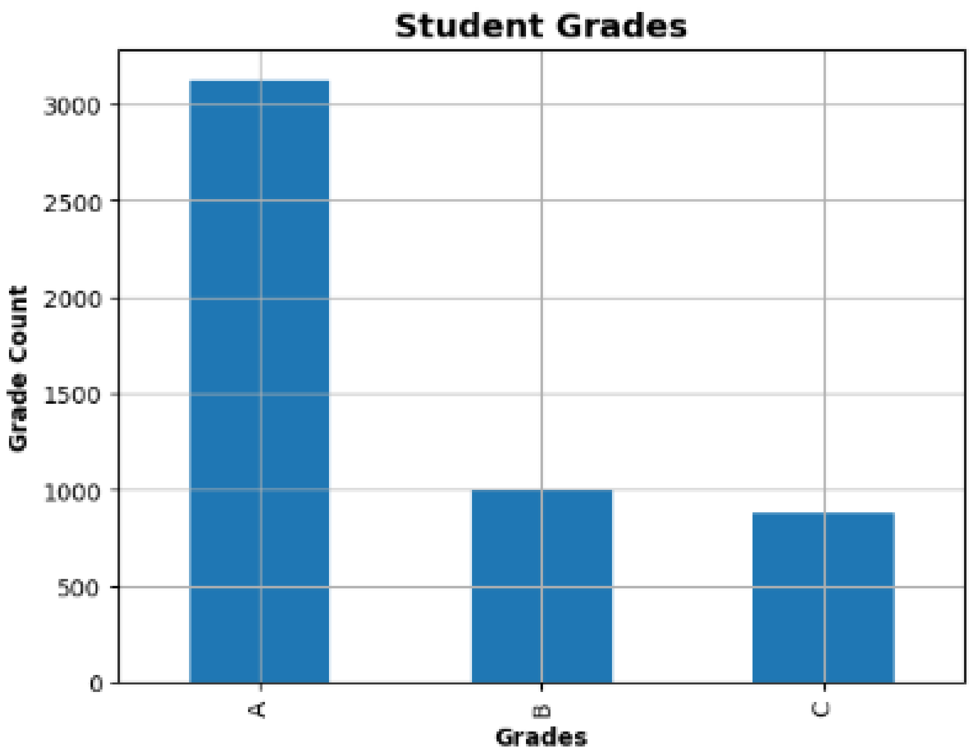
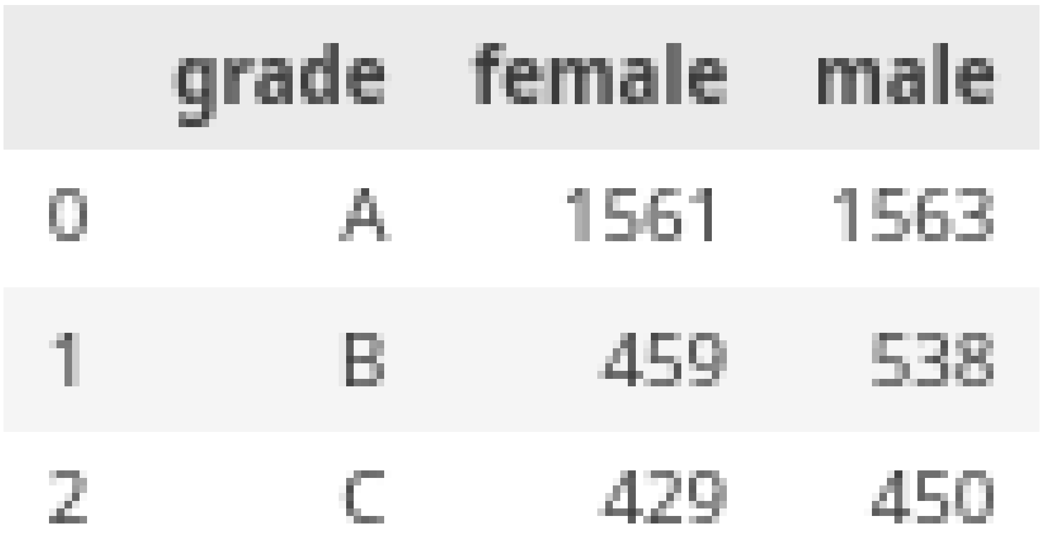


Figure Many.14: Student Extra Curricular Distribution



	grade	female	male
0	A	1561	1563
1	B	459	538
2	C	429	450

Figure Many.15: Student Extra Curricular Distribution

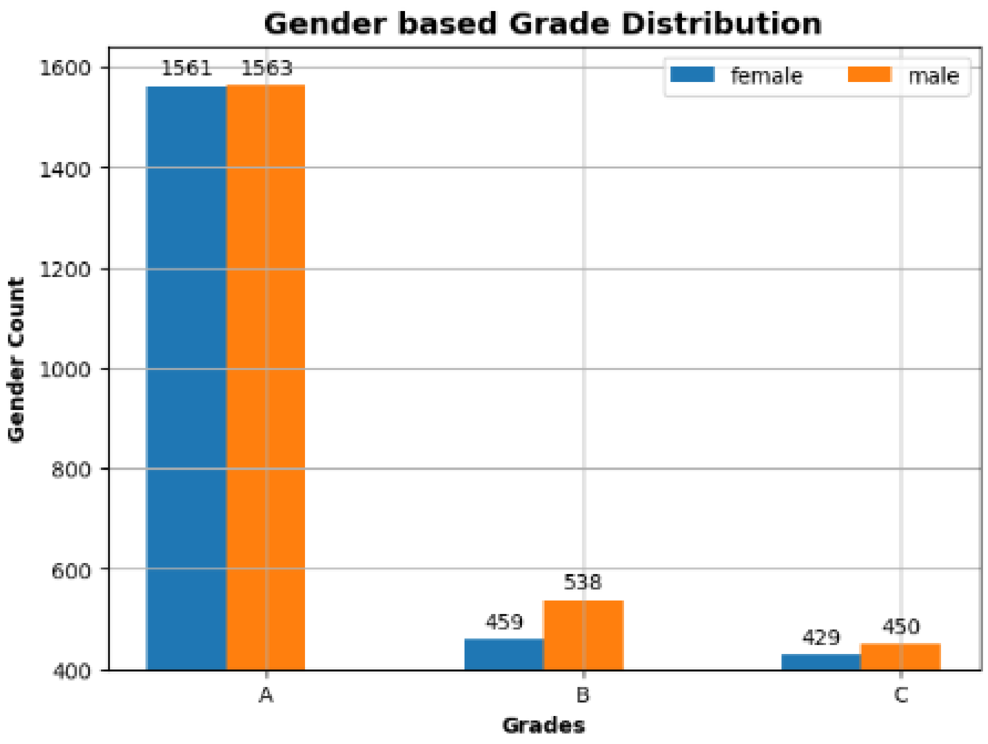


Figure Many.16: Student Extra Curricular Distribution

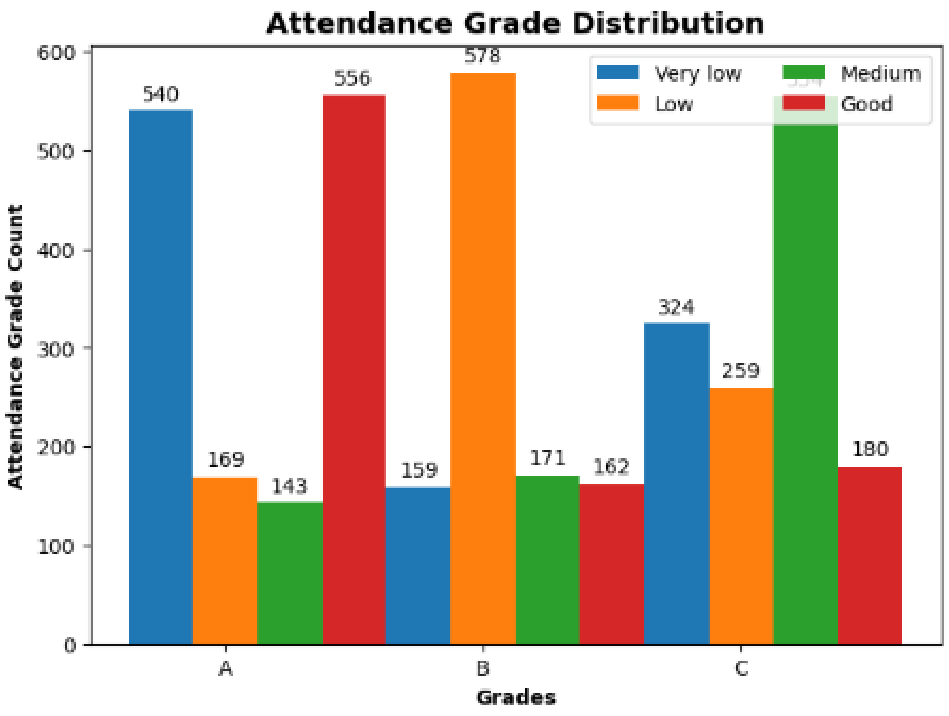


Figure Many.17: Student Extra Curricular Distribution

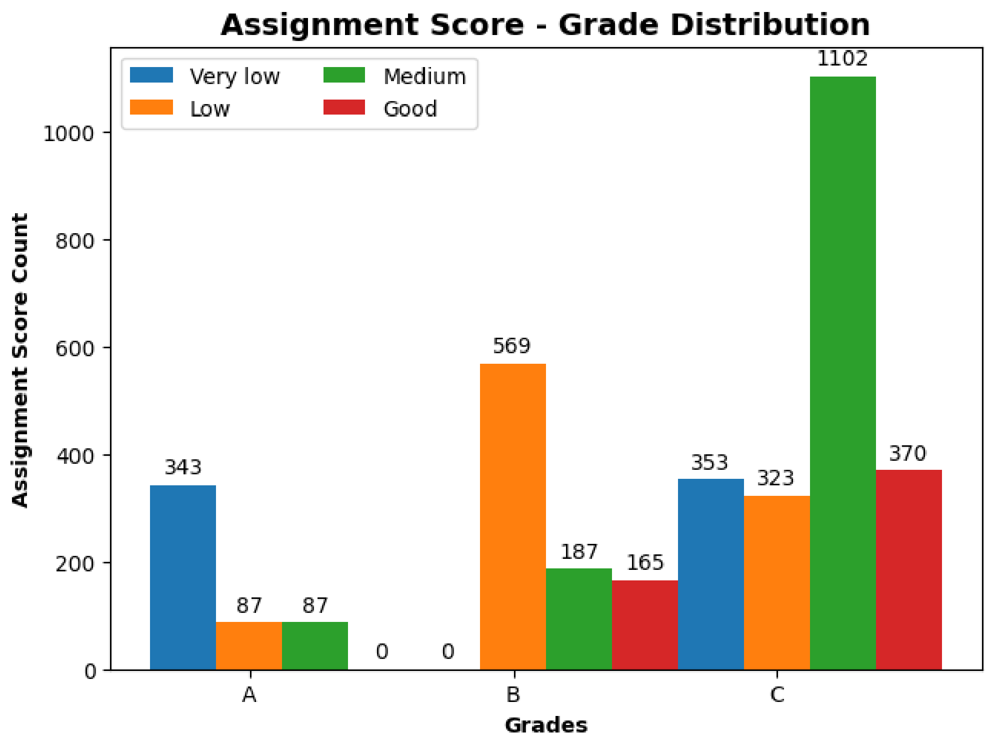


Figure Many.18: Student Assignment Score Distribution

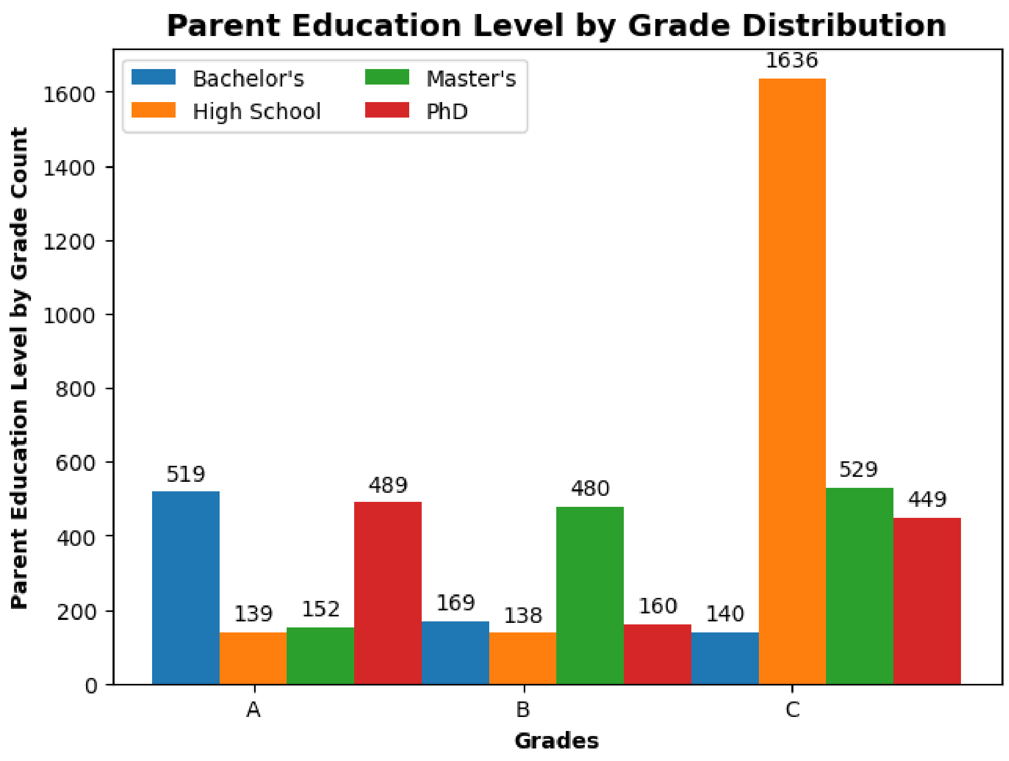


Figure Many.19: Student parent educational level and grade distribution.

```
Model: dtc
Accuracy: 0.0
              precision    recall  f1-score   support

           0       0.63      0.98      0.76       748
           1       0.43      0.04      0.07       239
           2       0.14      0.01      0.02       213

    accuracy                           0.62      1200
   macro avg       0.40      0.34      0.28      1200
weighted avg       0.50      0.62      0.49      1200
```

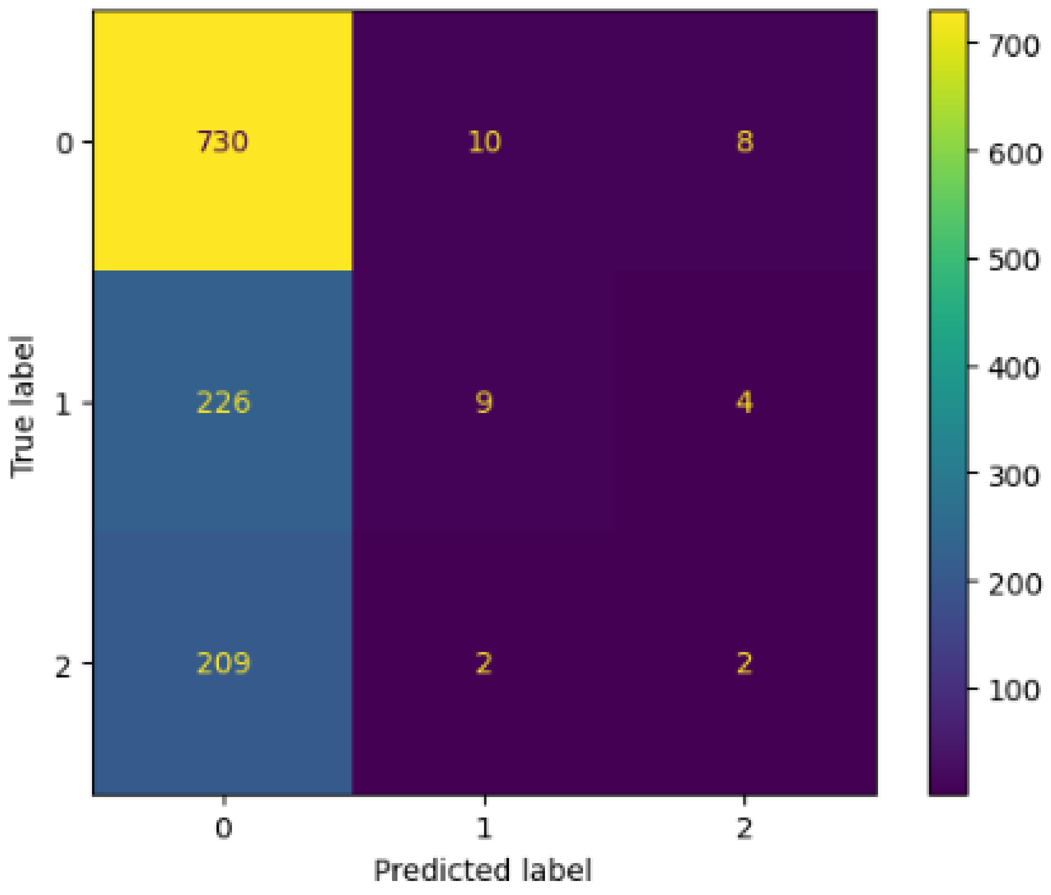


Figure Many.20: Best performing model.

Appendix

Listing 1: Import Needed Python Packages

```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Markdown as mkd

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,
↪  root_mean_squared_error, mean_absolute_error


from collections import Counter
```

Listing 2: Functions Used

```

# Functions used
# These are the functions used in the task

# function to fix grading inconsistent issue.
"""
Using;
↪  https://www.mastersportal.com/articles/2290/how-to-convert-uk-grades-for-masters

Converting British grades into American grades
In the US, the grading system uses letters A-F (without E) to
↪  evaluate students. D is the minimum passing grade.

+70% = A
60-69% = B
50-59% = C
40-49% = D
Below 40% = F (fail)
"""
def grader(total_score):
"""
This function is used to compute grades for the students.
This also applies a known, global and generally accepted grading
↪  system.
The input is the total score.
"""
"""Grades : A, B, C, D, E, F"""
if isinstance(total_score, int) or isinstance(total_score, float):
if total_score < 40:
return 'F'
# elif total_score <= 50:
#     return 'E'
elif total_score <= 49:
return 'D'
elif total_score <= 59:
return 'C'
elif total_score <= 69:
return 'B'
else:
return 'A'
else:
raise "Wrong value type. Int or float required"
```

Listing 3: Load dataset in

```
# load the dataset in the stud_record variable
stud_record = pd.read_csv("datasets/Students_Grading_Dataset.csv")

# convert all features to lovercase
stud_record.columns = [col.lower() for col in stud_record.columns]
```

Listing 4: Show data shape

```
# print data shape
mkd(f"The dataset has **{stud_record.shape[0]} rows** and
↪  **{stud_record.shape[1]} columns**")
```

Listing 5: Show data head

```
# inspect first five entries
stud_record.head()
```

Listing 6: Show dataset head

```
# inspect first five entries
stud_record.head()
```

Listing 7: Show dataset tail

```
# inspect first five entr# inspect last five entries of the dataset
stud_record.tail()
\captionof{listing}{Show inconsistent grading}
\label{code:inconsistent_grading}
\begin{minted}{python}
stud_record[['total_score', 'final_score', 'grade']]
```

Listing 8: Show dataset total null values

```
# inspect first five entr
# print number of null entries
mkd(f"There are **{stud_record.isna().sum().sum()} null entries** in
↪  the dataset")
```

Listing 9: Show data nulls

```
# inspect first five entr
# print column-wise number of null values
stud_record.isna().sum()
```

Listing 10: Fill null values

```
# let no Assignment_avg mean the student scored 0 in the assignment
# let no Attendance (%) be filled with the most common value, mode as
↪  this is not affected by outlier
# let Parent_Education_Level  be filled the mode of the feature as
↪  this is not affected by outlier
stud_record.fillna({
        'assignments_avg': 0,
        'attendance (%)' : stud_record['attendance (%)'].mode()[0],
        'parent_education_level':
         ↪  stud_record['parent_education_level'].mode()[0]},
inplace=True)

# confirm null values filled
stud_record.isna().sum()
```

Listing 11: Check for duplicates

```
# check if duplicate data exist
mkd(f" There are **{int(stud_record[stud_record.duplicated() ==
↪  True].sum().sum())} duplicates**")
```

Listing 12: Show descriptive statistics

```
# Descriptive statistics of the dataset. This can be used to identify
↪  outliers
stud_record.describe()
```

Listing 13: Confirm datatypes match expected types

```
# check and confirm each datatype fits the feature name
stud_record.info()
```

Listing 14: Show inconsistent grading

```
stud_record[['total_score', 'final_score', 'grade']]
```

Listing 15: Fix inconsistent data type

```
stud_record['grade_corrected'] =
↪  stud_record['total_score'].apply(grader)
# show grading inconsistency once more,
# investigate accurate fix
stud_record[['final_score', 'total_score', 'grade',
↪  'grade_corrected']].head()
```

Listing 16: Show studetn gender distribution

```
# students gender distribution
gender_distr = stud_record.groupby(by='gender').size()

gender_distr.plot(kind='bar')
plt.title('Student Gender Distribution', fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel('Student Gender', fontdict={'fontweight':'bold'})
plt.ylabel('Gender Count', fontdict={'fontweight':'bold'})
plt.ylim(2400, 2600)
plt.grid()

# show gender distribution figures
gender_distr
```

Listing 17: Show student age distribution

```
# student age distribution
age_dist = stud_record.groupby(by = 'age').size()
age_dist.plot(kind='bar')
plt.ylim(600, 800)
plt.grid()
plt.title("Student Age Distribution", fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel('Student Age', fontdict={'fontweight':'bold'})
plt.ylabel("Student Age Count",
↪  fontdict={'fontweight':'bold'})
```

```
# pd.DataFrame([age_dist.values, age_dist.index],
↪  columns=['Numbers', 'values'])
age_dist.index
pd.DataFrame({'Age Group': age_dist.index, 'Age Group Count':
↪  age_dist.values})
```

Listing 18: Student department distribution

```
# group dataset by department and get size of each
↪  department
dept_dist = stud_record.groupby(by='department').size()
dept_dist.plot(kind='bar')
plt.title('Department Distribution', fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel('Department', fontdict={'fontweight':'bold'})
plt.ylabel('Department Count',
↪  fontdict={'fontweight':'bold'})
plt.grid()
# plt.show()

pd.DataFrame({'Department':dept_dist.index, "No. of
↪  Students": dept_dist.values})
```

Listing 19: Show attendance plot

```
# plot student attendance
stud_record['attendance (%)'].plot()
plt.title("Attendance Plot [First 100]", fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel("Student Attendance",
↪  fontdict={'fontweight':'bold'})
plt.ylabel("Student Attendance (%)",
↪  fontdict={'fontweight':'bold'})
plt.xlim(1, 100)
plt.show()
```

Listing 20: Show midterm score

```
# midterm_score plot of first 100 students
stud_record['midterm_score'].plot()
plt.xlim(1,100)
plt.title("Midterm Score plot [First 100]",
↪  fontdict={'size':14, 'fontweight':'bold'})
plt.xlabel("Midterm score", fontdict={'fontweight':'bold'})
plt.ylabel("Midterm score count",
↪  fontdict={'fontweight':'bold'})
plt.show()
```

Listing 21: Final score plot for the first 200 students

```
# Final Score Plot
stud_record['final_score'].plot()
plt.xlim(1,100)
plt.title("Final Score [First 200]", fontdict={'size':14,
↪  'fontweight':'bold'})
plt.xlabel("Final Score", fontdict={'fontweight':'bold'})
plt.ylabel('Final Score Count', fontdict={'fontweight':'bold'})

plt.show()
```

Listing 22: Student Internet Home Access Distribution

```
# dataset grouping by internet_access_at_home and getting the sizes
internet_access_at_home =
↪  stud_record.groupby(by='internet_access_at_home').size()

internet_access_at_home.plot(kind='bar')
plt.title('Student Internet Home Access Distribution',
↪  fontdict={'size':14, 'fontweight':'bold'})
plt.xlabel('Home Internet Accesss', fontdict={'fontweight':'bold'})
plt.ylabel('Home Internet Access Count',
↪  fontdict={'fontweight':'bold'})
# plt.ylim(2400, 2600)
plt.grid()
```

Listing 23: Student family income level distribution

```
Counter(stud_record.family_income_level)

family_income_level =
↪  stud_record.groupby(by='family_income_level').size()

family_income_level.plot(kind='bar')
plt.title('Student Family Income Level Distribution',
↪  fontdict={'size':14, 'fontweight':'bold'})
plt.xlabel('Student Family Income Level',
↪  fontdict={'fontweight':'bold'})
plt.ylabel('Student Family Income Level Count',
↪  fontdict={'fontweight':'bold'})
plt.ylim(1000)
plt.grid()
```

Listing 24: Student stress level

```
# group dataset by stress_level(1-10) and get size of each stress
↪  level
```

```
stress_level = stud_record.groupby(by='stress_level (1-10)').size()
stress_level = pd.DataFrame(stress_level)
stress_level.plot(kind='bar')
plt.title('Student Stress Level Distribution', fontdict={'size':14,
 ↪  'fontweight':'bold'})
plt.xlabel('Student Stress Level', fontdict={'fontweight':'bold'})
plt.ylabel('Student Stress Level Count',
 ↪  fontdict={'fontweight':'bold'})
plt.ylim(400)
plt.grid()
plt.show();
```

Listing 25: Student parent educational level

```
# dataset grouped by paretn_education_level and the different sizes
 ↪  obtained
parent_education =
 ↪  stud_record.groupby(by='parent_education_level').size()
parent_education.plot(kind='bar')
plt.title("Parent Education Level", fontdict={'fontweight':'bold',
 ↪  'fontsize':14})
plt.xlabel('Parent Education Level', fontdict = {'fontweight':'bold',
 ↪  'fontsize':10})
plt.ylabel('Count', fontdict={'fontweight':'bold', 'fontsize':10})
plt.grid()

plt.show();
```

Listing 26: Student extra curricular activities

```
# data grouped by extra curricular activities of student. Size of yes
 ↪  / no extracted
extracurricular =
 ↪  stud_record.groupby(by='extracurricular_activities').size()

extracurricular.plot(kind='bar')
plt.title('Extrac Curricular Activities Dist',
 ↪  fontdict={'fontweight':'bold', 'size':14})
plt.xlabel('Extrac Curricular Activities',
 ↪  fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Extra Curricular Count', fontdict={'fontweight':'bold',
 ↪  'size':10})
plt.grid()
plt.show()
```

Listing 27: Student grade distribution

```
# group dataset by grade_corrected and get size of each grade
grade = stud_record.groupby(by='grade_corrected').size()
```

```
grade.plot(kind='bar')
plt.title("Student Grades", fontdict={'fontweight':'bold',
↪  'size':14})
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Grade Count', fontdict={'fontweight':'bold', 'size':10})
plt.grid()

plt.show()
```

Listing 28: Gender grade distribution

```
# group dataset by grade_corrected and gender
gender_grade = stud_record.groupby(by=['grade_corrected',
↪  'gender']).count()['student_id'].reset_index()
gender_grade

valuez = np.array(gender_grade['student_id'])
rows = gender_grade['grade_corrected'].unique() # rows to be plotted
num_rows = int(len(gender_grade['grade_corrected'].unique())) # num
↪  of rows for reshaping
num_cols = int(len(gender_grade) / len(rows))   # num of cols for
↪  reshaping

data = pd.DataFrame(valuez.reshape((num_rows, num_cols)),
index=pd.Index(list(rows), name='grade'),
columns=pd.Index(['female','male'], name=''))
data = data.reset_index()
data
```

Listing 29: Grade by department distribution

```
# investigation of grade-department relationship
department_grade = stud_record.groupby(by = ['grade_corrected',
↪  'department']).size().reset_index()
department_grade

rows = department_grade['grade_corrected'].unique() # rows to be
↪  plotted
num_rows = int(len(department_grade['grade_corrected'].unique())) #
↪  num of rows for reshaping
num_cols = int(len(department_grade) / len(rows))   # num of cols for
↪  reshaping

data = pd.DataFrame(department_grade[0].values.reshape(num_rows,
↪  num_cols),
index=pd.Index(list(rows), name='grade'),
columns = department_grade['department'].unique()
)
```

```python
data = data.reset_index()

grades = tuple(data['grade'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper right', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Department Grade Distrution',
 ↪  fontdict={'fontweight':'bold', 'size':10})
plt.title('Department Grade Distribution',
 ↪  fontdict={'fontweight':'bold', 'size':14})

plt.show()
```

Listing 30: Grade by attendance distribution

```python
# making a deep copy of the dataframe
stud_record_updated = stud_record.copy()

# Confirm the two records are different
stud_record_updated is stud_record

labels = ["Very low", "Low", "Medium", "Good", 'Very good']

stud_record_updated['attendance_classified'] =
 ↪  pd.cut(np.array(list(stud_record_updated['attendance (%)'])),
 ↪  len(labels), labels = labels)
```

```
attendance_grade = stud_record_updated.groupby(by =
 ↪  ['attendance_classified', 'grade_corrected'],
 ↪  observed=False).size().reset_index()

rows = attendance_grade['grade_corrected'].unique()
num_rows = int(len(attendance_grade['grade_corrected'].unique())) #
 ↪  num of rows for reshaping
num_cols = int(len(attendance_grade) / len(rows))   # num of cols for
 ↪  reshaping

data = pd.DataFrame(attendance_grade[0].values.reshape(num_rows,
 ↪  num_cols),
index=pd.Index(list(rows), name='grade'),
columns = attendance_grade['attendance_classified'].unique()
)
data = data.reset_index()
grades = tuple(data['grade'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper right', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Attendance Grade Count', fontdict={'fontweight':'bold',
 ↪  'size':10})
plt.title('Attendance Grade Distribution',
 ↪  fontdict={'fontweight':'bold', 'size':14})

plt.show();
```

Listing 31: Grade by assignment distribution

```
# Assignments Avg Score Distribution was classified as Very low, Low,
↪   Medium, Good, Very good
stud_record_updated['assignment_classified'] =
↪   pd.cut(np.array(list(stud_record_updated['assignments_avg'])),
↪   len(labels), labels = labels)
assignment_avg = stud_record_updated.groupby(by =
↪   ['assignment_classified', 'grade_corrected'],
↪   observed=False).size().reset_index()


# stud_record_updated['assignment_avg_classified'] =
↪   pd.cut(np.array(list(stud_record_updated['assignments_avg'])),
↪   len(labels), labels = labels)
# assignment_avg = stud_record_updated.groupby(by =
↪   ['assignment_avg_classified', 'grade_corrected'],
↪   observed=False).size().reset_index()

rows = assignment_avg['grade_corrected'].unique()
num_rows = int(len(assignment_avg['grade_corrected'].unique())) # num
↪   of rows for reshaping
num_cols = int(len(assignment_avg) / len(rows))   # num of cols for
↪   reshaping

data = pd.DataFrame(assignment_avg[0].values.reshape(num_rows,
↪   num_cols),
index=pd.Index(list(rows), name='grade_corrected'),
columns = assignment_avg['assignment_classified'].unique()
)

data = data.reset_index()
grades = tuple(data['grade_corrected'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1
```

```
# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper left', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Assignment Score Count', fontdict={'fontweight':'bold',
↪  'size':10})
plt.title('Assignment Score - Grade Distribution',
↪  fontdict={'fontweight':'bold', 'size':14})

plt.show();
```

Listing 32: Grade by quizz score distribution

```
stud_record_updated['quizzes_avg_classified'] =
↪  pd.cut(np.array(list(stud_record_updated['quizzes_avg'])),
↪  len(labels), labels = labels)
assignment_avg = stud_record_updated.groupby(by =
↪  ['assignment_classified', 'grade_corrected'],
↪  observed=False).size().reset_index()


# stud_record_updated['assignment_avg_classified'] =
↪  pd.cut(np.array(list(stud_record_updated['assignments_avg'])),
↪  len(labels), labels = labels)
# assignment_avg = stud_record_updated.groupby(by =
↪  ['assignment_avg_classified', 'grade_corrected'],
↪  observed=False).size().reset_index()

rows = assignment_avg['grade_corrected'].unique()
num_rows = int(len(assignment_avg['grade_corrected'].unique())) # num
↪  of rows for reshaping
num_cols = int(len(assignment_avg) / len(rows))    # num of cols for
↪  reshaping
num_cols

data = pd.DataFrame(assignment_avg[0].values.reshape(num_rows,
↪  num_cols),
index=pd.Index(list(rows), name='grade_corrected'),
columns = assignment_avg['assignment_classified'].unique()
)

data = data.reset_index()
grades = tuple(data['grade_corrected'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper left', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Quizz Score Count', fontdict={'fontweight':'bold',
↪  'size':10})
plt.title('Quizz Score Distribution', fontdict={'fontweight':'bold',
↪  'size':14})

plt.show();
```

Listing 33: Grade by study hours per week distribution

```
stud_record_updated['study_hours_per_week_classified'] =
↪  pd.cut(np.array(list(stud_record_updated['study_hours_per_week'])),
↪  len(labels), labels = labels)
assignment_avg = stud_record_updated.groupby(by =
↪  ['study_hours_per_week_classified', 'grade_corrected'],
↪  observed=False).size().reset_index()

rows = assignment_avg['grade_corrected'].unique()
num_rows = int(len(assignment_avg['grade_corrected'].unique())) # num
↪  of rows for reshaping
num_cols = int(len(assignment_avg) / len(rows))    # num of cols for
↪  reshaping
num_cols

data = pd.DataFrame(assignment_avg[0].values.reshape(num_rows,
↪  num_cols),
index=pd.Index(list(rows), name='grade_corrected'),
columns = assignment_avg['study_hours_per_week_classified'].unique()
)

data = data.reset_index()
grades = tuple(data['grade_corrected'].values)
scores = {
        data.columns[1] : tuple(data[data.columns[1]].values),
        data.columns[2] : tuple(data[data.columns[2]].values),
        data.columns[3] : tuple(data[data.columns[3]].values),
        data.columns[4] : tuple(data[data.columns[4]].values),
}

x = np.arange(len(grades))
width = 0.25 # bar width
multiplier = 0

fig, ax = plt.subplots(layout='constrained')

for attribute, measurement in scores.items():
offset = width * multiplier
rects = ax.bar(x + offset, measurement, width, label = attribute)
ax.bar_label(rects, padding=3)
multiplier += 1

# Add some text for labels, title and custom x-axis tick labels etc
ax.set_ylabel("ylable")
ax.set_title("Title here", fontdict={'fontweight':'bold', 'size':14})
ax.set_xticks(x + width, rows)
ax.legend(loc='upper left', ncols = len(rows) - 1)
# ax.set_ylim()
plt.xlabel('Grades', fontdict={'fontweight':'bold', 'size':10})
plt.ylabel('Quizz Score Count', fontdict={'fontweight':'bold',
↪  'size':10})
plt.title('Quizz Score Distribution', fontdict={'fontweight':'bold',
↪  'size':14})

plt.show();
```

Listing 34: Label encoding and scaling functions

```
def encoder(dataframe, val = LabelEncoder):
"""
Function accepts dataframe and uses the encoding method passed in to
↪  encode the categorical data
argument: dataframe to be encoded
returns encoded dataframe
"""
le = val()

for feature in dataframe:
dataframe[feature] = le.fit_transform(dataframe[feature])
```

```
return dataframe

def scaler(dataframe):
"""
Function normalizes data before it is used in machine learning model.
↪  This process makes it easier for models to understand
the features and predict them better.
argument is the dataframe to be scaled.
return : scaled dataframe
"""
standard_scaler = StandardScaler()
return standard_scaler.fit_transform(dataframe)

# # inspect result
# stud_encoded = encoder(stud_record_choice_features).head()
# scaler(stud_encoded)
```

Listing 35: X and y features separation

```
# select dependent and independent features
X_features = stud_record_choice_features.drop('grade_corrected',
↪  axis='columns')
y_feature = stud_record_choice_features['grade_corrected']

# split data into train test set
X_features_train, X_features_test, y_feature_train, y_feature_test =
↪  train_test_split(X_features, y_feature, train_size=0.7,
↪  random_state=42)

# create validation test set
X_features_test, X_features_valid, y_feature_test, y_feature_valid =
↪  train_test_split(X_features_test, y_feature_test, train_size=0.8,
↪  random_state=42)
```

Listing 36: heading

```
# encode and scale parameters
X_train =
↪  scaler(encoder(X_features_train.select_dtypes(include='object')))
X_test =
↪  scaler(encoder(X_features_test.select_dtypes(include='object')))
X_valid =
↪  scaler(encoder(X_features_valid.select_dtypes(include='object')))

le = LabelEncoder()
y_train = le.fit_transform(y_feature_train)
y_test = le.fit_transform(y_feature_test)
y_valid = le.fit_transform(y_feature_valid)
```

Listing 37: Import model building packages

```python
# import modelling packages
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
```

Listing 38: Model dictionary

```python
# building baseline model
models = {
        'xgb' : XGBClassifier,
        'rdf' : RandomForestClassifier,
        'dtc' : DecisionTreeClassifier,
}
```

Listing 39: Iterate model dictionary, create, train and evaluate models

```python
for model in models:
print(f"Model: {model}")
clf = models[model]()
clf.fit(X_train, y_feature_train)
y_pred = clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_feature_test, y_pred)}")
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
↪  display_labels=clf.classes_)
disp.plot()
plt.show()
```