Project Report Format

1. INTRODUCTION

1.1 Project Overview

Student performance analysis and prediction using datasets has become an essential component of modern education systems. With the increasing availability of data on student demographics, academic history, and other relevant factors, schools and universities are using advanced analytics and machine learning algorithms to gain insights into student performance and predict future outcomes. This approach helps educators identify areas of improvement, personalize learning experiences, and provide targeted support to struggling students. Furthermore, student performance analysis and prediction can also aid in decision-making processes for school administrators and policymakers, helping them allocate resources more effectively. In this article, we will explore the benefits of using datasets for student performance analysis and prediction and discuss some of the methods and tools used in this field.

1.2 Purpose

This project understands how the student's performance (test scores) is affected by other variables such as Gender, Ethnicity, Parental level of education, and Lunch and Test preparation course.

The primary objective of higher education institutions is to impart quality education to their students. To achieve the highest level of quality in the education system, knowledge must be discovered to predict student enrollment in specific courses, identify issues with traditional classroom teaching models, detect unfair means used in online examinations, detect abnormal values in student result sheets, and predict student performance. This knowledge is hidden within educational datasets and can be extracted through data mining techniques.

2. IDEATION & PROPOSED SOLUTION

2.1 Problem Statement Definition

As an educator or administrator, you want to understand how well your students are performing academically so that you can identify areas for improvement and provide targeted support to help them succeed. However, you may be struggling with time-consuming and manual processes for collecting and analyzing student performance data. This can make it difficult to gain a comprehensive view of student progress and to identify patterns and trends in performance that may require action. Additionally, you may find it challenging to effectively communicate student progress to parents and other stakeholders. Therefore, you are looking for an efficient and effective solution that can automate the process of analyzing student performance data and provide meaningful insights that can be easily shared with others.

2.2 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes.

It is a useful tool to helps teams better understand their users.
Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Example

Sure! Here is an Empathy Map Canvas for student performance analysis:

Who are we empathizing with?: Educators/Administrators who are responsible for analyzing student performance data.

What do they see?:

Varying levels of student academic performance
Time-consuming and manual data collection and analysis processes
Difficulty in identifying patterns and trends in student performance data
Inability to provide targeted support due to lack of actionable insights
Challenges in effectively communicating student progress to parents and other stakeholders

What do they hear?:

Student progress reports from teachers and other educators
Concerns and feedback from parents and guardians
Suggestions from other administrators on how to improve student performance
Frustration and stress from trying to manage and analyze large amounts of data

What do they say and do?:

Collect and analyze student performance data to identify areas for improvement
Seek out resources and support to help analyze and interpret data effectively
Work collaboratively with teachers and other educators to identify targeted support for struggling students
Communicate student progress to parents and other stakeholders using various formats (e.g., reports, meetings, etc.)
Monitor student progress over time and adjust support strategies as needed

What do they think and feel?:

Concerned about student progress and success
Overwhelmed by the amount of data to manage and analyze
Frustrated with time-consuming and manual processes
Motivated to find new and innovative ways to support student success
Dedicated to improving student outcomes and achieving academic goals

Overall, the empathy map highlights the need for educators and administrators to have an efficient and effective solution for analyzing student performance data that can provide actionable insights and support targeted interventions for student success.

Regenerate response

Inability to provide targeted support due to lack of actionable insights
Challenges in effectively communicating student progress to parents and other stakeholders

What do they hear?:

Student progress reports from teachers and other educators
Concerns and feedback from parents and guardians
Suggestions from other administrators on how to improve student performance
Frustration and stress from trying to manage and analyze large amounts of data

What do they say and do?:

Collect and analyze student performance data to identify areas for improvement
Seek out resources and support to help analyze and interpret data effectively
Work collaboratively with teachers and other educators to identify targeted support for struggling students
Communicate student progress to parents and other stakeholders using various formats (e.g., reports, meetings, etc.)
Monitor student progress over time and adjust support strategies as needed

What do they think and feel?:

Concerned about student progress and success
Overwhelmed by the amount of data to manage and analyze
Frustrated with time-consuming and manual processes
Motivated to find new and innovative ways to support student success
Dedicated to improving student outcomes and achieving academic goals

Overall, the empathy map highlights the need for educators and administrators to have an efficient and effective solution for analyzing student performance data that can provide actionable insights and support targeted interventions for student success.

Regenerate response

2.3 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Great! Here's a possible approach for a team gathering and collaboration session to select the problem statement for student performance analysis:

1. Define the purpose of the session: The purpose of this session is to collaboratively identify and select the most pressing problem statement related to student performance analysis that our team can address through a project or initiative.
2. Review the empathy map canvas: Share the empathy map canvas that we created earlier with the team, and ask them to review it individually for a few minutes. Encourage them to add any additional thoughts, feelings, or actions that they think should be included.
3. Share the problem statements: Based on the empathy map canvas, ask the team to brainstorm and write down as many problem statements related to student performance analysis as they can think of. These can be broad or specific, and should be written on separate sticky notes or index cards.
4. Group and categorize the problem statements: Ask the team to group the problem statements into categories or themes that capture the main issues or challenges related to student performance analysis. This can be done through a group discussion, or by physically moving the sticky notes or index cards into piles based on common themes.

Step-2: Brainstorm, Idea Listing and Grouping

1. Brainstorm ideas: Encourage the team to individually brainstorm and write down as many ideas as they can think of for addressing the problem statement. These can be broad or specific, and should be written on separate sticky notes or index cards.
2. Group and categorize the ideas: Once the brainstorming time is up, ask the team to group and categorize the ideas into themes or categories. This can be done through a group discussion or by physically moving the sticky notes or index cards into piles based on common themes.
3. Discuss and refine the ideas: As a group, discuss each idea and identify any potential strengths, weaknesses, or opportunities. Refine and clarify the ideas as necessary to ensure that they align with the problem statement and have the potential to address the key issues and challenges related to student performance analysis.

Step-3: Idea Prioritization

1. Discuss and refine the ranking: As a group, discuss the ranking of the ideas and identify any potential strengths, weaknesses, or opportunities associated with each idea. Refine and adjust the ranking as necessary to ensure that the most feasible, impactful, and aligned ideas are at the top of the list.
2. Select the top ideas: Once the ranking has been refined and adjusted, select the top ideas that we will focus on implementing to address the problem statement related to student performance analysis. These should be the ideas that have the highest scores based on the defined criteria and are feasible to implement within our resources and time frame.

3. Define next steps: Once the top ideas have been selected, work together as a team to define the next steps for implementing them, including assigning responsibilities, setting deadlines, and identifying any necessary resources or support.

3. REQUIREMENT ANALYSIS

3.1 Functional requirement

Jupyter Notebook

3.2 Non-Functional requirements

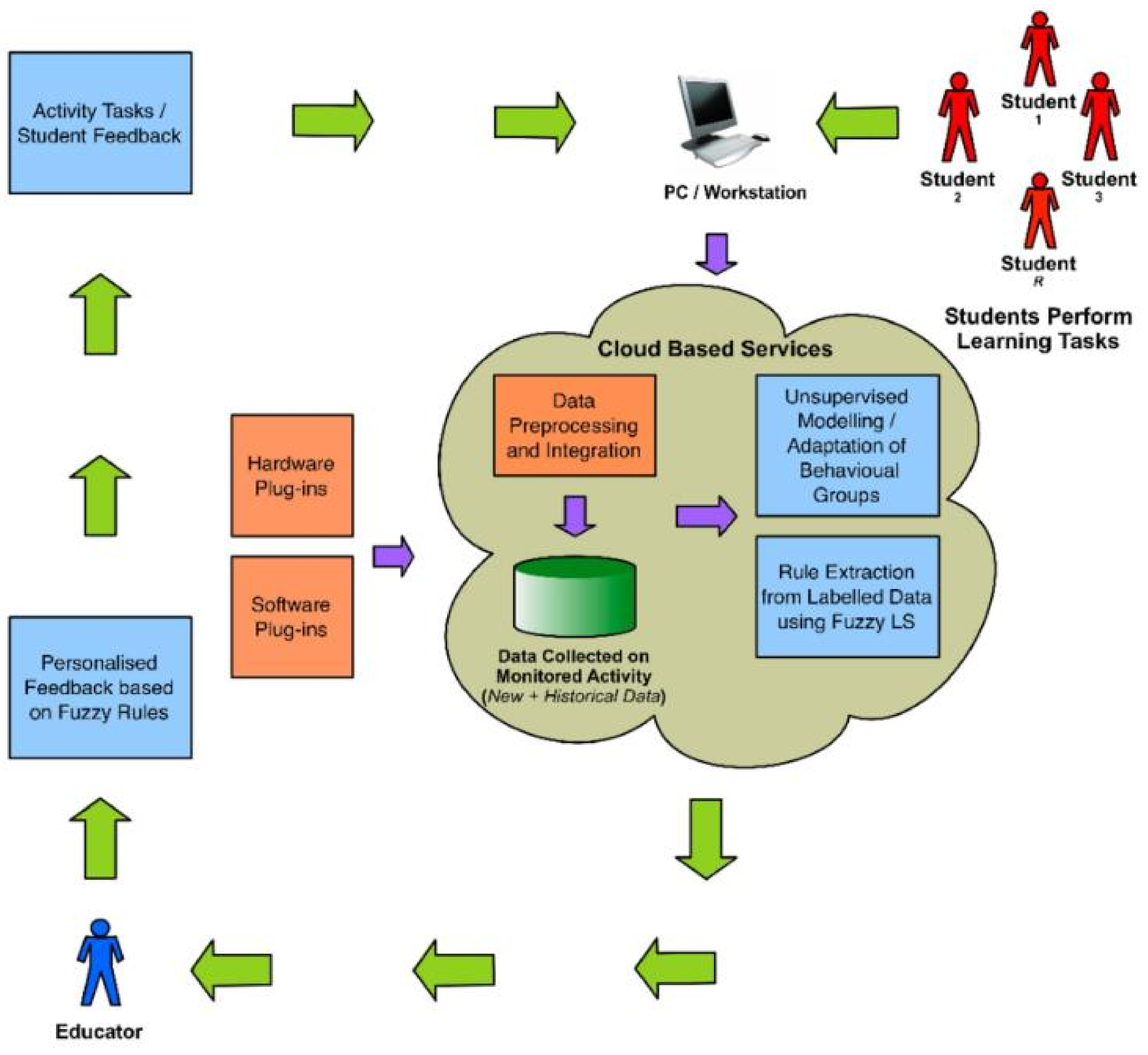
Dataset about student details.

4. PROJECT DESIGN

4.1 Data Flow Diagrams

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Example: DFD Level 0 (Industry Standard)


Activity Tasks / Student Feedback
PC / Workstation
Student 1
Student 2
Student 3
Student R
Students Perform Learning Tasks
Cloud Based Services
Data Preprocessing and Integration
Unsupervised Modelling / Adaptation of Behavioual Groups
Rule Extraction from Labelled Data using Fuzzy LS
Data Collected on Monitored Activity (New + Historical Data)
Hardware Plug-ins
Software Plug-ins
Personalised Feedback based on Fuzzy Rules
Educator

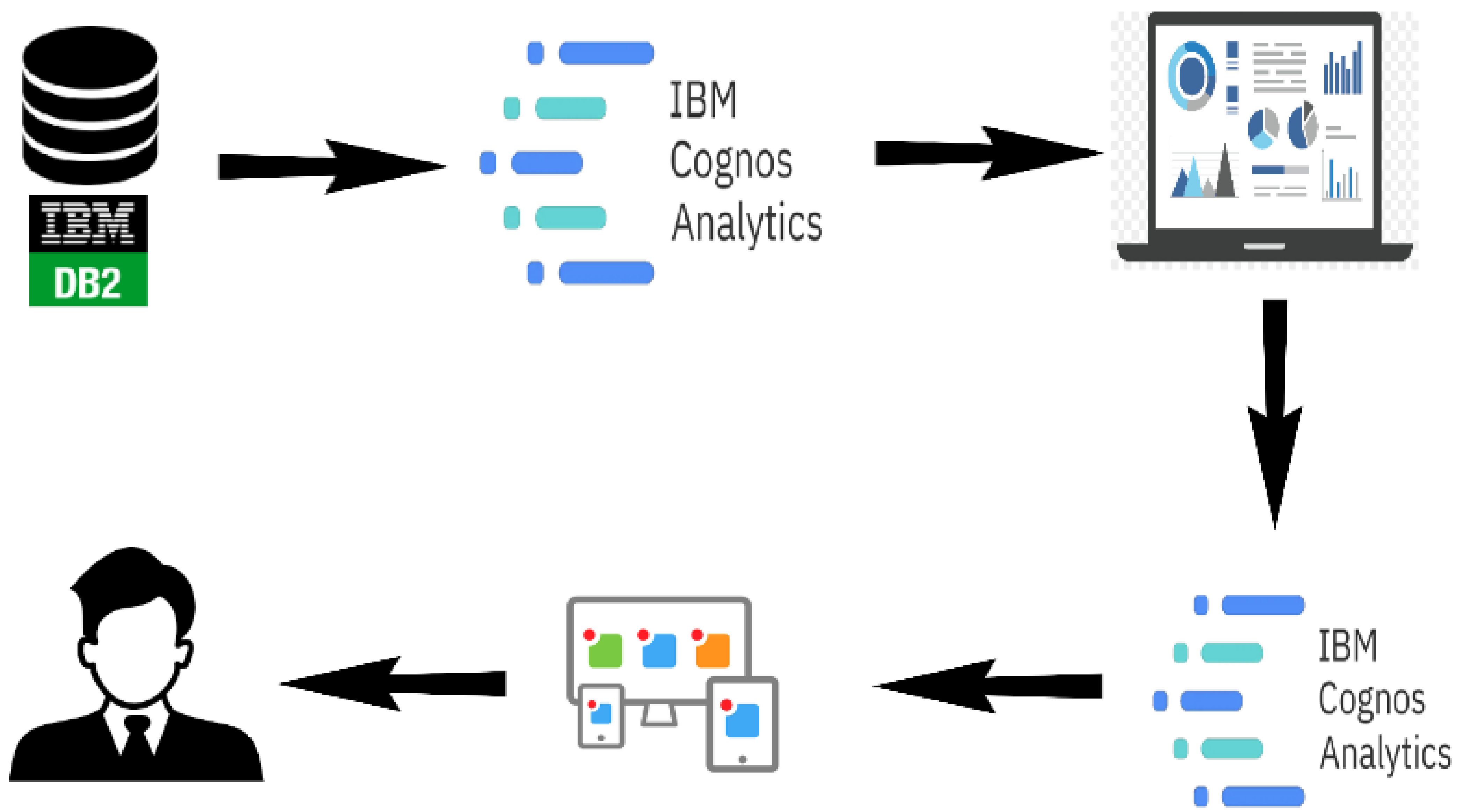
4.2 Solution & Technical Architecture

Solution architecture is a complex process – with many sub–processes – that bridges the
gap between business problems and technology solutions. Its goals are to:

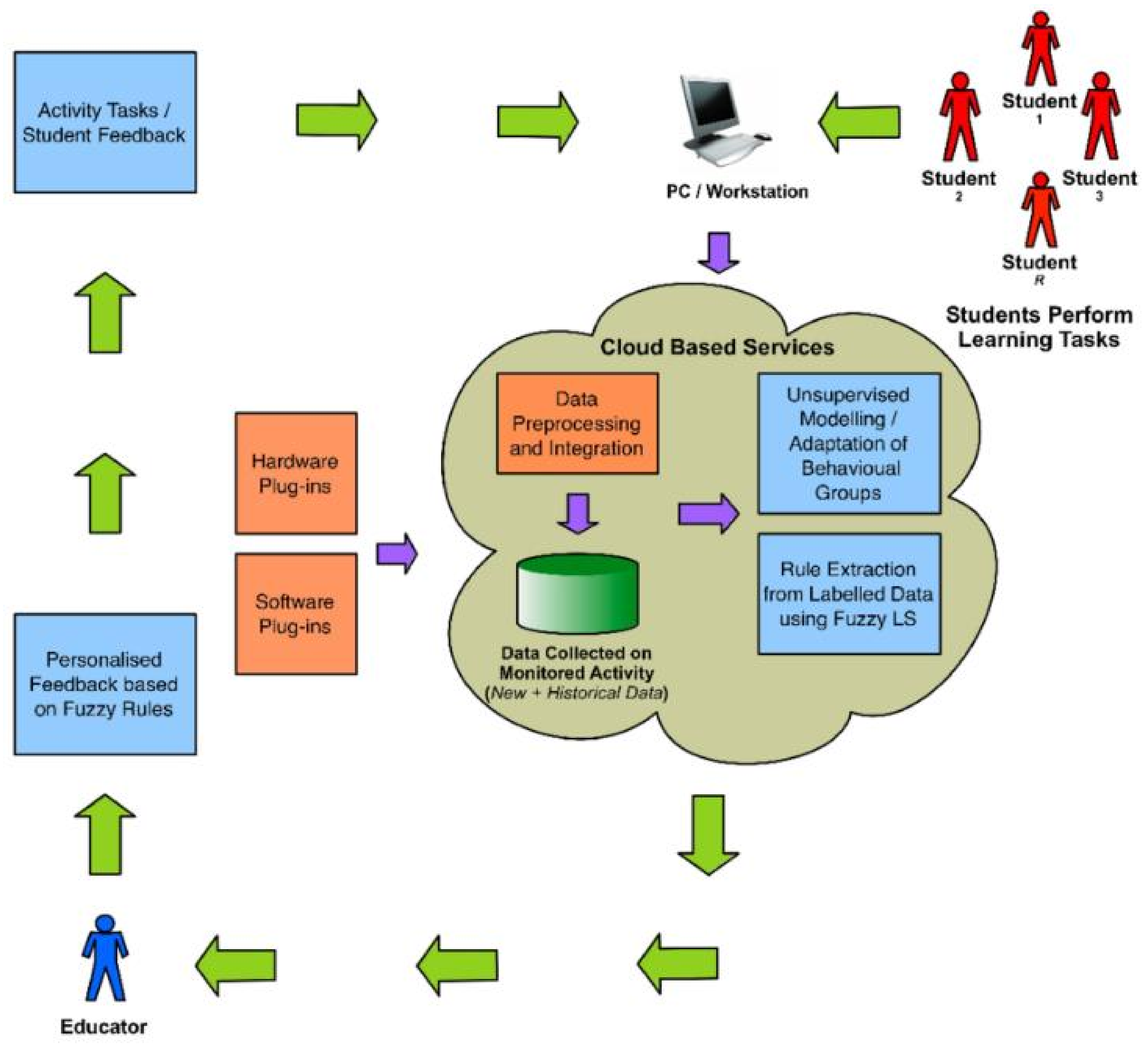
- By integrating data analytics tools to an LMS educators and administrators can gain deeper insights to the student performance Overall LMS with data analytics capabilities can help educators and administrators make more informed decisions about student performance and improve outcome for all students .
- When communicating with the project stake holders about the software project it is clearly important to provide concise description of a structures ,characteristics, behaviour and other aspects of software.
- Feature: ability to track the student progress,analyze student performance data provide personalised feedback and generate reports.
- Development phases: planning, requirement gathering,design development testing and deployment
- Solution Requirements: Includes functional requirements such as ability to track student progress and generate reports as well as non –functional requirements like scalability ,security and useability .

- specifications according to which the solution is defined, managed, and delivered here requirements, technology stack ,architecture ,project management,testing,deployment,documentation.

Example – Solution Architecture Diagram:



4.3 User Stories


Activity Tasks / Student Feedback
PC / Workstation
Student 1
Student 2
Student 3
Student R
Students Perform Learning Tasks
Cloud Based Services
Data Preprocessing and Integration
Unsupervised Modelling / Adaptation of Behavioual Groups
Rule Extraction from Labelled Data using Fuzzy LS
Data Collected on Monitored Activity (New + Historical Data)
Hardware Plug-ins
Software Plug-ins
Personalised Feedback based on Fuzzy Rules
Educator

5. CODING

Importing Pandas, Numpy, Matplotlib, Seaborn and Warings Library.

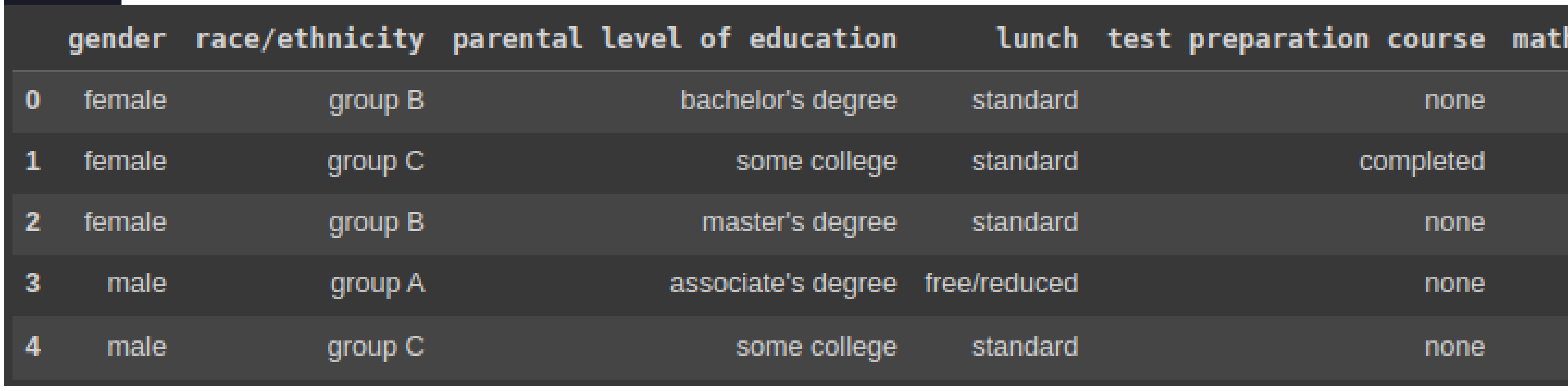
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Import the CSV Data as Pandas DataFrame

```
df = pd.read_csv("data/StudentsPerformance.csv")
```

Show the top 5 Recoreds

```
df.head()
```



	gender	race/ethnicity	parental level of education	lunch	test preparation course	mat
0	female	group B	bachelor's degree	standard	none	
1	female	group C	some college	standard	completed	
2	female	group B	master's degree	standard	none	
3	male	group A	associate's degree	free/reduced	none	
4	male	group C	some college	standard	none	

show the top 5 records on the dataset and look at the features.

To see the shape of the dataset

```
df.shape
```

```
(1000, 8)
```

And it will help to find the shape of the dataset.

Dataset Information

gender: sex of students —> (Male/Female)

race/ethnicity: ethnicity of students —> (Group A, B, C, D, E)

parental level of education: parents' final education —>(bachelor's degree, some college, master's degree, associate's degree)

lunch: having lunch before test (standard or free/reduced)

test preparation course: complete or not complete before test

math score

reading score

writing score

After that, we check the data as the next step. There are a number of categorical features contained in the dataset, including multiple missing value kinds, duplicate values, check data types, and a number of unique value types.

Data Checks to Perform

Check Missing values

Check Duplicates

Check data type

Check the number of unique values in each column

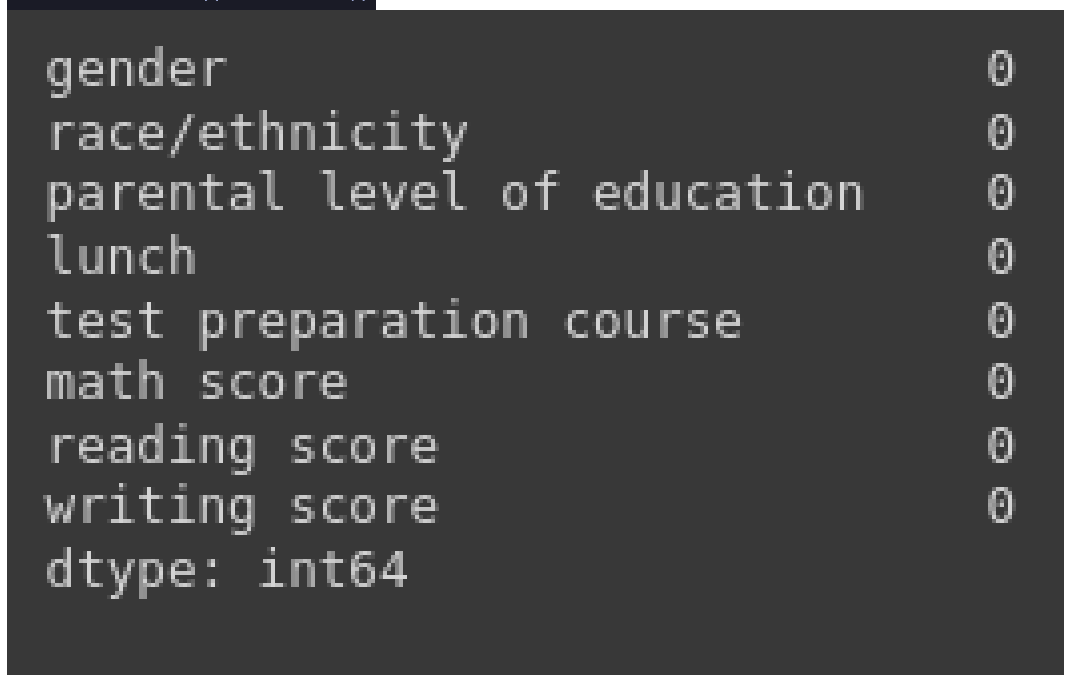
Check the statistics of the data set

Check various categories present in the different categorical column

Check Missing Values

To check every column of the missing values or null values in the dataset.

```
df.isnull().sum()
```



```
gender                         0
race/ethnicity                 0
parental level of education    0
lunch                          0
test preparation course        0
math score                     0
reading score                  0
writing score                  0
dtype: int64
```

If there are no missing values in the dataset.

Check Duplicates

If checking the our dataset has any duplicated values present or not

```
df.duplicated().sum()
```

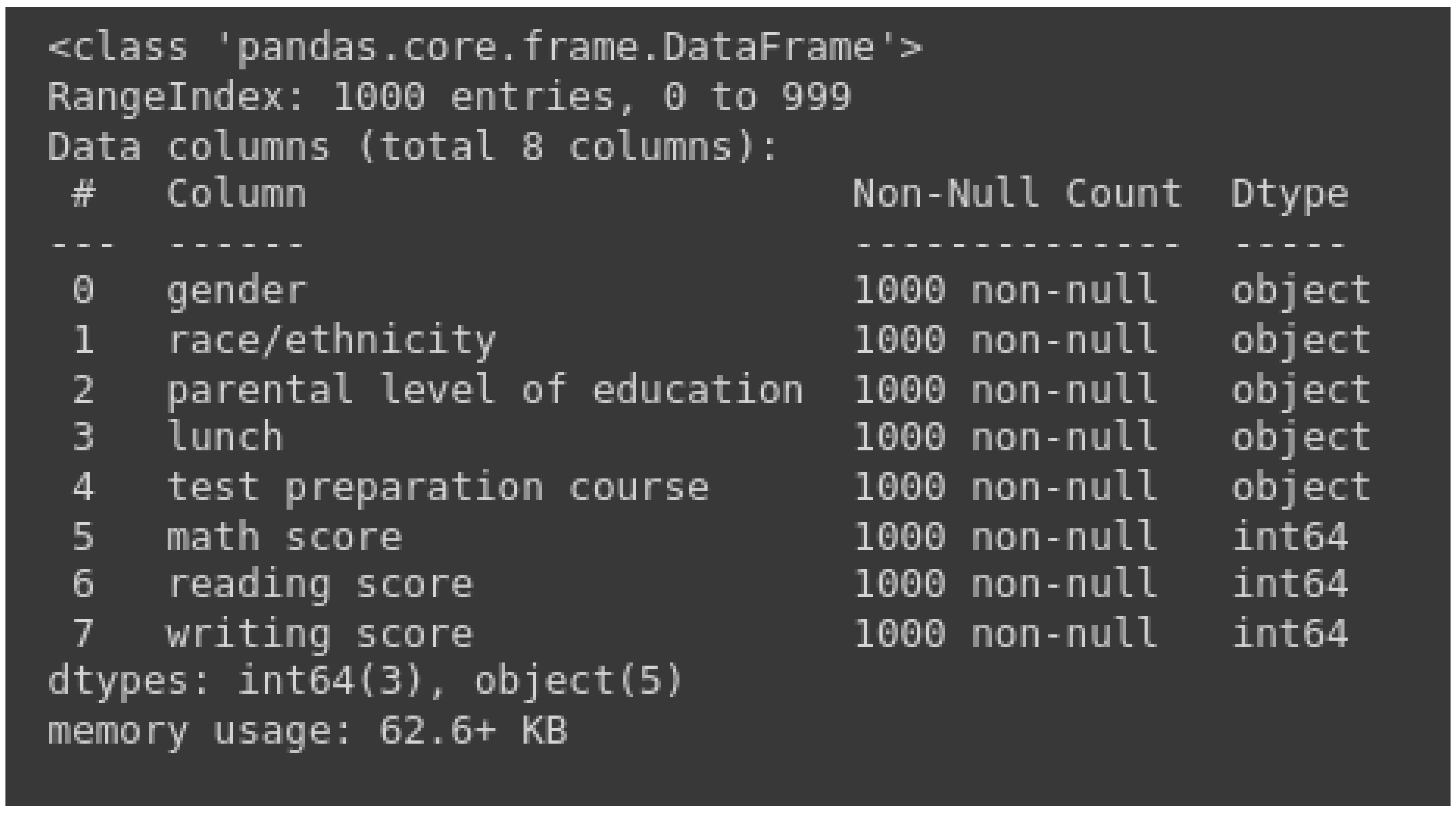
```
0
```

There are no duplicates values in the dataset.

Check the Data Types

To check the information of the dataset like datatypes, any null values present in the dataset.

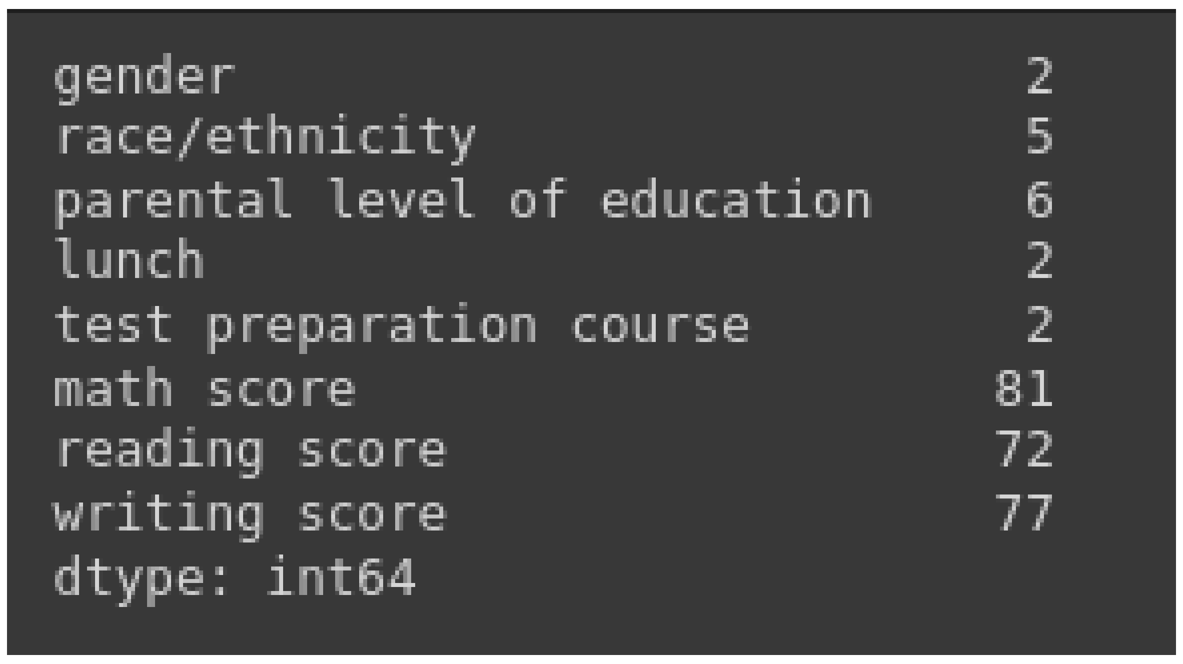
```
#check the null and Dtypes
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64 
 6   reading score                1000 non-null   int64 
 7   writing score                1000 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

Check the Number of Unique Values in Each Column

```
df.nunique()
```



```
gender                          2
race/ethnicity                  5
parental level of education     6
lunch                           2
test preparation course         2
math score                     81
reading score                  72
writing score                  77
dtype: int64
```

Check Statistics of the Data Set

To examine the dataset's statistics and determine the data's statistics.

Insight

The numerical data shown above shows that all means are fairly similar to one another, falling between 66 and 68.05.

The range of all standard deviations, between 14.6 and 15.19, is also narrow.

While there is a minimum score of 0 for math, the minimums for writing and reading are substantially higher at 10 and 17, respectively.

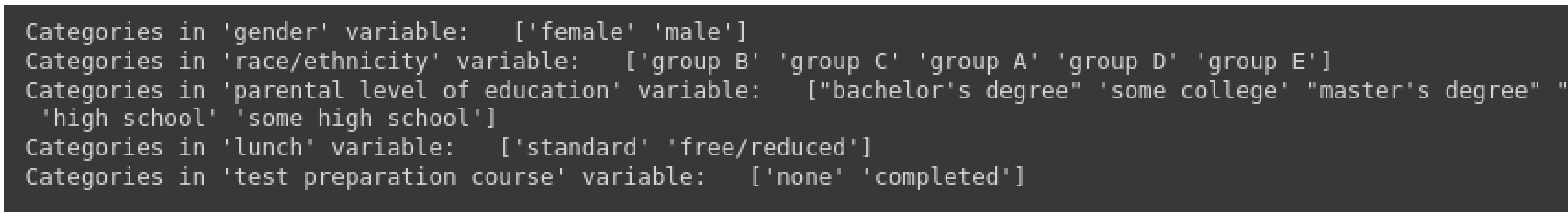
We don't have any duplicate or missing values, and the following codes will provide a good data checking.

Exploring Data

```
print("Categories in 'gender' variable:   ",end=" ")
print(df["gender"].unique())

print("Categories in 'race/ethnicity' variable:   ",end=" ")
print(df["race/ethnicity"].unique())

print("Categories in 'parental level of education' variable:   ",end=" ")
print(df["parental level of education"].unique())

print("Categories in 'lunch' variable:   ",end=" ")
print(df["lunch"].unique())

print("Categories in 'test preparation course' variable:   ",end=" ")
print(df["test preparation course"].unique())
```

The unique values in the dataset will be provided and presented in a pleasant way in the code above.

The output will following:



```
Categories in 'gender' variable:   ['female' 'male']
Categories in 'race/ethnicity' variable:   ['group B' 'group C' 'group A' 'group D' 'group E']
Categories in 'parental level of education' variable:   ["bachelor's degree" 'some college' "master's degree" "
 'high school' 'some high school']
Categories in 'lunch' variable:   ['standard' 'free/reduced']
Categories in 'test preparation course' variable:   ['none' 'completed']
```

We define the numerical and categorical columns:

```
#define numerical and categorical columns
numeric_features = [feature for feature in df.columns if df[feature].dtype != "object"]
categorical_features = [feature for feature in df.columns if df[feature].dtype == "object"]

print("We have {} numerical features: {}".format(len(numeric_features),numeric_features))
print("We have {} categorical features: {}".format(len(categorical_features),categorical_features))
```

The above code will use separate the numerical and categorical features and count the feature values.

```
We have 3 numerical features: ['math score', 'reading score', 'writing score']
We have 5 categorical features: ['gender', 'race/ethnicity', 'parental level of education', '
```

Exploring Data (Visualization)

Visualize Average Score Distribution to Make Some Conclusion

Histogram

Kernel Distribution Function (KDE)

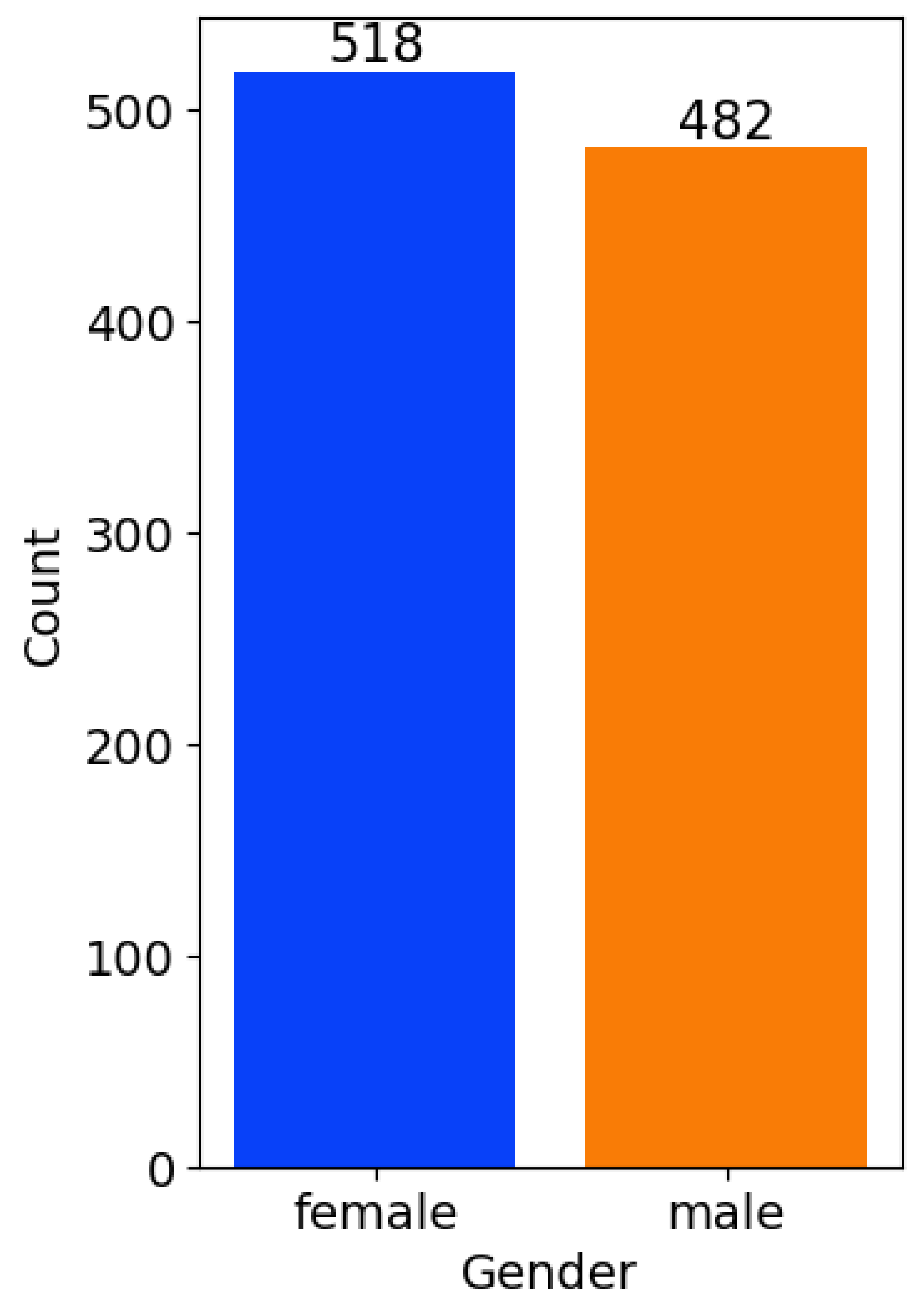
Histogram & KDE

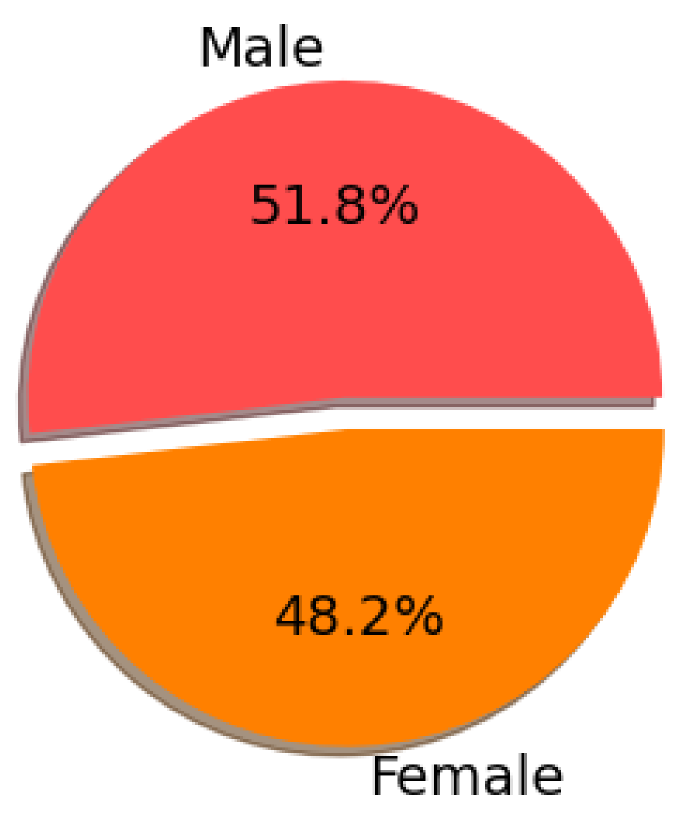
Gender Column

How is distribution of Gender?

Is gender has any impact on student's performance?

```
#Create a figure with two subplots
f,ax=plt.subplots(1,2,figsize=(8,6))


#Create a countplot of the 'gender' column and add labels to the bars
sns.countplot(x=df['gender'],data=df,palette ='bright',ax=ax[0],saturation=0.95)
for container in ax[0].containers:
    ax[0].bar_label(container,color='black',size=15)

#Set font size of x-axis and y-axis labels and tick labels
ax[0].set_xlabel('Gender', fontsize=14)
ax[0].set_ylabel('Count', fontsize=14)
ax[0].tick_params(labelsize=14)

#Create a pie chart of the 'gender' column and add labels to the slices
plt.pie(x=df['gender'].value_counts(),labels=['Male','Female'],explode=[0,0.1],autopct='%1.1f%%',shadow
=True,colors=['#ff4d4d','#ff8000'], textprops={'fontsize': 14})

#Display the plot
plt.show()
```



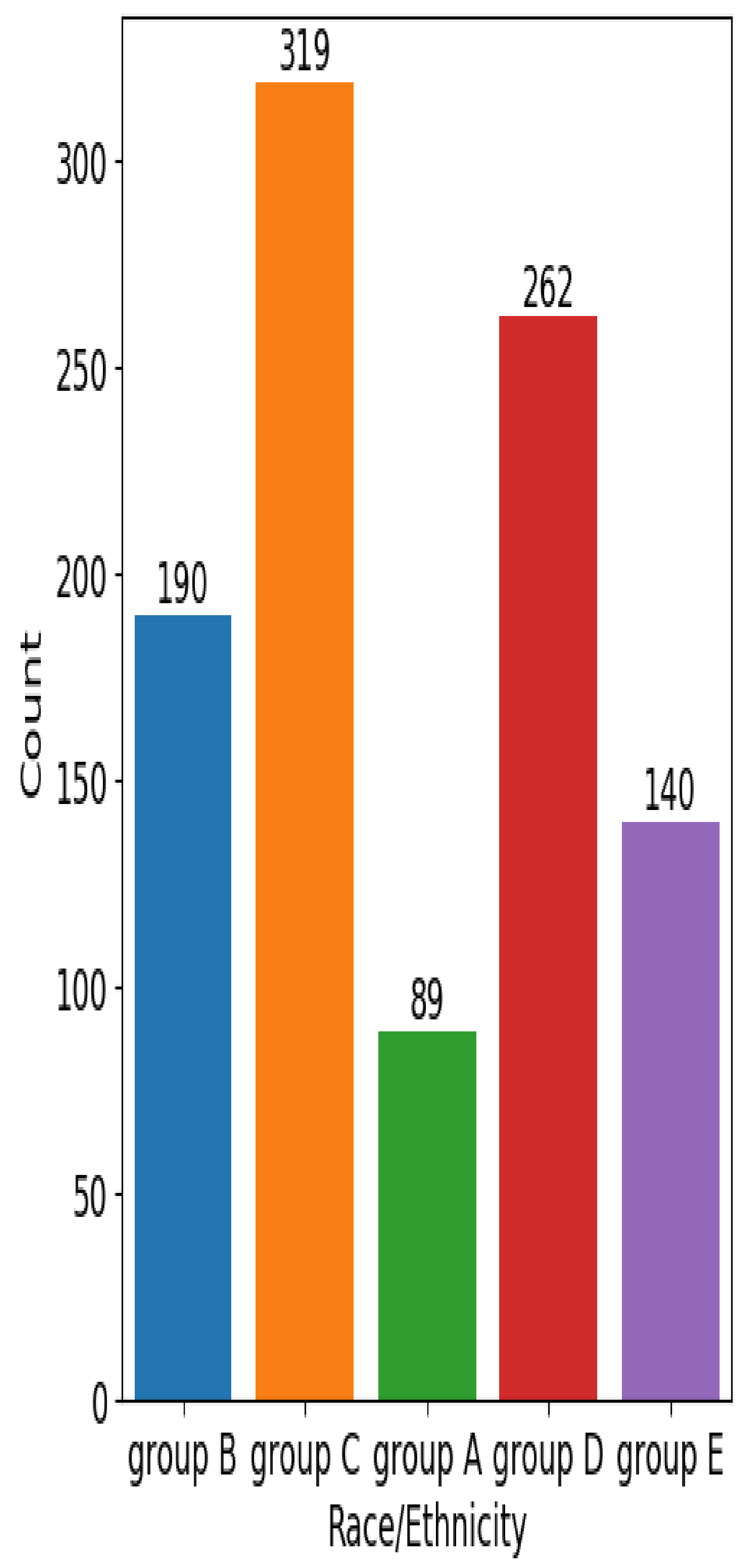


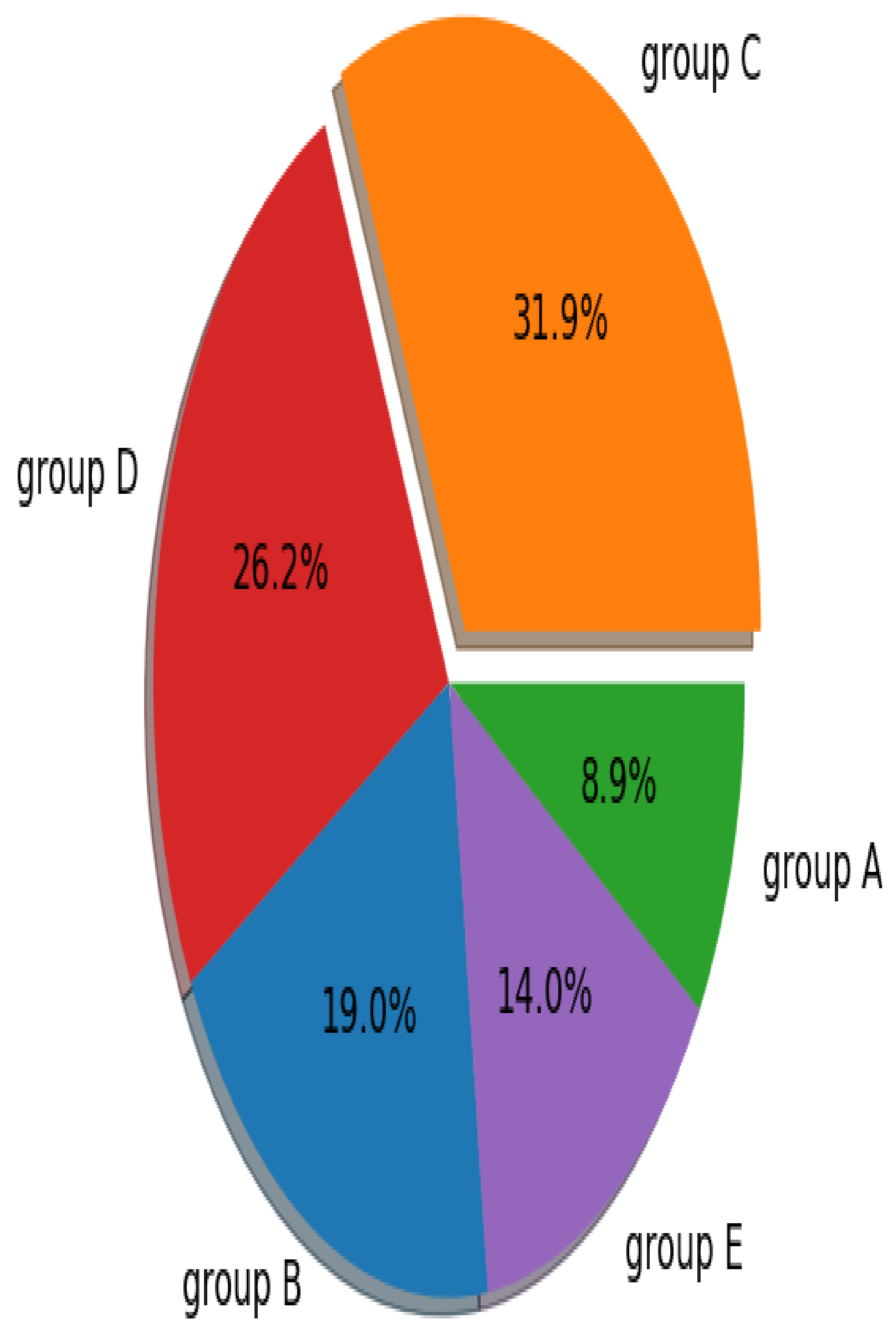
Gender has balanced data with female students are 518 (48%) and male students are 482 (52%)

Race/Ethnicity Column

```
# Define a color palette for the countplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
# blue, orange, green, red, purple are respectiively the color names for the color codes used above

# Create a figure with two subplots
```

```python
f, ax = plt.subplots(1, 2, figsize=(12, 6))

# Create a countplot of the 'race/ethnicity' column and add labels to the bars
sns.countplot(x=df['race/ethnicity'], data=df, palette=colors, ax=ax[0], saturation=0.95)
for container in ax[0].containers:
    ax[0].bar_label(container, color='black', size=14)

# Set font size of x-axis and y-axis labels and tick labels
ax[0].set_xlabel('Race/Ethnicity', fontsize=14)
ax[0].set_ylabel('Count', fontsize=14)
ax[0].tick_params(labelsize=14)

# Create a dictionary that maps category names to colors in the color palette
color_dict = dict(zip(df['race/ethnicity'].unique(), colors))

# Map the colors to the pie chart slices
pie_colors = [color_dict[race] for race in df['race/ethnicity'].value_counts().index]

# Create a pie chart of the 'race/ethnicity' column and add labels to the slices
plt.pie( x=df['race/ethnicity'].value_counts(), labels=df['race/ethnicity'].value_counts().index,
explode=[0.1, 0, 0, 0, 0], autopct='%1.1f%%', shadow=True, colors=pie_colors, textprops={'fontsize': 14})

# Set the aspect ratio of the pie chart to 'equal' to make it a circle
plt.axis('equal')

# Display the plot
plt.show()
```





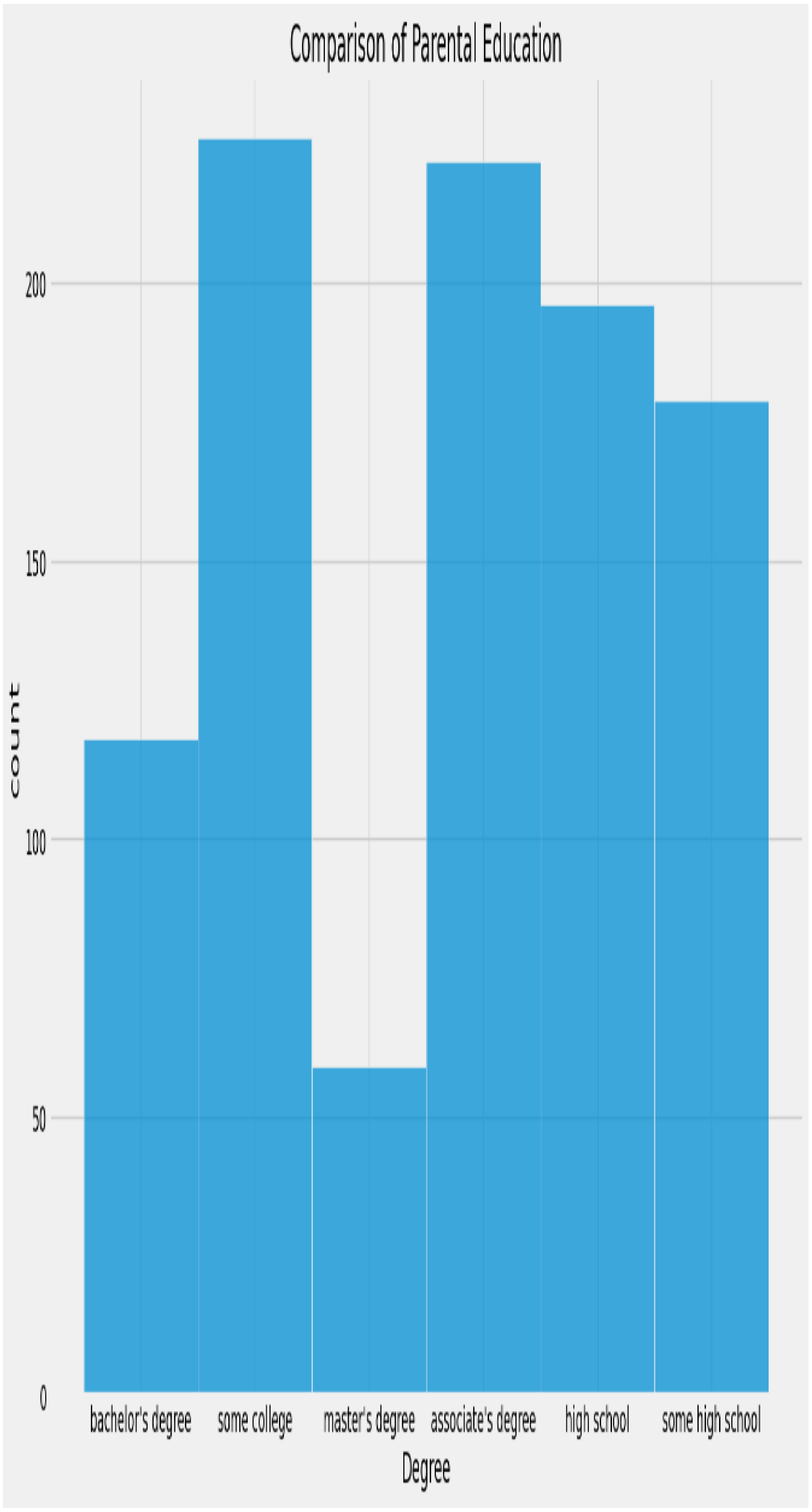
id = Insights>Insights

Most of the student belonging from group C /group D.

Lowest number of students belong to group A.

Parental Level of Education Column

```
plt.rcParams['figure.figsize'] = (15, 9)
plt.style.use('fivethirtyeight')
sns.histplot(df["parental level of education"], palette = 'Blues')
plt.title('Comparison of Parental Education', fontweight = 30, fontsize = 20)
plt.xlabel('Degree')
plt.ylabel('count')
plt.show()
```

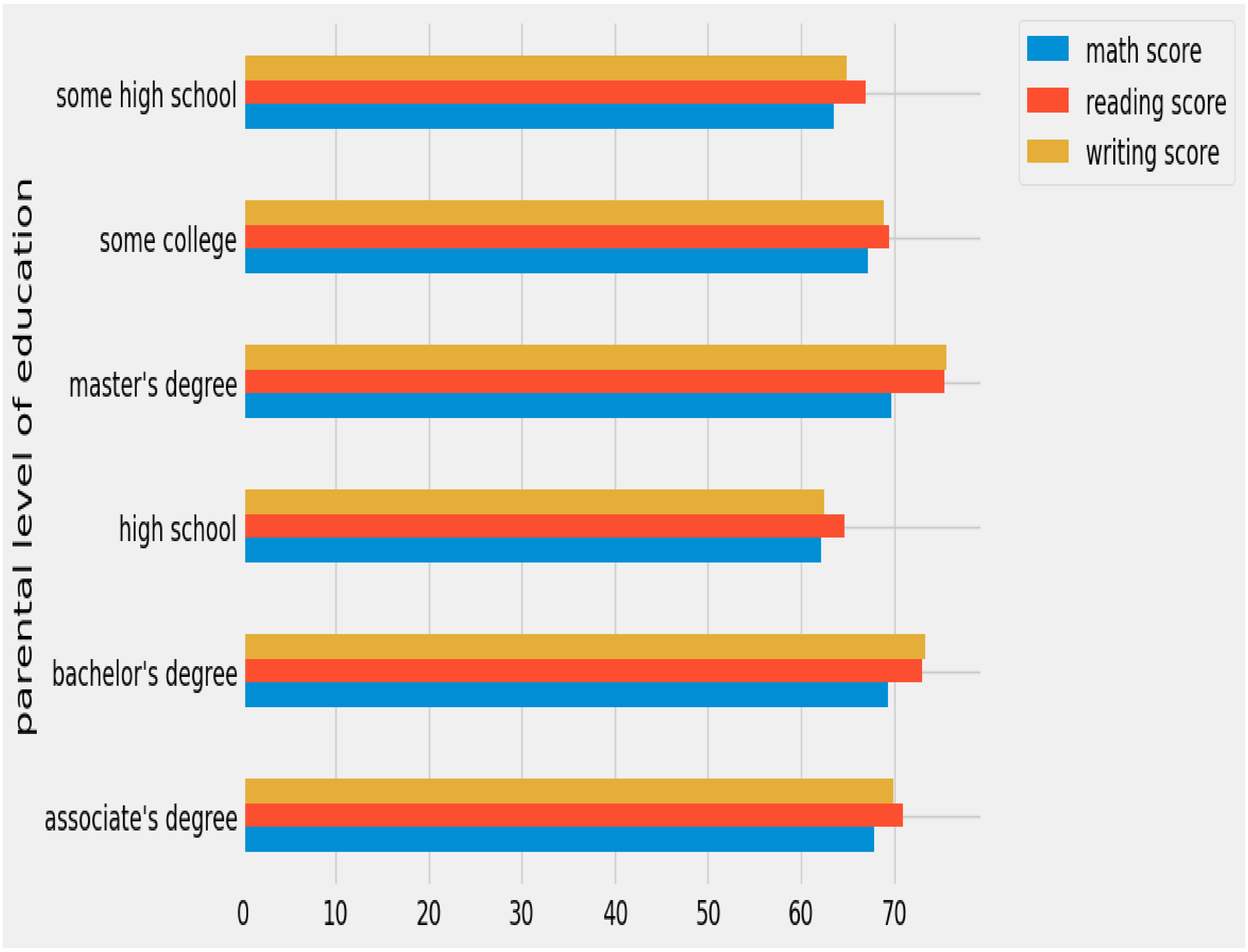

Comparison of Parental Education
count
0
50
100
150
200
bachelor's degree
some college
master's degree
associate's degree
high school
some high school
Degree

id = Insights>Insights

Largest number of parents are from college.

Bivariate Analysis

```
df.groupby('parental level of education').agg('mean').plot(kind='barh',figsize=(10,10))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

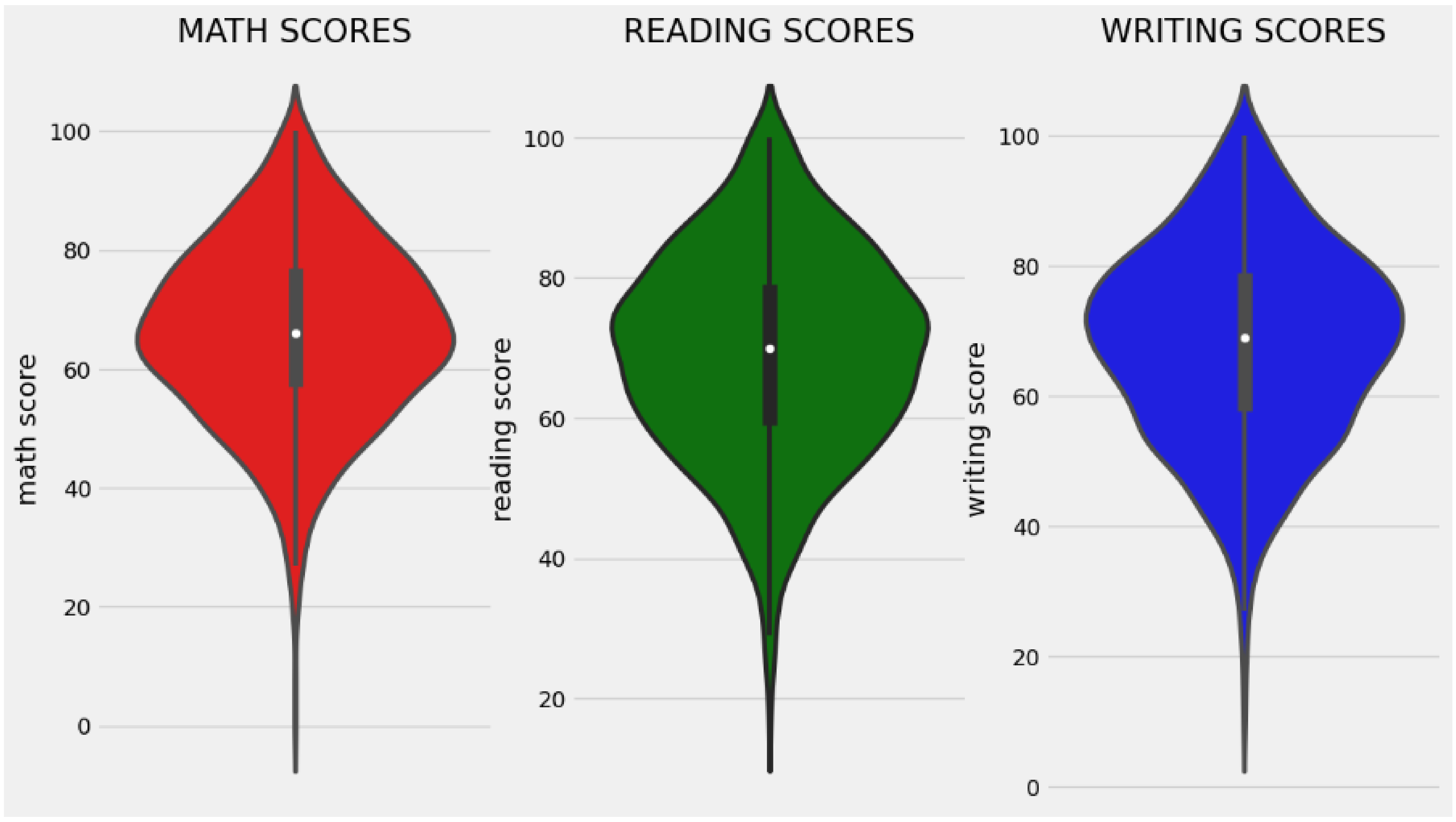


id = Insights>Insights

The score of student whose parents possess master and bachelor level education are higher than others.

Maximum Score of Students in All Three Subjects

```
plt.Fig
figsize=(18,8))
plt.subplot(1, 4, 1)
plt.title('MATH SCORES')
sns.violinplot(y='math score',data=df,color='red',linewidth=3) plt.subplot(1, 4, 2) plt.title('READING
SCORES') sns.vi

plot(y='reading score',data=df,color='green',linewidth=3) plt.subplot(1, 4, 3) plt.title('WRITING
SCORES') sns.violinplot(y='writing score',data=df,color='blue',linewidth=3) plt.show()
```



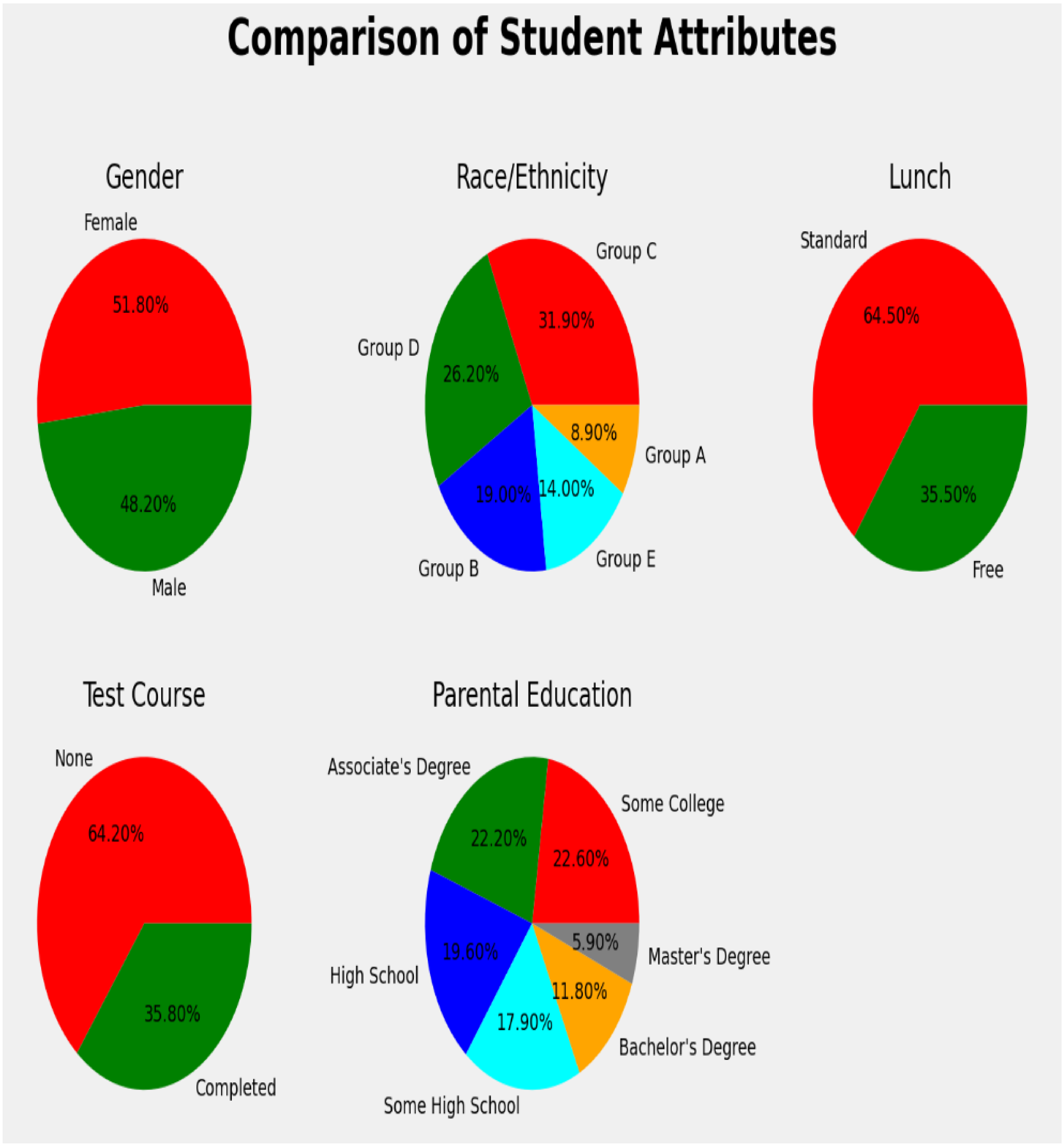
id = Insights>Insights

From the above three plots its clearly visible that most of the students score in between 60-80 in Maths whereas in reading and writing most of them score from 50-80.

Multivariate Analysis Using Pie Plot

```
#Set Figure size
plt.rcParams['figure.figsize'] = (12, 9)
```

```
# First row of pie charts
plt.subplot(2, 3, 1)
size = df['gender'].value_counts()
labels = 'Female', 'Male'
color = ['red','green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Gender', fontsize=20)
plt.axis('off')

plt.subplot(2, 3, 2)
size = df['race/ethnicity'].value_counts()
labels = 'Group C', 'Group D', 'Group B', 'Group E', 'Group A'
color = ['red', 'green', 'blue', 'cyan', 'orange']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Race/Ethnicity', fontsize=20)
plt.axis('off')

plt.subplot(2, 3, 3)
size = df['lunch'].value_counts()
labels = 'Standard', 'Free'
color = ['red', 'green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Lunch', fontsize=20)
plt.axis('off')

# Second row of pie charts
plt.subplot(2, 3, 4)
size = df['test preparation course'].value_counts()
labels = 'None', 'Completed'
color = ['red', 'green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Test Course', fontsize=20)
plt.axis('off')
plt.subplot(2, 3, 5)
size = df['parental level of education'].value_counts()
labels = 'Some College', "Associate's Degree", 'High School', 'Some High School', "Bachelor's Degree",
"Master's Degree"
color = ['red', 'green', 'blue', 'cyan', 'orange', 'grey']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Parental Education', fontsize=20)
plt.axi
ff') # Remove extra subplot plt.subplot(2, 3, 6).remove() # Add super title plt.suptitle('Comparison of
Student Attributes', fontsize=20, fontweight='bold') # Adjust layout and show plot # This is removed as
there are only 5 subplots in this figure and we want to arrange them in a 2x3 grid. # Since there is no
6th subplot, it is removed to avoid an empty subplot being shown in the figure. plt.tight_layout()
plt.subplots_adjust(top=0.85) plt.show()
```


Comparison of Student Attributes
Gender
Female
51.80%
48.20%
Male
Race/Ethnicity
Group C
31.90%
Group D
26.20%
8.90%
Group A
19.00%
14.00%
Group B
Group E
Lunch
Standard
64.50%
35.50%
Free
Test Course
None
64.20%
35.80%
Completed
Parental Education
Associate's Degree
22.20%
Some College
22.60%
5.90%
Master's Degree
19.60%
High School
11.80%
17.90%
Bachelor's Degree
Some High School

id = Insights>Insights

The number of Male and Female students is almost equal.

The number of students is higher in Group C.

The number of students who have standard lunch is greater.

The number of students who have not enrolled in any test preparation course is greater.

The number of students whose parental education is "Some College" is greater followed closely by "Associate's Degree".

From the above plot, it is clear that all the scores increase linearly with each other.

Student's Performance is related to lunch, race, and parental level education.

Females lead in pass percentage and also are top-scorers.

Student Performance is not much related to test preparation course.

The finishing preparation course is beneficial.

Model Training

Import Data and Required Packages

Importing scikit library algorithms to import regression algorithms.

```
# Modelling
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection imp
RandomizedSearchCV from catboost import CatBoostRegressor from xgboost import XGBRegressor import
warnings
```

Splitting the X and Y Variables

This separation of the dependent variable(y) and independent variables(X) is one the most important in our project we use the math score as a dependent variable. Because so many students lack in math subjects it will almost 60% to 70% of students in classes 7-10 students are fear of math subjects that's why I am choosing the math score as a dependent score.

It will use to improve the percentage of math scores and increase the grad F students and also remove fear in math.

```
X = df.drop(columns="math score",axis=1)
y = df["math score"]
```

Create Column Transformer with 3 Types of Transformers

```
num_features = X.select_dtypes(exclude="object").columns
cat_features = X.select_dtypes(include="object").columns

from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder()

preprocessor = Column
transformer( [ ("OneHotEncoder", oh_transformer, cat_features), ("StandardScaler",
numeric_transformer, num_features), ] ) X = preprocessor.fit_transform(X)
```

Separate Dataset into Train and Test

To separate the dataset into train and test to identify the training size and testing size of the dataset.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape
```

```
((800, 19), (200, 19))
```

Create an Evaluate Function for Model Training

This function is use to evaluate the model and build a good model.

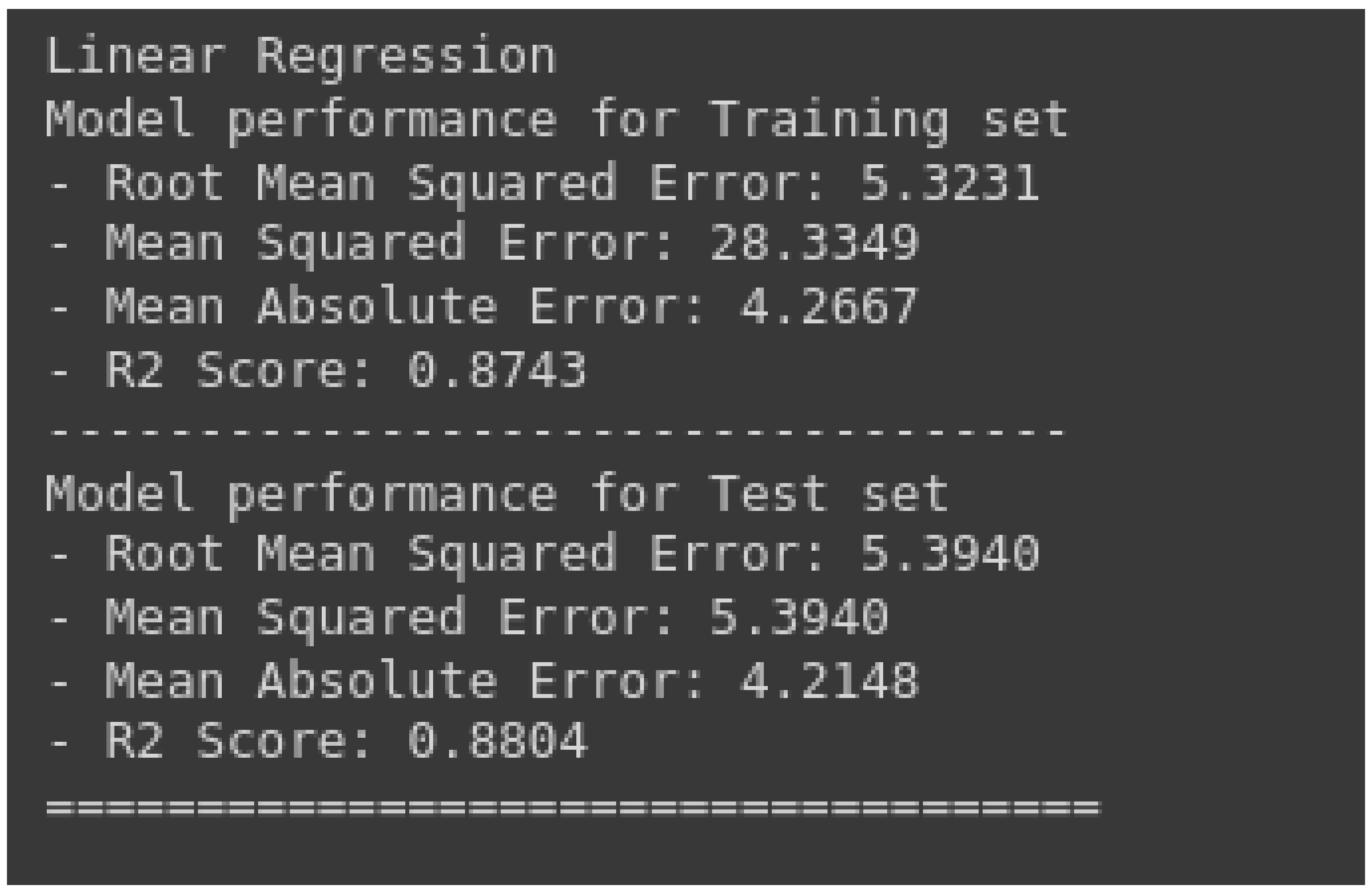
```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

To create a models variable and form a dictionary formate.

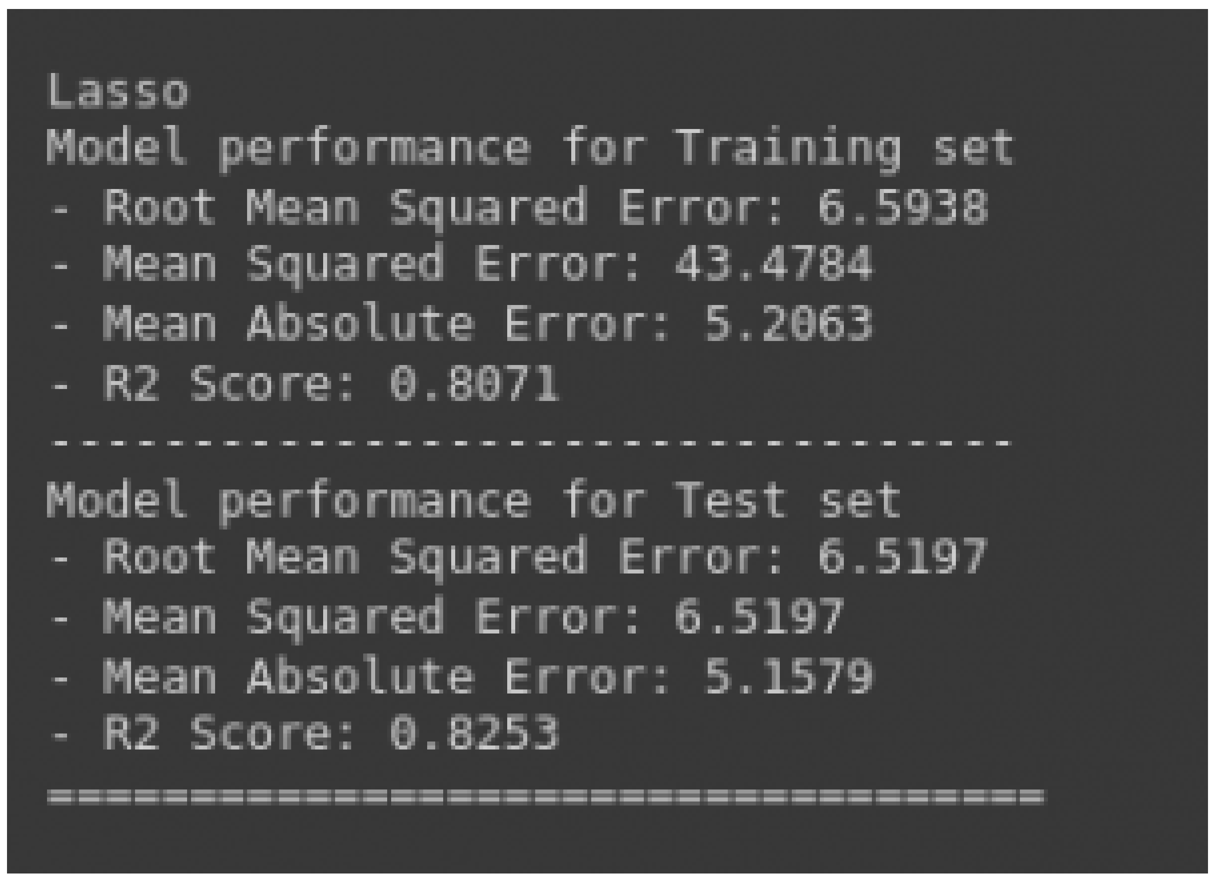
```
models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "XGBRegressor": XGBRegressor(),
    "CatBoosting Regressor": CatBoostRegressor(verbose=False),
    "AdaBoost Regressor": AdaBoostRegressor()
}
model_list = []
r2_list =[]
```

```
for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate Train and Test dataset
    model_train_mae, model_train_mse, model_train_rmse, model_train_r2 = evaluate_model(y_train,
 y_train_pred)

    model_test_mae, model_test_mse, model_test_rmse, model_test_r2 = evaluate_model(y_test,
y_test_pred)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Squared Error: {:.4f}".format(model_train_mse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')
    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".
    for

model_test_r2)) r2_list.append(model_test_r2) print('='*35) print('\n')
```

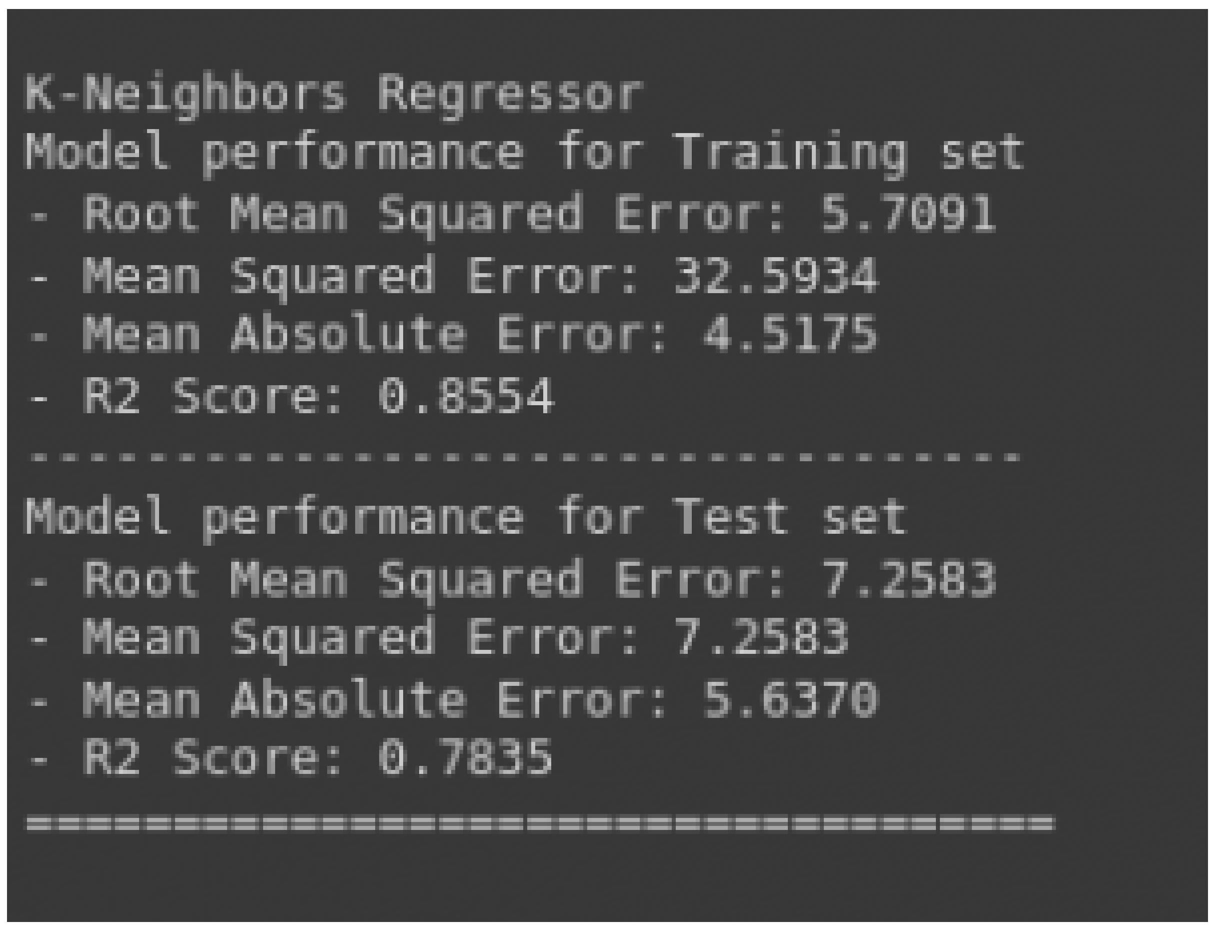
The output of before tuning all algorithms' hyperparameters. And it provides the RMSE, MSE, MAE, and R2 score values for training and test data.



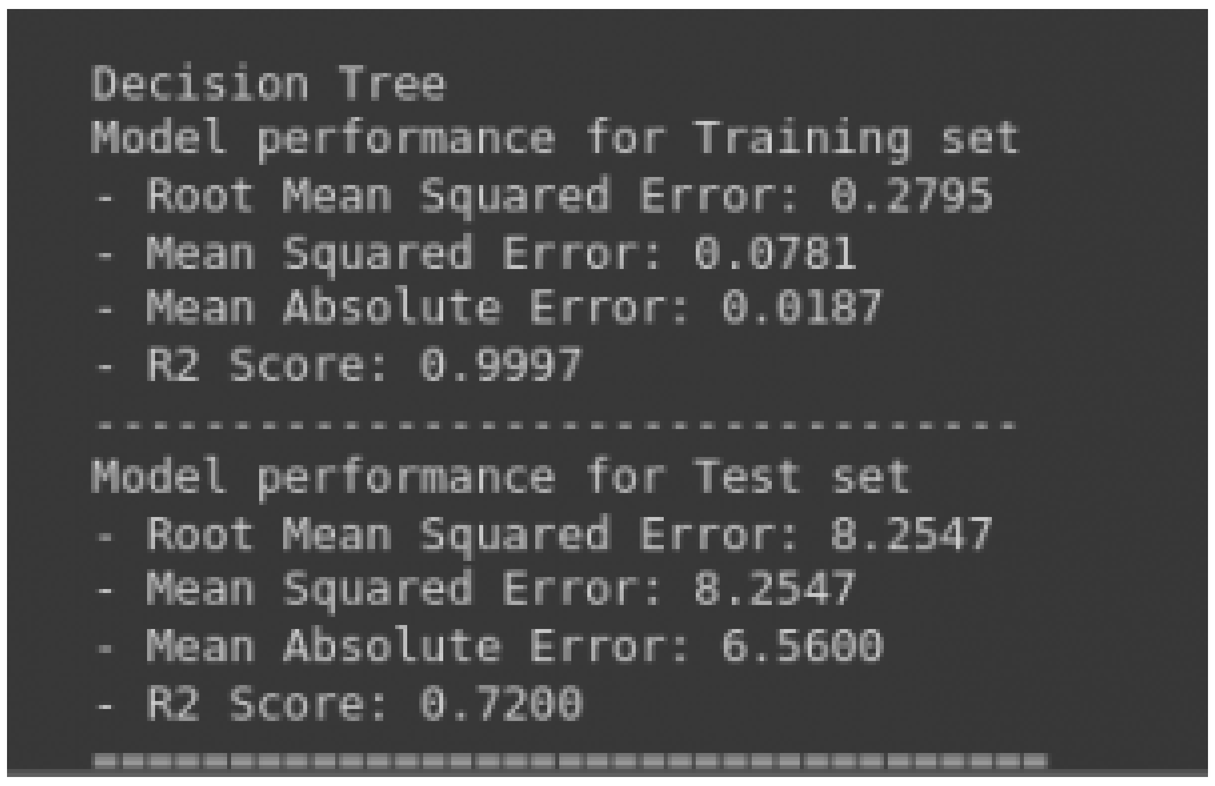
```
Linear Regression
Model performance for Training set
- Root Mean Squared Error: 5.3231
- Mean Squared Error: 28.3349
- Mean Absolute Error: 4.2667
- R2 Score: 0.8743
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3940
- Mean Squared Error: 5.3940
- Mean Absolute Error: 4.2148
- R2 Score: 0.8804
===================================
```



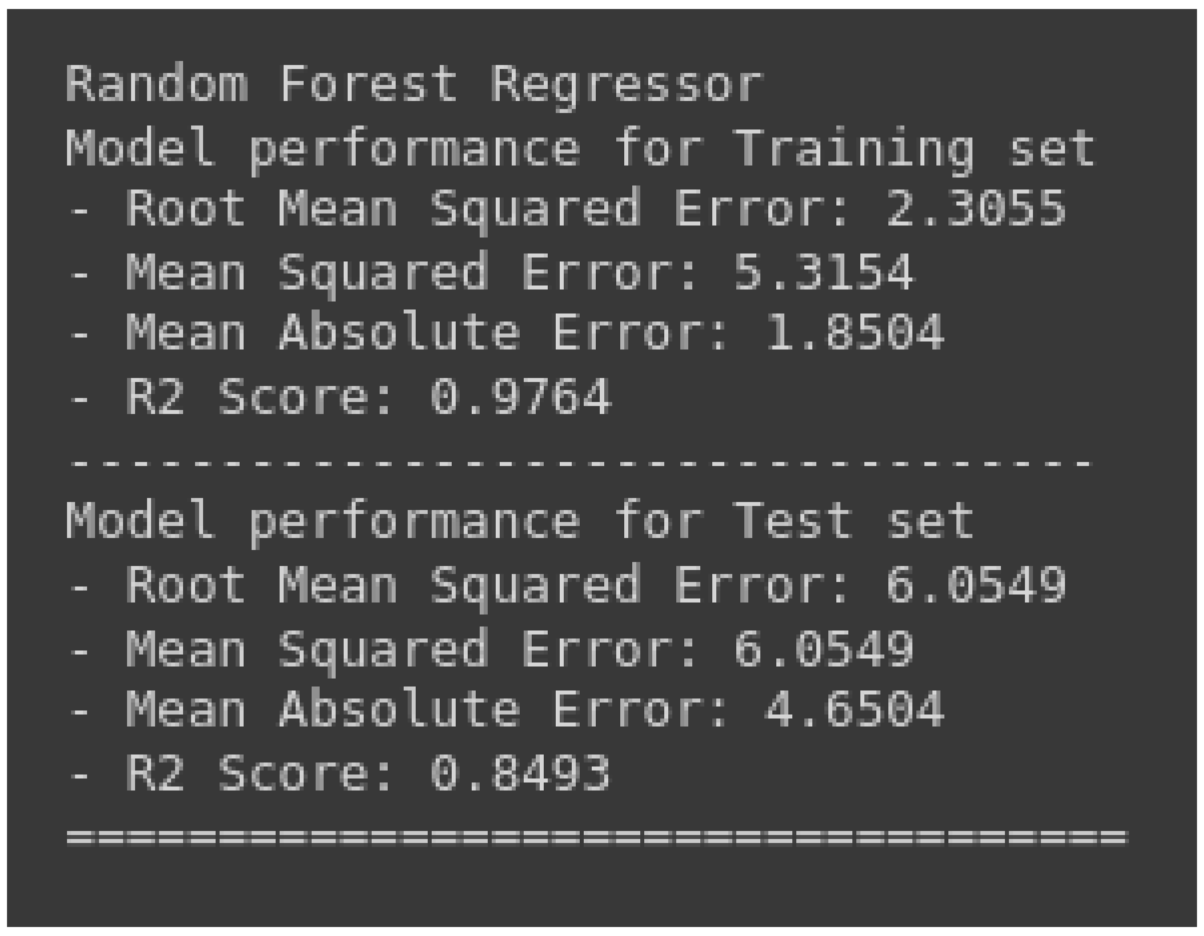
```
Lasso
Model performance for Training set
- Root Mean Squared Error: 6.5938
- Mean Squared Error: 43.4784
- Mean Absolute Error: 5.2063
- R2 Score: 0.8071
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.5197
- Mean Squared Error: 6.5197
- Mean Absolute Error: 5.1579
- R2 Score: 0.8253
===================================
```



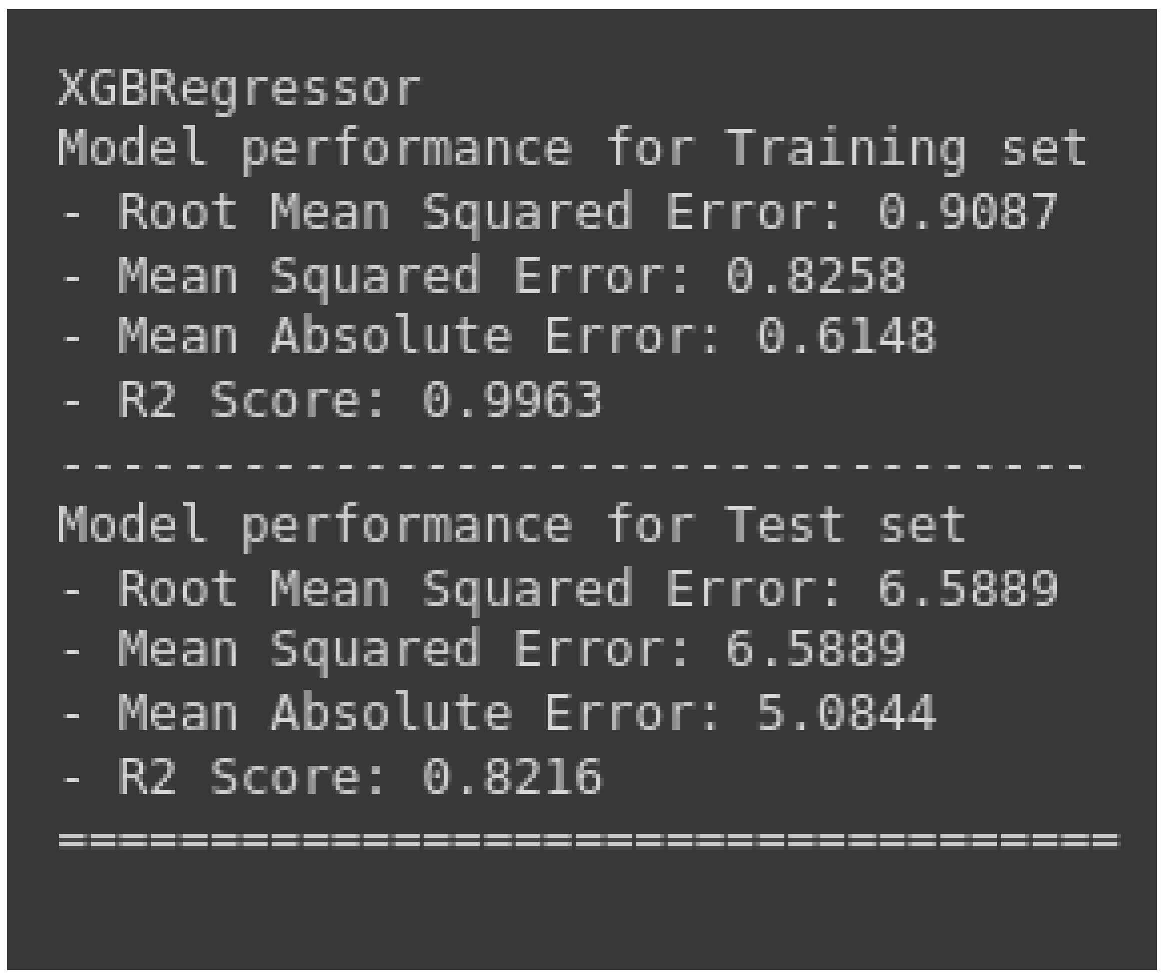
```
K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 5.7091
- Mean Squared Error: 32.5934
- Mean Absolute Error: 4.5175
- R2 Score: 0.8554
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 7.2583
- Mean Squared Error: 7.2583
- Mean Absolute Error: 5.6370
- R2 Score: 0.7835
===================================
```



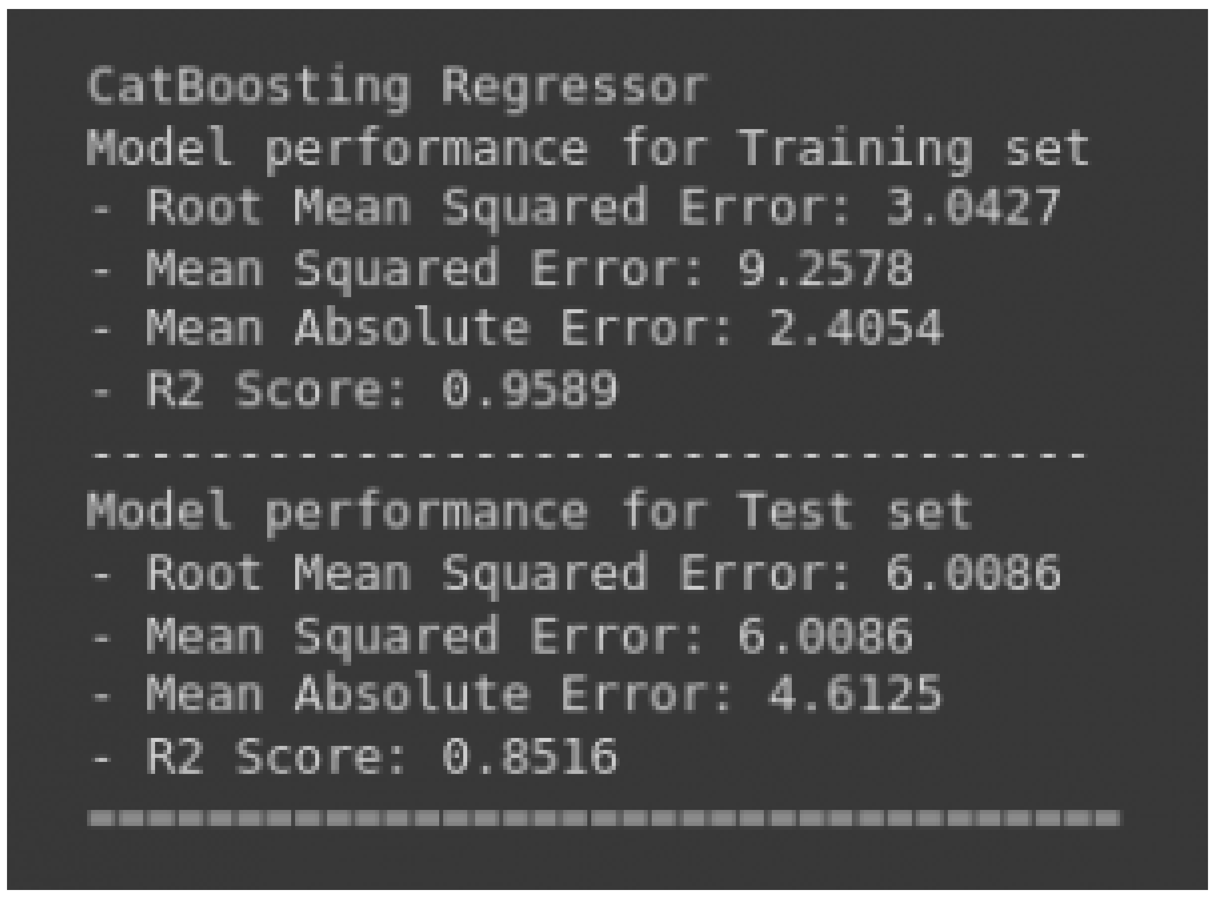
```
Decision Tree
Model performance for Training set
- Root Mean Squared Error: 0.2795
- Mean Squared Error: 0.0781
- Mean Absolute Error: 0.0187
- R2 Score: 0.9997
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 8.2547
- Mean Squared Error: 8.2547
- Mean Absolute Error: 6.5600
- R2 Score: 0.7200
===================================
```



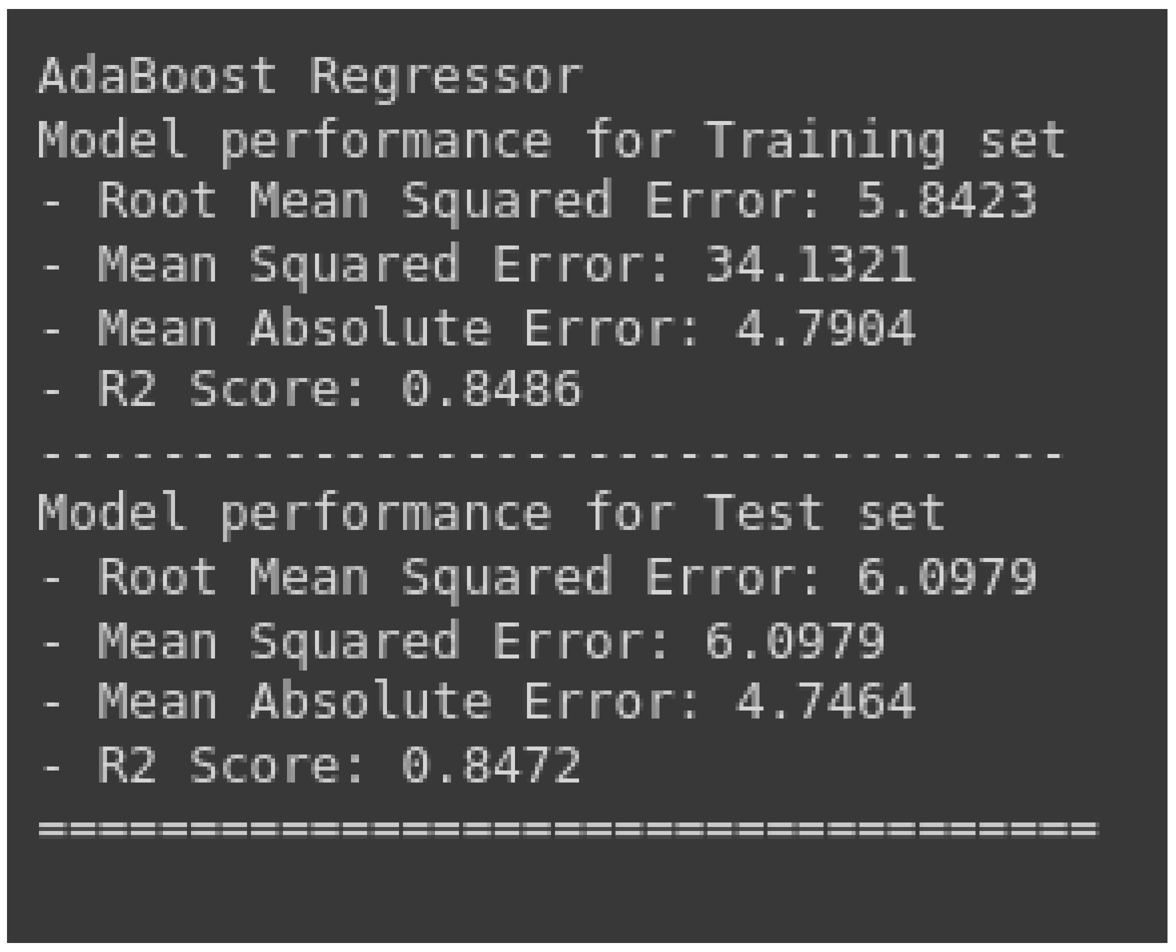
```
Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 2.3055
- Mean Squared Error: 5.3154
- Mean Absolute Error: 1.8504
- R2 Score: 0.9764
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.0549
- Mean Squared Error: 6.0549
- Mean Absolute Error: 4.6504
- R2 Score: 0.8493
===================================
```



```
XGBRegressor
Model performance for Training set
- Root Mean Squared Error: 0.9087
- Mean Squared Error: 0.8258
- Mean Absolute Error: 0.6148
- R2 Score: 0.9963
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.5889
- Mean Squared Error: 6.5889
- Mean Absolute Error: 5.0844
- R2 Score: 0.8216
===================================
```



```
CatBoosting Regressor
Model performance for Training set
- Root Mean Squared Error: 3.0427
- Mean Squared Error: 9.2578
- Mean Absolute Error: 2.4054
- R2 Score: 0.9589
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.0086
- Mean Squared Error: 6.0086
- Mean Absolute Error: 4.6125
- R2 Score: 0.8516
===================================
```



```
AdaBoost Regressor
Model performance for Training set
- Root Mean Squared Error: 5.8423
- Mean Squared Error: 34.1321
- Mean Absolute Error: 4.7904
- R2 Score: 0.8486
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.0979
- Mean Squared Error: 6.0979
- Mean Absolute Error: 4.7464
- R2 Score: 0.8472
===================================
```

Hyperparameter Tuning

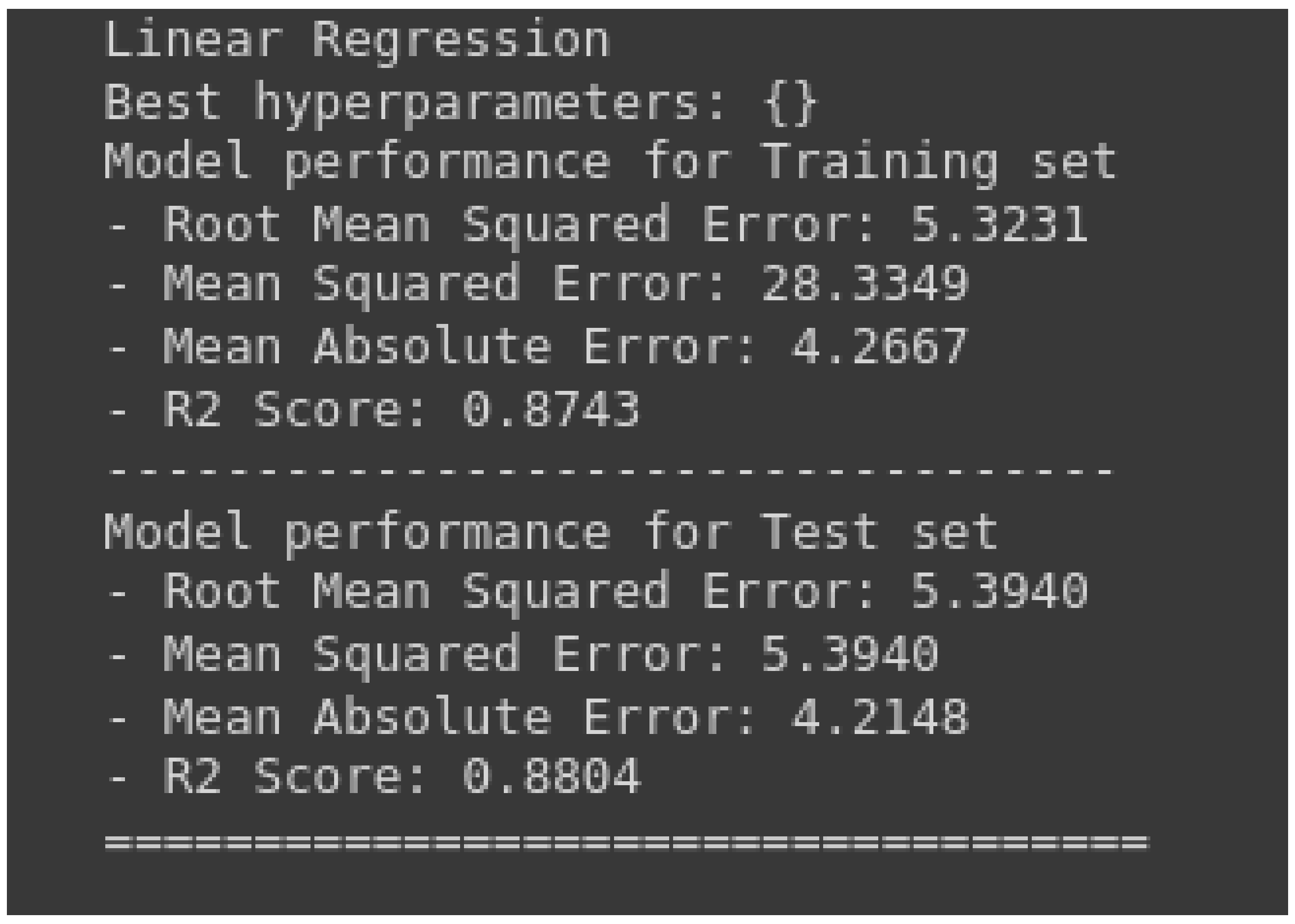
It will give the model with most accurate predictions and improve prediction accuracy.

This will give the optimized value of hyperparameters, which maximize your model predictive accuracy.

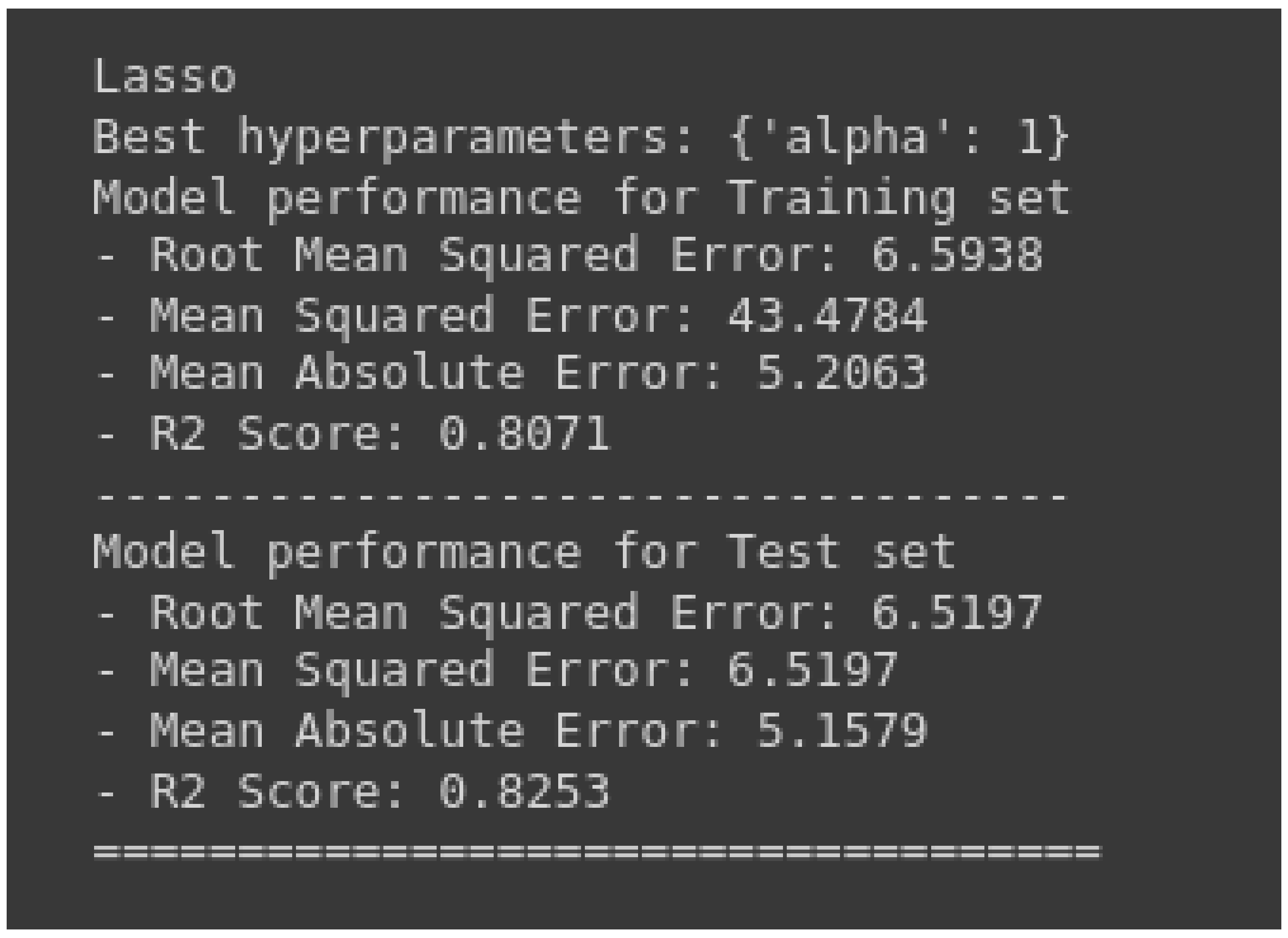
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import make_scorer

# Define hyperparameter ranges for each model
param_grid = {
    "Linear Regression": {},
    "Lasso": {"alpha": [1]},
    "K-Neighbors Regressor": {"n_neighbors": [3, 5, 7],},
    "Decision Tree": {"max_depth": [3, 5, 7],'criterion':['squared_error', 'friedman_mse',
'absolute_error', 'poisson']},
    "Random Forest Regressor":{'n_estimators': [8,16,32,64,128,256], "max_depth": [3, 5, 7]},
    "Gradient Boosting": {'learning_rate':[.1,.01,.05,.001],'subsample':[0.6,0.7,0.75,0.8,0.85,0.9],
                         'n_estimators': [8,16,32,64,128,256]},
    "XGBRegressor":{'depth': [6,8,10],'learning_rate': [0.01, 0.05, 0.1],'iterations': [30, 50, 100]},
    "CatBoosting Regressor":{"iterations": [100, 500], "depth": [3, 5, 7]},
    "AdaBoost Regressor": {'learning_rate':[.1,.01,0.5,.001],'n_estimators': [8,16,32,64,128,256]}
}

model_list = []
r2_list =[]

for model_name, model in models.items():
    # Create a scorer object to use in grid search
    scorer = make_scorer(r2_score)

    # Perform grid search to find the best hyperparameters
    grid_search = GridSearchCV(
        model,
        param_grid[model_name],
        scoring=scorer,
        cv=5,
        n_jobs=-1
    )

    # Train the model with the best hyperparameters
rid_search.fit(X_train, y_train) # Make predictions y_train_pred = grid_search.predict(X_train)
y_test_pred = grid_search.predict(X_test) # Evaluate Train and Test dataset model_train_mae,
model_train_mse, model_train_rmse, model_train_r2 = evaluate_model(y_train, y_train_pred)
model_test_mae, model_test_mse, model_test_rmse, model_test_r2 = evaluate_model(y_test,
y_test_pred) print(model_name) model_list.append(model_name) print('Best hyperparameters:',
grid_search.best_params_) print('Model performance for Training set') print("- Root Mean Squared
Error: {:.4f}".format(model_train_rmse)) print("- Mean Squared Error:
{:.4f}".format(model_train_mse)) print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
print("- R2 Score: {:.4f}".format(model_train_r2)) print('----------------------------
------') print('Model performance for Test set') print("- Root Mean Squared Error:
{:.4f}".format(model_test_rmse)) print("- Mean Squared Error: {:.4f}".format(model_test_rmse))
print("- Mean Absolute Error: {:.4f}".format(model_test_mae)) print("- R2 Score: {:.4f}".for

model_test_r2)) r2_list.append(model_test_r2) print('='*35) print('\n')
```

6.RESULTS

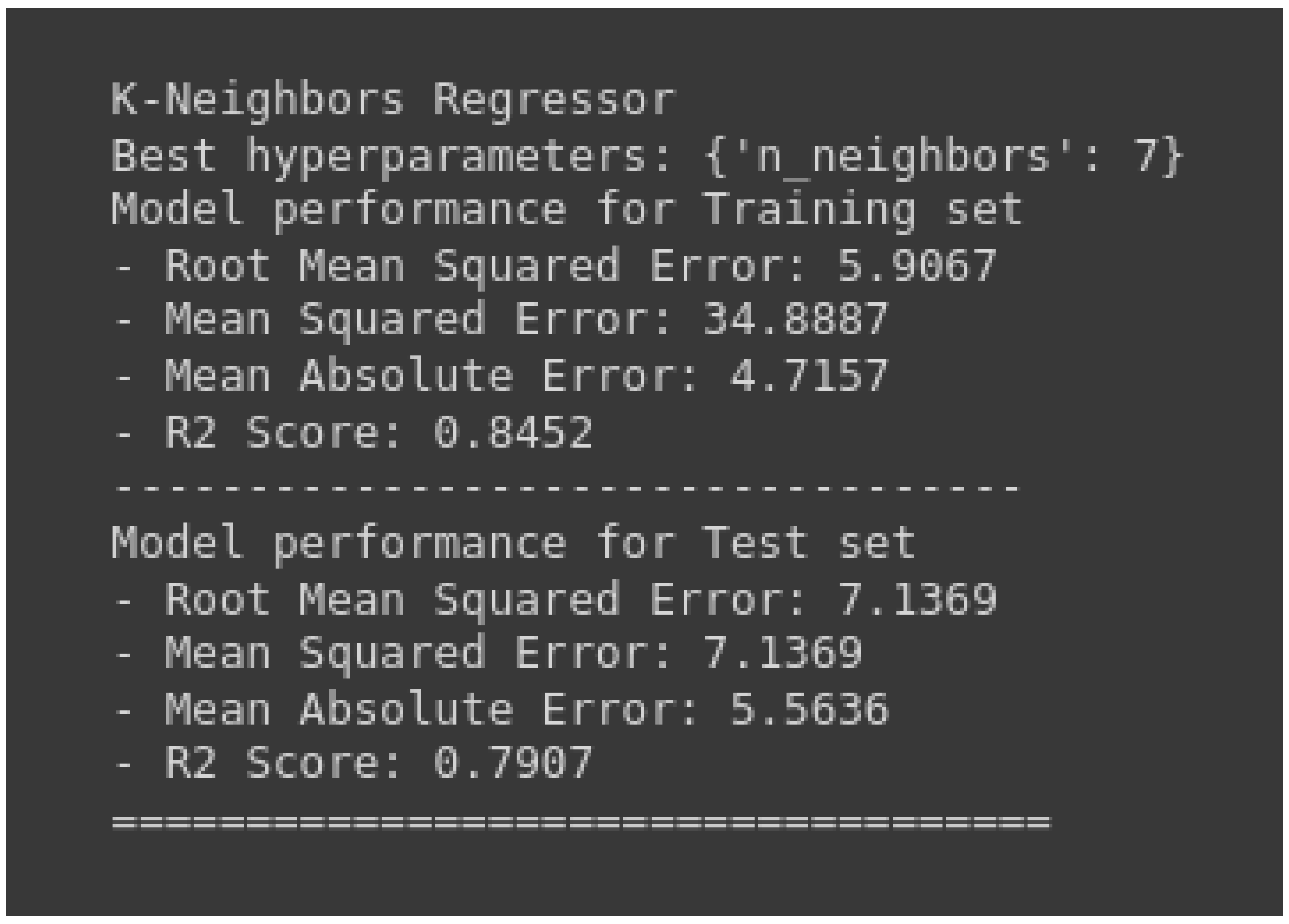
The output of after tuning all algorithms' hyperparameters. And it provides the RMSE, MSE, MAE, and R2 score values for training and test data.



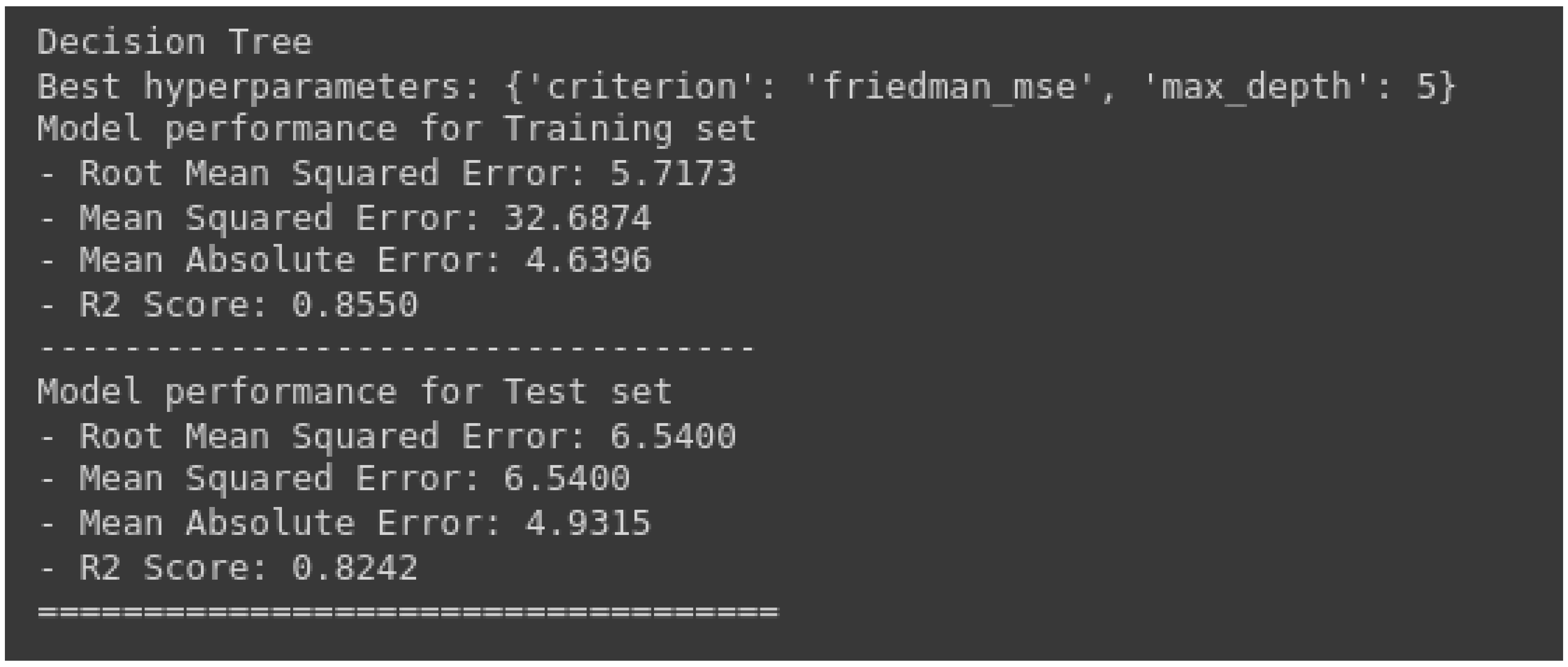
```
Linear Regression
Best hyperparameters: {}
Model performance for Training set
- Root Mean Squared Error: 5.3231
- Mean Squared Error: 28.3349
- Mean Absolute Error: 4.2667
- R2 Score: 0.8743
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3940
- Mean Squared Error: 5.3940
- Mean Absolute Error: 4.2148
- R2 Score: 0.8804
===================================
```



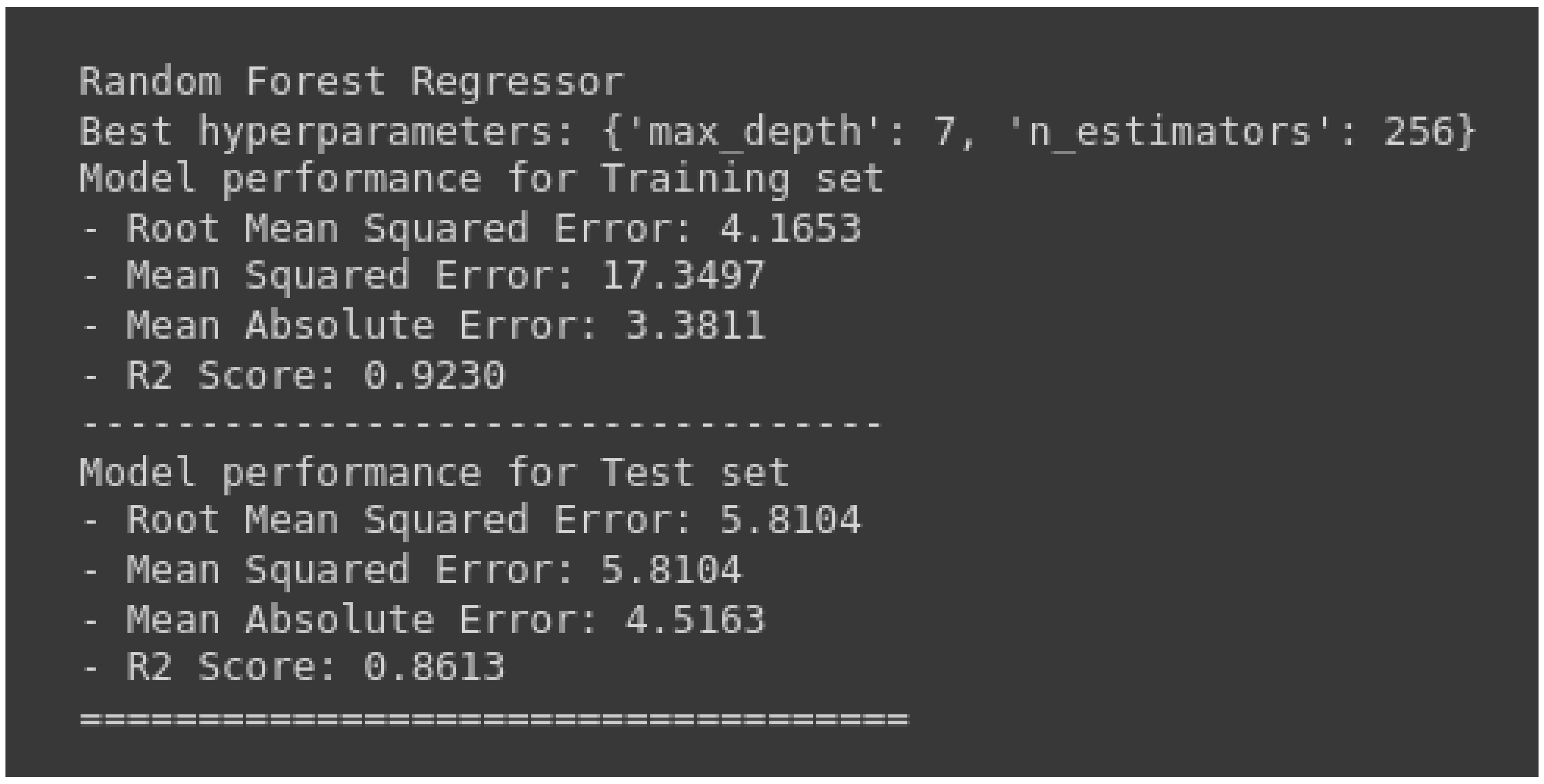
```
Lasso
Best hyperparameters: {'alpha': 1}
Model performance for Training set
- Root Mean Squared Error: 6.5938
- Mean Squared Error: 43.4784
- Mean Absolute Error: 5.2063
- R2 Score: 0.8071
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.5197
- Mean Squared Error: 6.5197
- Mean Absolute Error: 5.1579
- R2 Score: 0.8253
===================================
```



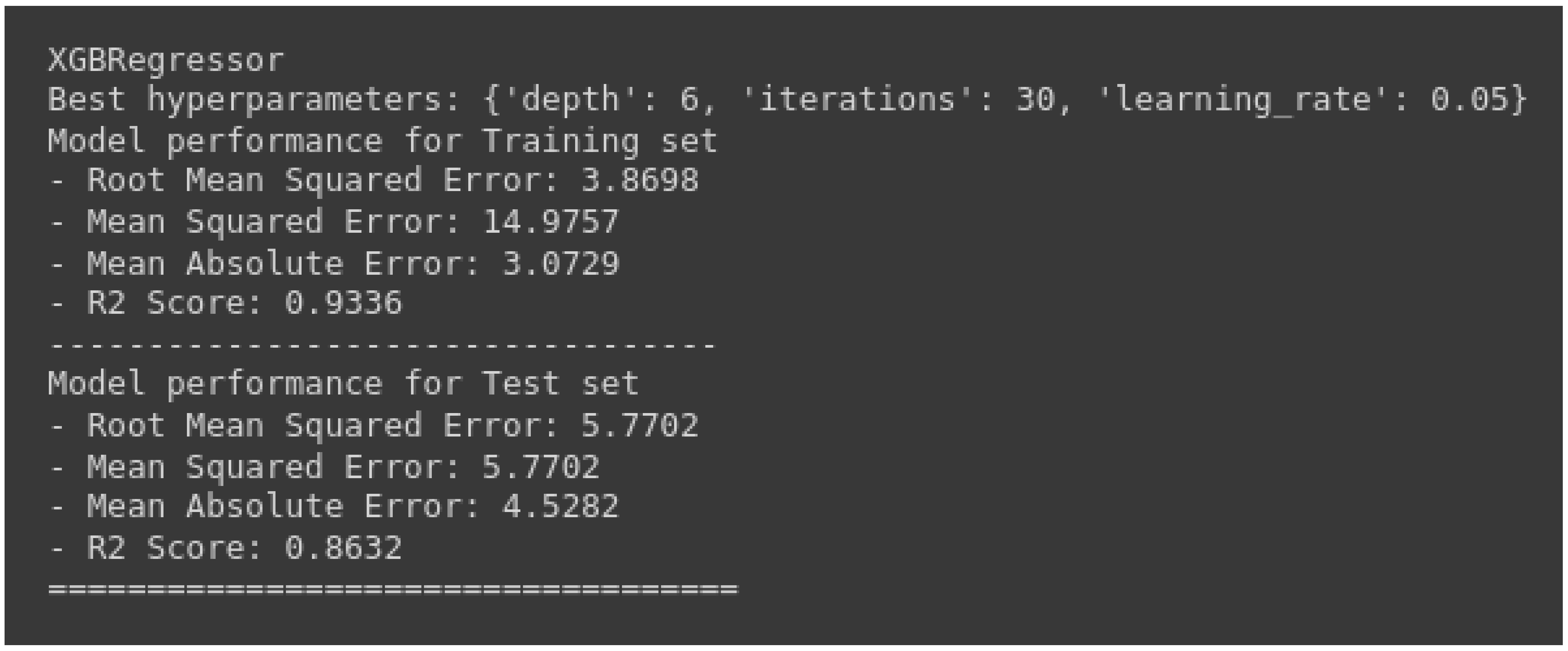
```
K-Neighbors Regressor
Best hyperparameters: {'n_neighbors': 7}
Model performance for Training set
- Root Mean Squared Error: 5.9067
- Mean Squared Error: 34.8887
- Mean Absolute Error: 4.7157
- R2 Score: 0.8452
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 7.1369
- Mean Squared Error: 7.1369
- Mean Absolute Error: 5.5636
- R2 Score: 0.7907
===================================
```



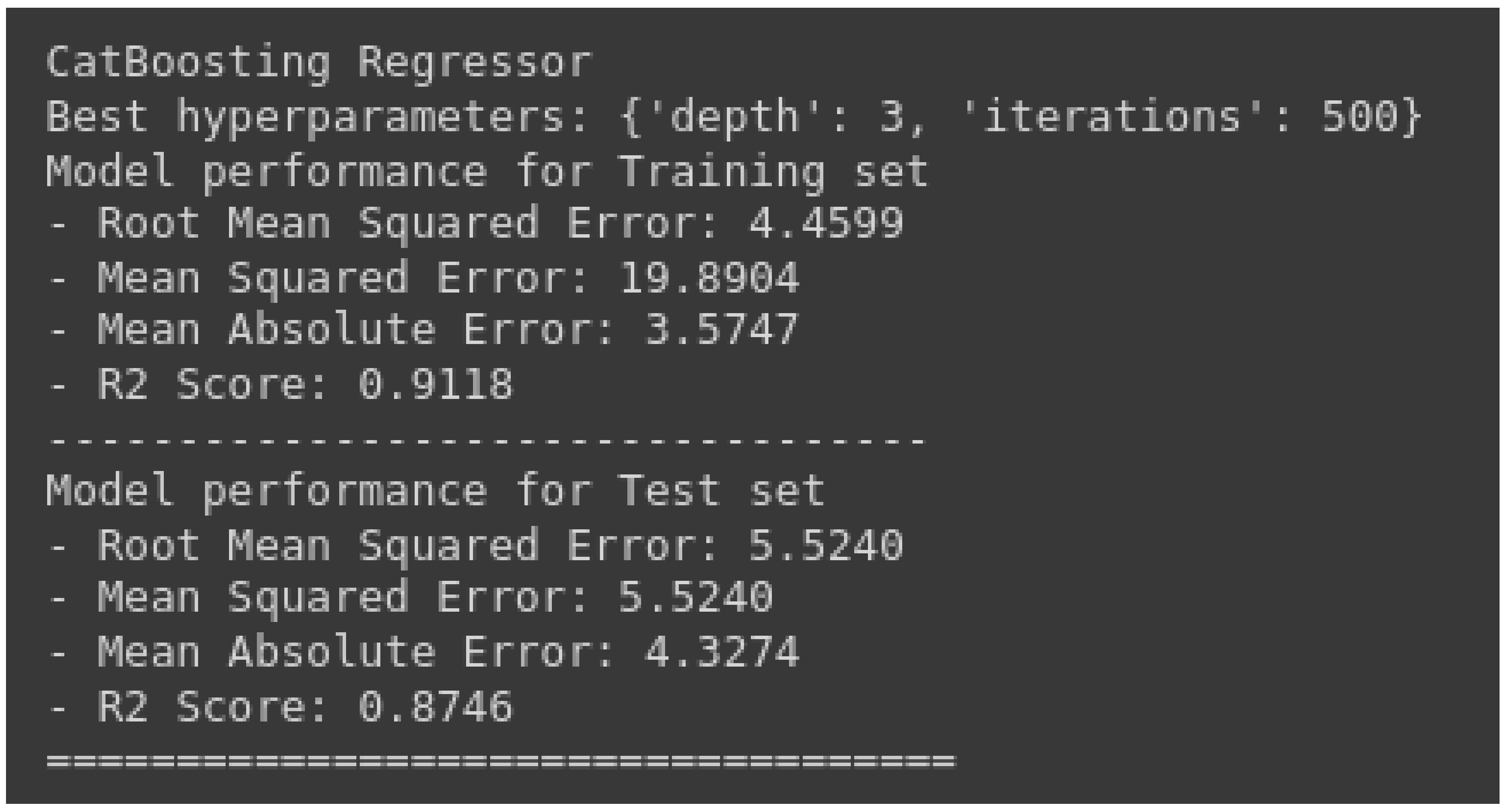
```
Decision Tree
Best hyperparameters: {'criterion': 'friedman_mse', 'max_depth': 5}
Model performance for Training set
- Root Mean Squared Error: 5.7173
- Mean Squared Error: 32.6874
- Mean Absolute Error: 4.6396
- R2 Score: 0.8550
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.5400
- Mean Squared Error: 6.5400
- Mean Absolute Error: 4.9315
- R2 Score: 0.8242
===================================
```



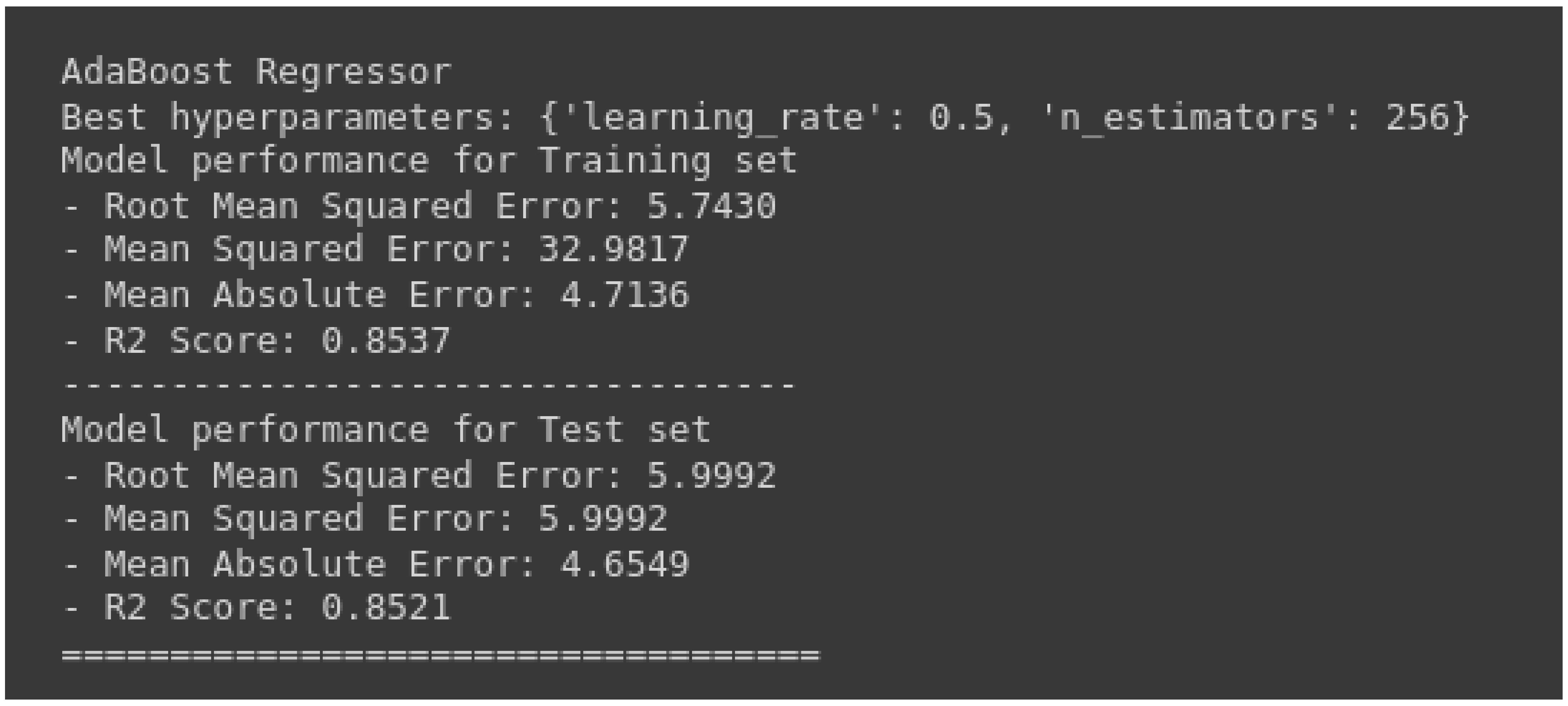
```
Random Forest Regressor
Best hyperparameters: {'max_depth': 7, 'n_estimators': 256}
Model performance for Training set
- Root Mean Squared Error: 4.1653
- Mean Squared Error: 17.3497
- Mean Absolute Error: 3.3811
- R2 Score: 0.9230
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.8104
- Mean Squared Error: 5.8104
- Mean Absolute Error: 4.5163
- R2 Score: 0.8613
===================================
```



```
XGBRegressor
Best hyperparameters: {'depth': 6, 'iterations': 30, 'learning_rate': 0.05}
Model performance for Training set
- Root Mean Squared Error: 3.8698
- Mean Squared Error: 14.9757
- Mean Absolute Error: 3.0729
- R2 Score: 0.9336
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.7702
- Mean Squared Error: 5.7702
- Mean Absolute Error: 4.5282
- R2 Score: 0.8632
===================================
```



```
CatBoosting Regressor
Best hyperparameters: {'depth': 3, 'iterations': 500}
Model performance for Training set
- Root Mean Squared Error: 4.4599
- Mean Squared Error: 19.8904
- Mean Absolute Error: 3.5747
- R2 Score: 0.9118
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.5240
- Mean Squared Error: 5.5240
- Mean Absolute Error: 4.3274
- R2 Score: 0.8746
===================================
```



```
AdaBoost Regressor
Best hyperparameters: {'learning_rate': 0.5, 'n_estimators': 256}
Model performance for Training set
- Root Mean Squared Error: 5.7430
- Mean Squared Error: 32.9817
- Mean Absolute Error: 4.7136
- R2 Score: 0.8537
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.9992
- Mean Squared Error: 5.9992
- Mean Absolute Error: 4.6549
- R2 Score: 0.8521
===================================
```

If we choose Linear regression as the final model because that model will get a training set r2 score is 87.42 and a testing set r2 score is 88.03.

7. ADVANTAGE & DISADVANTAGE

Better Management of Student Data

A student management system is also a student database management system. It helps you manage all the student-related data in a well-organized manner.

A Student Management System gives you a unique ID against every student. And using that ID, you can easily track the fee status, assignments, exam results, grades, parent info within seconds.

Improves Overall Teacher Productivity

Similar to the students, the teachers also become more productive with a student management system. It's because, with a system in place, teachers won't have to manually take attendance, manage timetables and assignments.

Everything can be loaded on the management software, which can be accessed by all. This way, teachers can focus more on academics and learning than administrative tasks.

Provide Deeper Insights to Parents

Parents find it quite hard to get all the info about their kids. Parent–Teacher meetings are the only times when they can talk to the teacher and ask for details.

But with a student management system in play, it all becomes easy. Parents can access their kid's data on their phones whenever they like. This way, parents can always stay updated about their kid's activities.

Easy Fee Management

No one likes to call the parents to ask for a fee, right? Well, with a student management system, you won't have to.

The parents can access their ward's fee details on their mobiles. You can even enable notifications for parents regarding fee payments.

This way, the parents won't ever miss fee payments. Also, you won't have to direct teachers to remind the parents of fee payment.

Save Cost

Cost–cutting is quite crucial for schools but is quite hard. One cannot decide on what's required and what's not.

However, you can cut some costs with a student management system. We're talking about the "paper costs."

If you have a software managing all the records, you won't have to purchase/maintain huge piles of paper based records. This way you can save some money.

Although it may not be a huge saving, "a penny saved is a penny earned."

What are the disadvantages of the student management system?

There aren't any disadvantages or limitations of a student management system as such. A better word for disadvantages would be "issues" faced by schools while using a student management system.

Fortunately, there aren’ t many. The most common issue schools/teachers/students/parents face while using a student management system is: a complex user interface. And that’ s a solvable issue.

We’ re humans and we need some time to get a hang of things. If the tool you’ re using is genuinely tough to use, you can go for a tool that’ s simple or get a training.

And that’ s it. There are benefits and benefits only of a student management system. You can remove the discrepancies, if any, by researching and finding an ideal student management system.

8.CONCLUSION

This brings us to an end to the student’s performance prediction. Let us review our work. First, we started by defining our problem statement, looking into the algorithms we were going to use and the regression implementation pipeline. Then we moved on to practically implementing the identification and regression algorithms like Linear Regression, Lasso, K-Neighbors Regressor, Decision Tree, Random Forest Regressor, XGBRegressor, CatBoosting Regressor, and AdaBoost

Regressor. Moving forward, we compared the performances of these models. Lastly, we built a Linear regression model that proved that it works best for student performance prediction problems.

The key takeaways from this student performance prediction are:

Identification of student performance prediction is important for many institutions.

Linear regression gives better accuracy compared to other regression problems.

Linear regression is the best fit for the problem

9.FUTURE SCOPE

This application can be further developed to show more student performance of a country or a certain region where it’ s important for student to have education.

10. APPENDIX

This brings us to an end to the student’s performance prediction. Let us review our work. First, we started by defining our problem statement, looking into the algorithms we were going to use and the regression implementation pipeline. Then we moved on to practically implementing the identification and regression algorithms like Linear Regression, Lasso, K-Neighbors Regressor, Decision Tree, Random Forest Regressor, XGBRegressor, CatBoosting Regressor, and AdaBoost Regressor. Moving forward, we compared the performances of these models. Lastly, we built a Linear regression model that proved that it works best for student performance prediction problems

SOURCE CODE:

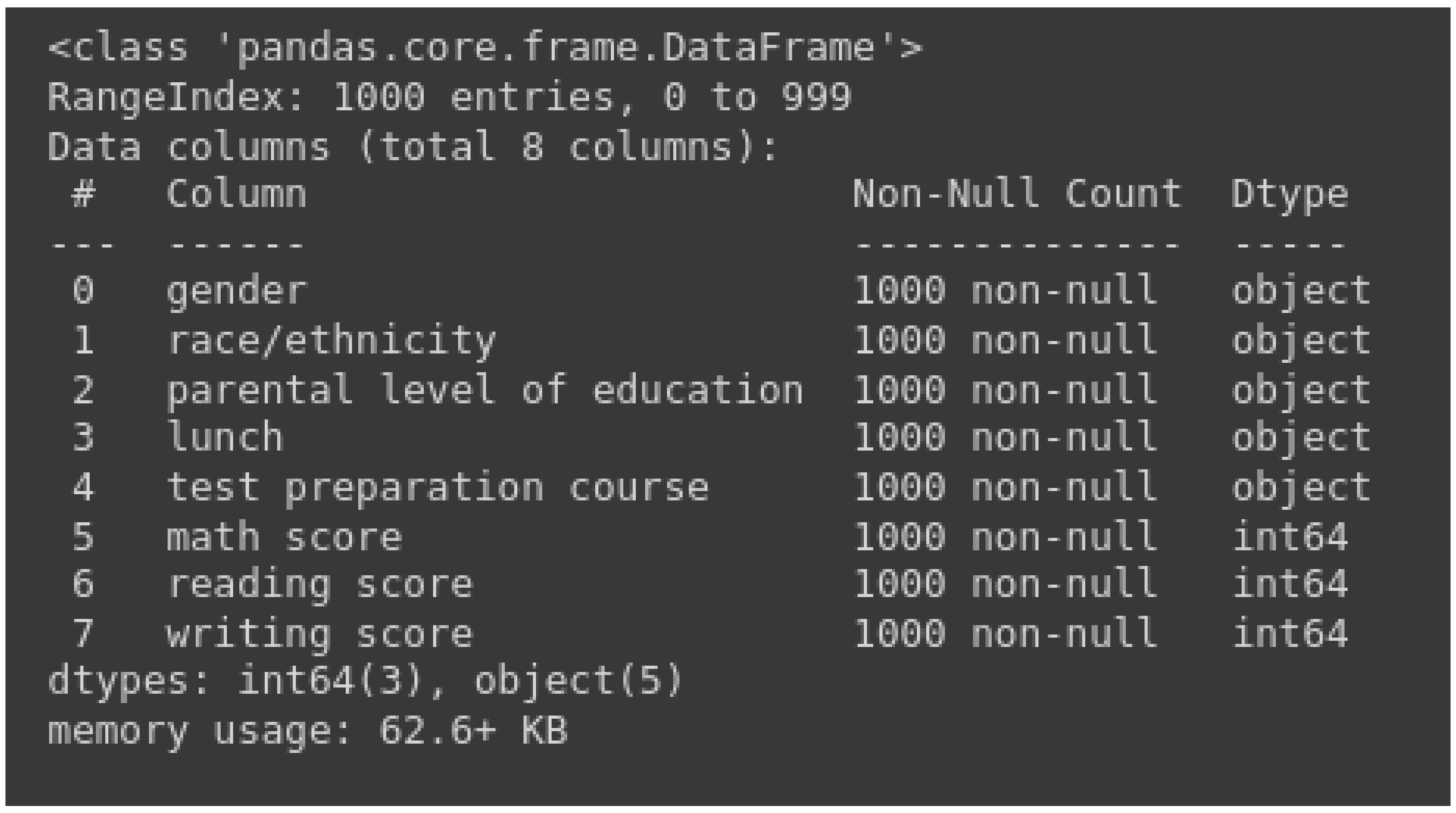
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv("data/StudentsPerformance.csv")
```

.

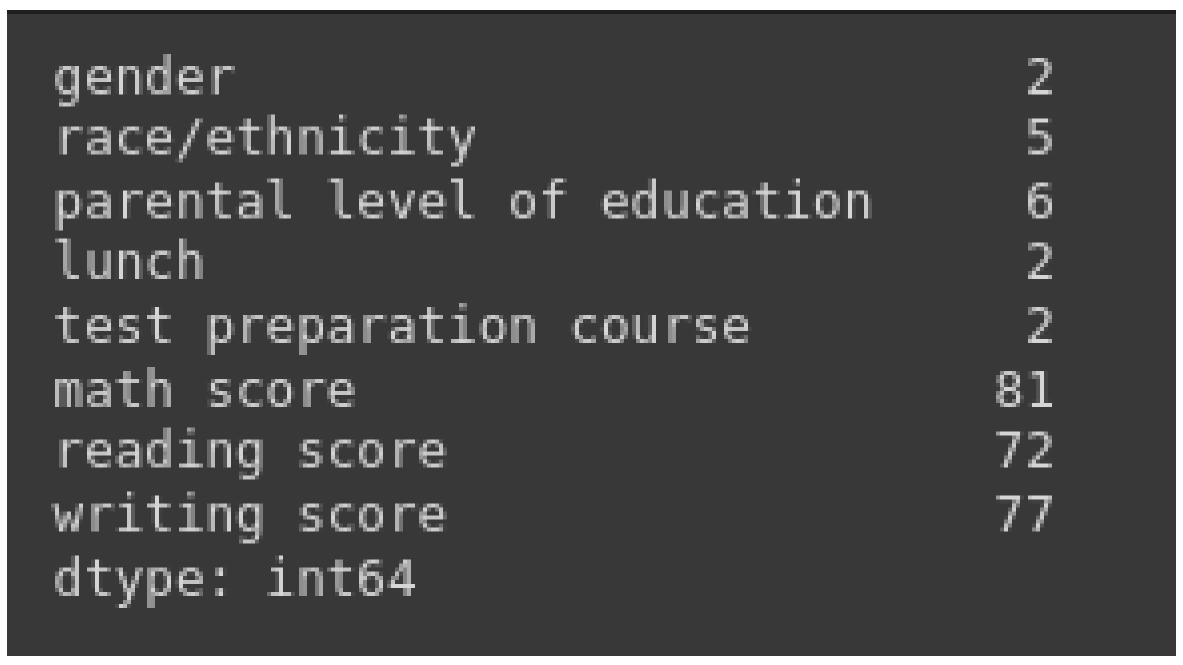
```
df.shape
```

```
df.isnull().sum()
```

```
#check the null and Dtypes
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64 
 6   reading score                1000 non-null   int64 
 7   writing score                1000 non-null   int64 
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```



```
gender                          2
race/ethnicity                  5
parental level of education     6
lunch                           2
test preparation course         2
math score                     81
reading score                  72
writing score                  77
dtype: int64
```

```
print("Categories in 'gender' variable:   ",end=" ")
print(df["gender"].unique())

print("Categories in 'race/ethnicity' variable:   ",end=" ")
print(df["race/ethnicity"].unique())

print("Categories in 'parental level of education' variable:   ",end=" ")
print(df["parental level of education"].unique())
print("Categories in 'lunch' variable:   ",end=" ")
print(df["lunch"].unique())

print("Categories in 'test preparation course' variable:   ",end=" ")
print(df["test preparation course"].unique())



#define numerical and categorical columns
numeric_features = [feature for feature in df.columns if df[feature].dtype != "object"]
categorical_features = [feature for feature in df.columns if df[feature].dtype == "object"]

print("We have {} numerical features: {}".format(len(numeric_features),numeric_features))
print("We have {} categorical features: {}".format(len(categorical_features),categorical_features))



# Create a figure with two subplots
f,ax=plt.subplots(1,2,figsize=(8,6))


# Create a countplot of the 'gender' column and add labels to the bars
sns.countplot(x=df['gender'],data=df,palette ='bright',ax=ax[0],saturation=0.95)
for container in ax[0].containers:
    ax[0].bar_label(container,color='black',size=15)

# Set font size of x-axis and y-axis labels and tick labels
ax[0].set_xlabel('Gender', fontsize=14)
ax[0].set_ylabel('Count', fontsize=14)
ax[0].tick_params(labelsize=14)

# Create a pie chart of the 'gender' column and add labels to the slices
```

```
plt.pie(x=df['gender'].value_counts(),labels=['Male','Female'],explode=[0,0.1],autopct='%1.1f%%',shadow
=True,colors=['#ff4d4d','#ff8000'], textprops={'fontsize': 14})

# Display the plot
plt.show()



# Define a color palette for the countplot
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
# blue, orange, green, red, purple are respectiively the color names for the color codes used above

# Create a figure with two subplots
f, ax = plt.subplots(1, 2, figsize=(12, 6))

# Create a countplot of the 'race/ethnicity' column and add labels to the bars
sns.countplot(x=df['race/ethnicity'], data=df, palette=colors, ax=ax[0], saturation=0.95)
for container in ax[0].containers:
    ax[0].bar_label(container, color='black', size=14)

# Set font size of x-axis and y-axis labels and tick labels
ax[0].set_xlabel('Race/Ethnicity', fontsize=14)
ax[0].set_ylabel('Count', fontsize=14)
ax[0].tick_params(labelsize=14)

# Create a dictionary that maps category names to colors in the color palette
color_dict = dict(zip(df['race/ethnicity'].unique(), colors))

# Map the colors to the pie chart slices
pie_colors = [color_dict[race] for race in df['race/ethnicity'].value_counts().index]

# Create a pie chart of the 'race/ethnicity' column and add labels to the slices
plt.pie(x=df['race/ethnicity'].value_counts(), labels=df['race/ethnicity'].value_counts().index,
explode=[0.1, 0, 0, 0, 0], autopct='%1.1f%%', shadow=True, colors=pie_colors, textprops={'fontsize': 14})
```

```
# Set the aspect ratio of the pie chart to 'equal' to make it a circle
plt.axis('equal')

# Display the plot
plt.show()
```

id = Insights>Insights

```
plt.rcParams['figure.figsize'] = (15, 9)
plt.style.use('fivethirtyeight')
sns.histplot(df["parental level of education"], palette = 'Blues')
plt.title('Comparison of Parental Education', fontweight = 30, fontsize = 20)
plt.xlabel('Degree')
plt.ylabel('count')
plt.show()
```

id = Insights>Insights

```
df.groupby('parental level of education').agg('mean').plot(kind='barh',figsize=(10,10))
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

id = Insights>Insights

```
plt.Fig
figsize=(18,8))
plt.subplot(1, 4, 1)
plt.title('MATH SCORES')
sns.violinplot(y='math score',data=df,color='red',linewidth=3) plt.subplot(1, 4, 2) plt.title('READING
SCORES') sns.vi

plot(y='reading score',data=df,color='green',linewidth=3) plt.subplot(1, 4, 3) plt.title('WRITING
SCORES') sns.violinplot(y='writing score',data=df,color='blue',linewidth=3) plt.show()
```

id = Insights>Insights

```
# Set figure size
plt.rcParams['figure.figsize'] = (12, 9)

# First row of pie charts
plt.subplot(2, 3, 1)
size = df['gender'].value_counts()
labels = 'Female', 'Male'
color = ['red','green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Gender', fontsize=20)
plt.axis('off')

plt.subplot(2, 3, 2)
size = df['race/ethnicity'].value_counts()
labels = 'Group C', 'Group D', 'Group B', 'Group E', 'Group A'
color = ['red', 'green', 'blue', 'cyan', 'orange']
```

```
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Race/Ethnicity', fontsize=20)
plt.axis('off')

plt.subplot(2, 3, 3)
size = df['lunch'].value_counts()
labels = 'Standard', 'Free'
color = ['red', 'green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Lunch', fontsize=20)
plt.axis('off')

# Second row of pie charts
plt.subplot(2, 3, 4)
size = df['test preparation course'].value_counts()
labels = 'None', 'Completed'
color = ['red', 'green']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Test Course', fontsize=20)
plt.axis('off')

plt.subplot(2, 3, 5)
size = df['parental level of education'].value_counts()
labels = 'Some College', "Associate's Degree", 'High School', 'Some High School', "Bachelor's Degree",
"Master's Degree"
color = ['red', 'green', 'blue', 'cyan', 'orange', 'grey']
plt.pie(size, colors=color, labels=labels, autopct='%.2f%%')
plt.title('Parental Education', fontsize=20)
plt.axi
ff') # Remove extra subplot plt.subplot(2, 3, 6).remove() # Add super title plt.suptitle('Comparison of
Student Attributes', fontsize=20, fontweight='bold') # Adjust layout and show plot # This is removed as
there are only 5 subplots in this figure and we want to arrange them in a 2x3 grid. # Since there is no
6th subplot, it is removed to avoid an empty subplot being shown in the figure. plt.tight_layout()
plt.subplots_adjust(top=0.85) plt.show()
```

```
# Modelling
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection imp
RandomizedSearchCV from catboost import CatBoostRegressor from xgboost import XGBRegressor import
warnings
X = df.drop(columns="math score",axis=1)
y = df["math score"]
```

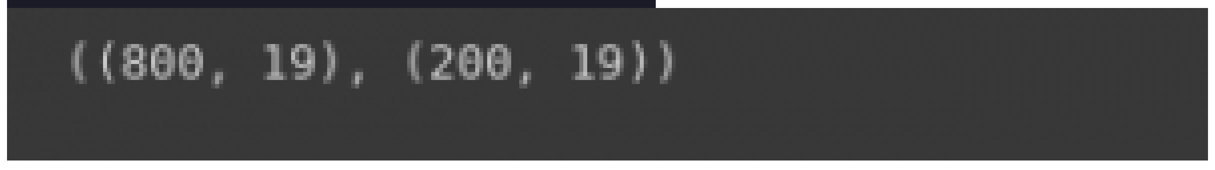
Create Column Transformer with 3 Types of Transformers

```
num_features = X.select_dtypes(exclude="object").columns
cat_features = X.select_dtypes(include="object").columns

from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder()

preprocessor = Column
transformer( [ ("OneHotEncoder", oh_transformer, cat_features), ("StandardScaler",
numeric_transformer, num_features), ] ) X = preprocessor.fit_transform(X)
```

Separate Dataset into Train and Test

To separate the dataset into train and test to identify the training size and testing size of the dataset.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
X_train.shape, X_test.shape
```



```
((800, 19), (200, 19))
```

Create an Evaluate Function for Model Training

This function is use to evaluate the model and build a good model.

```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, mse, rmse, r2_square
```

To create a models variable and form a dictionary formate.

```
models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest Regressor": RandomForestRegressor(),
    "Gradient Boosting": GradientBoostingRegressor(),
    "XGBRegressor": XGBRegressor(),
    "CatBoosting Regressor": CatBoostRegressor(verbose=False),
    "AdaBoost Regressor": AdaBoostRegressor()
}
model_list = []
r2_list =[]

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate Train and Test dataset
    model_train_mae, model_train_mse, model_train_rmse, model_train_r2 = evaluate_model(y_train,
 y_train_pred)

    model_test_mae, model_test_mse, model_test_rmse, model_test_r2 = evaluate_model(y_test,
y_test_pred)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Squared Error: {:.4f}".format(model_train_mse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')
    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".
    For

model_test_r2)) r2_list.append(model_test_r2) print('='*35) print('\n')
```

GITHUB LINK:

https://github.com/naanmudhalvan-SI/PBL-NT-GP--7266-1681056033

PROJECT VIDEO DEMO LINK:

https://drive.google.com/file/d/1VnxlnPWL
3TD9tZe

lbcEjWaFPBR3c6TD/view?usp=drivesdk