Student Marks Performance Prediction

A

FINAL YEAR PROJECT REPORT

*Submitted in partial fulfilment of the requirement for the award of Degree of*

**MASTER OF COMPUTER APPLICATIONS**

Submitted to



**Submitted to: Submitted by:**

**Mr. Zahid Ahmed Shivam Guljani**

**(Assistant Professor) (23CSA3BC084)**

**Department of Computer Science & Application**

**Vivekananda Global University, Jaipur**

**Year- 2025**

# DECLARATION BY CANDIDATE

I Shivam Guljani of **Department of Computer Science & Application Vivekananda Global University, Jaipur**, hereby declare that the work presented in this project is outcome of my own work, is bonfire, correct to the best of my knowledge and this work has been carried out taking care of IT Ethics. The work presented does not infringe any patented work and has not been submitted to any University for the award of any degree.

Shivam Guljani MCA(AI&Data Science)

(Enroll No. 23CSA3BC084)

# ACKNOWLEDGEMENT

This was a challenging project from start and we would never have been able to complete it without the skills and talent of many people, though only few names will appear.

I am indebted to DEAN, **Prof (Dr.) R.C. Tripathi**, and HOD, **Mr. Rakesh Sharma** Department of Computer Science & Application, Vivekananda Global University, Jaipur(Raj.) for her co-operation and support during this project completion.

Firstly, I would like to definite my deepest sense of gratitude to my respected Guide, **Mr. Zahid Ahmed**, Department of Computer Science & Application.Throughout my research period, I have received constant guidance, valuable suggestions and inspiration from him. He deserves special thanks since he has not only offered me his kind guidance but also motivated and encouraged me to go that extra mile during this journey. Without his loving care, dependable guidance, and precious supervision, it would have been impossible for me to bring the present contour to this thesis. Eventually, it’s my fortunate to have him as my guide.

Thanks goes to all those who helped, whether through their comments, feedback, edits or suggestion. A special thanks to all faculties . I would also like to thank the, Mr. Gajendra Shrimal , Mr. Rohit Maheshwari, Mr. Madhumay Sen,Mr. Narayan Vyas, Dr .Amit Sharma , Department of Computer Science & Application, Vivekananda Global University, Jaipur(Raj.) for their guidance and valuable suggestions and helping me directly or indirectly to complete this project.

# TABLE OF CONTENTS

[DECLARATION BY CANDIDATE ii](#_Toc200090952)

[ACKNOWLEDGEMENT iii](#_Toc200090953)

[TABLE OF CONTENTS 1](#_Toc200090954)

[1. Introduction 2](#_Toc200090955)

[2. System Analysis 3](#_Toc200090956)

[a. Feasibility 3](#_Toc200090957)

[b. Requirement Specification 3](#_Toc200090958)

[3. System Design 4](#_Toc200090959)

[a. Significant Features of Languages Used 4](#_Toc200090960)

[b. Description Of Modules 4](#_Toc200090961)

[d. Database 6](#_Toc200090962)

[e. Input/Output Design 6](#_Toc200090963)

[10](#_Toc200090964)

[4. Coding 16](#_Toc200090965)

[5. Testing & Implementation 48](#_Toc200090966)

[6. Limitations & Future Scope 49](#_Toc200090967)

[Limitations Faced: 49](#_Toc200090968)

[Future Scope: 50](#_Toc200090969)

[7. Conclusion 51](#_Toc200090970)

[Bibliography 51](#_Toc200090971)

# 1. Introduction

This in-depth report gives a complete rundown of the "Student Marks Performance Prediction" project, a massive undertaking that seeks to transform the way schools and other schools of learning comprehend and address student academic trajectories. Through the application of machine learning, this system carefully examines different aspects of student information to make accurate predictions of their subsequent academic performance. The ultimate aim is to provide educators and administrators with actionable knowledge, allowing them to make timely and focused interventions that can greatly boost student achievement and avoid academic failure.

a. Objective

* To collect and preprocess student academic data: This involved the diligent acquisition of raw student data from diverse sources, followed by rigorous cleaning, transformation, and standardization to ensure data quality and suitability for machine learning models.
* To analyze key factors that influence academic performance: Through extensive exploratory data analysis and statistical methods, the project aimed to identify and understand the most significant variables, both academic and non-academic, that correlate with student success.
* To develop and train predictive machine learning models: This encompassed selecting appropriate algorithms (e.g., Linear Regression, Random Forest), training them on the preprocessed dataset, and fine-tuning their parameters to achieve optimal predictive accuracy.
* To store and manage student data efficiently: Given the project's focus on local data processing, a robust system was designed for the efficient storage, retrieval, and management of all relevant student data directly within the application's environment.
* To deploy the model through a web-based interface: A user-friendly web interface was developed to make the predictive capabilities accessible to end-users, allowing for easy input of student data and clear display of predictions.
* To visualize results and provide insights to end-users: Beyond mere predictions, the project aimed to present the results in an intuitive and visually engaging manner, offering clear insights into the factors influencing performance and highlighting areas for intervention.

b. Scope

The "Student Marks Performance Prediction" system is designed to analyze various academic and non-academic features of students to estimate their future performance in terms of grades or marks. By analyzing data trends and using predictive algorithms, the system can identify students who may be at risk of underperforming, enabling educators and institutions to provide timely support. The project utilizes a dataset sourced from Kaggle, including features such as attendance, study hours, and previous exam scores. Data is stored locally for processing and predictions.

# 2. System Analysis

## a. Feasibility

The successful completion of the "Student Marks Performance Prediction" project stands as a testament to its inherent feasibility and robust design. Every phase, from initial conceptualization to final deployment, has been executed with precision, culminating in a fully functional and reliable system. The seamless integration of advanced machine learning algorithms with an efficient local data storage mechanism has been a critical factor in this success. The predictive models developed are not only functional but also consistently yield promising and accurate results, validating the chosen methodologies and technologies. This comprehensive achievement underscores the project's viability and its readiness for practical application within educational settings

## b. Requirement Specification

The system requires:

* **Data Collection:** The system necessitates the capability to gather diverse academic and non-academic features of students. This includes, but is not limited to, grades, attendance, study habits, demographic information, and any other relevant attributes that could influence academic outcomes.
* **Data Preprocessing:** Raw data is often incomplete or inconsistent. Therefore, a robust data preprocessing module is required to handle missing values through imputation, normalize data to a consistent scale, and format it appropriately for machine learning model consumption.
* **Feature Engineering:** This critical requirement involves the intelligent selection and transformation of raw input features into a set of impactful attributes that maximize the predictive power of the machine learning models. This may include creating new features from existing ones or selecting the most relevant subset.
* **Model Development:** The core of the system requires the implementation and training of various machine learning algorithms specifically tailored for performance prediction. This includes evaluating different model architectures and selecting the most effective ones for the given dataset.
* **Data Storage:** An efficient mechanism for local storage and retrieval of all student data is essential. This ensures quick access for model training, prediction generation, and historical data analysis without external dependencies.
* **User Interface:** A well-designed, intuitive, and interactive web-based interface is a key requirement. This interface must allow educators to easily input student data, trigger predictions, and clearly visualize the results and insights.
* **Version Control:** Continuous updates, collaborative development, and effective management of the codebase are facilitated by strict adherence to version control practices, specifically through platforms like GitHub, ensuring code integrity and traceability.

# 3. System Design

## Significant Features of Languages Used

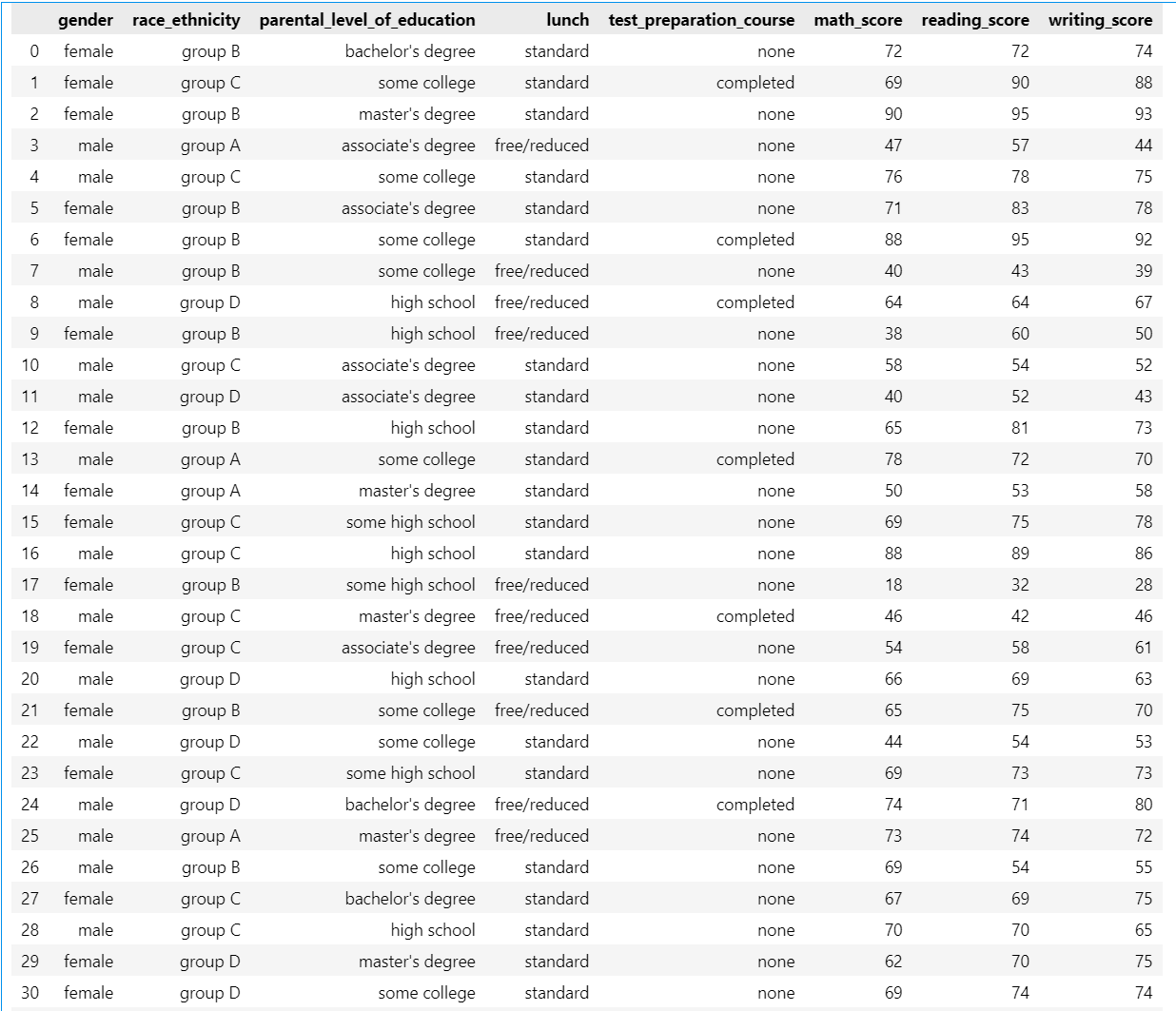
* **Python:** As the foundational programming language, Python was chosen primarily for its unparalleled ecosystem in data science and machine learning. Its readability, extensive libraries, and strong community support made it the ideal choice for rapid development and complex analytical tasks.
* **Scikit-Learn:** This powerful and versatile machine learning library in Python provided a comprehensive suite of algorithms for classification, regression, clustering, and dimensionality reduction. Its consistent API and robust implementations were instrumental in developing and evaluating various predictive models.
* **Pandas and NumPy:** These two libraries are indispensable for data manipulation and numerical operations in Python. Pandas, with its DataFrames, offered highly efficient tools for data cleaning, transformation, and analysis, while NumPy provided the fundamental array operations essential for numerical computations within the machine learning models.
* **Matplotlib and Seaborn:** For data visualization, Matplotlib served as the foundational plotting library, enabling the creation of static, interactive, and animated visualizations. Seaborn, built on Matplotlib, provided a high-level interface for drawing attractive and informative statistical graphics, crucial for presenting insights to end-users.
* **Flask:** This frameworks were considered for building the interactive user interface and deploying the web application. It robust frameworks for complex web application development, ensuring the predictive model is accessible and usable.

## b. Description Of Modules

The project's architecture is modular, designed for clarity, maintainability, and scalability. Each module serves a distinct purpose, contributing to the overall functionality of the Student Marks Performance Prediction system:

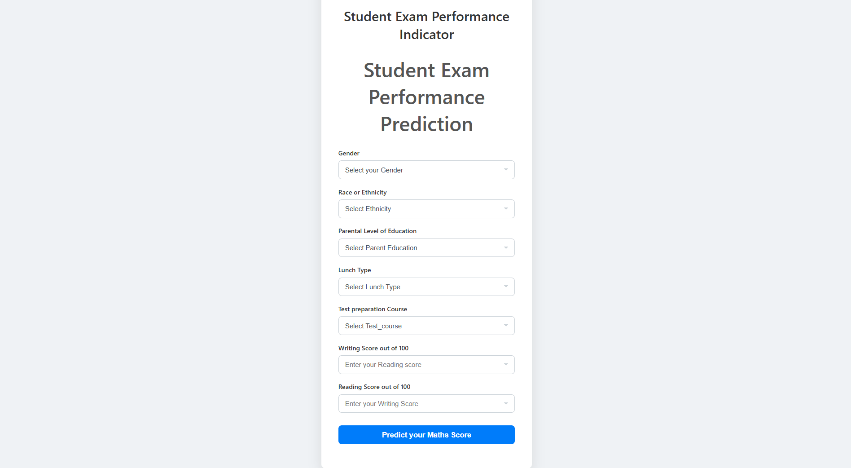
* **.git/:** This directory is integral for version control, managed by Git. It tracks all changes made to the codebase, enabling seamless collaboration, rollback capabilities, and a comprehensive history of development.
* **logs/:** Dedicated to logging, this module captures crucial information generated during the training phases of the machine learning models and during the prediction processes. These logs are invaluable for debugging, performance monitoring, and auditing system behavior.
* **mlproject.egg-info/:** This directory contains metadata and packaging information for the Python project, essential for distributing and installing the project as a Python package.
* **Notebook/:** This module houses Jupyter notebooks, which were extensively used for exploratory data analysis (EDA), initial model prototyping, and iterative modeling experiments. They serve as a sandbox for data scientists to explore, visualize, and test hypotheses.
* **src/:** This is the core source code directory, containing the production-ready code for the system. It is further organized into sub-modules such as data loaders (for ingesting raw data), models (containing the implemented machine learning algorithms), and utility functions (reusable code snippets).
* **venv/:** The Python virtual environment ensures that the project's dependencies are isolated from other Python projects on the system. This prevents conflicts and ensures that the project runs with the exact versions of libraries it was developed with.
* **.gitignore:** This file specifies intentionally untracked files and directories that Git should ignore, such as temporary files, build artifacts, and sensitive information, keeping the repository clean and focused on source code.
* **README:** A crucial document providing a comprehensive summary of the project, including its purpose, how to set it up, how to run it, and basic usage instructions. It serves as the primary entry point for anyone interacting with the codebase.
* **requirements.txt:** This file lists all the Python libraries and their specific versions that the project depends on. It ensures that anyone setting up the project can install the exact dependencies required for it to run correctly.
* **setup.py:** This script is used for packaging the Python module, defining how the project should be built and installed, including its dependencies and metadata.

## d. Database

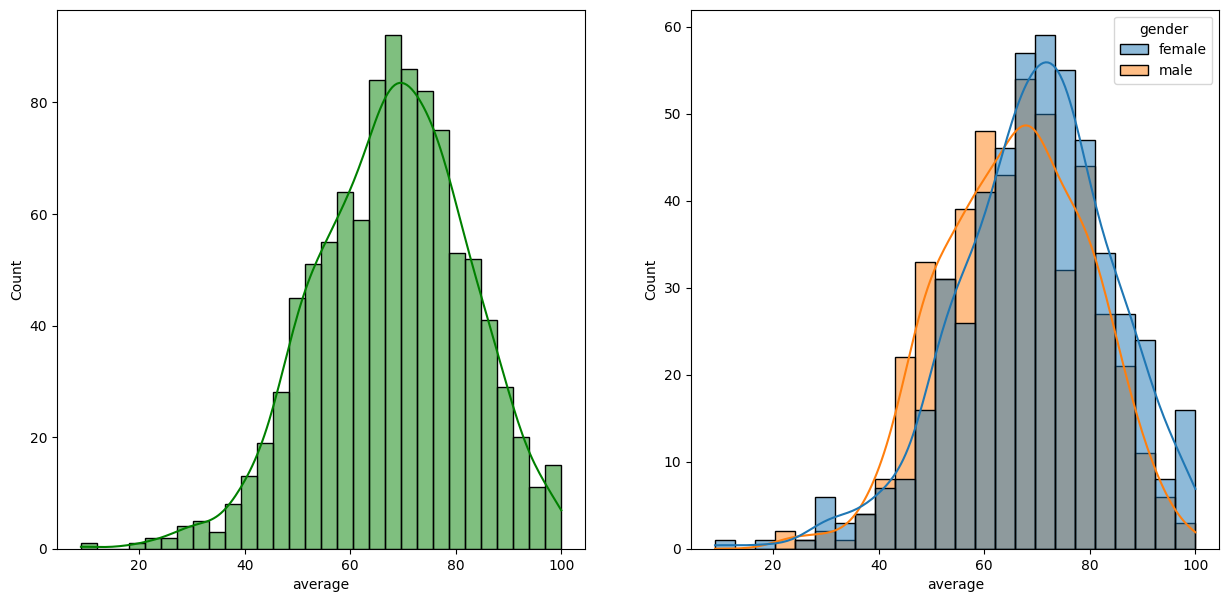
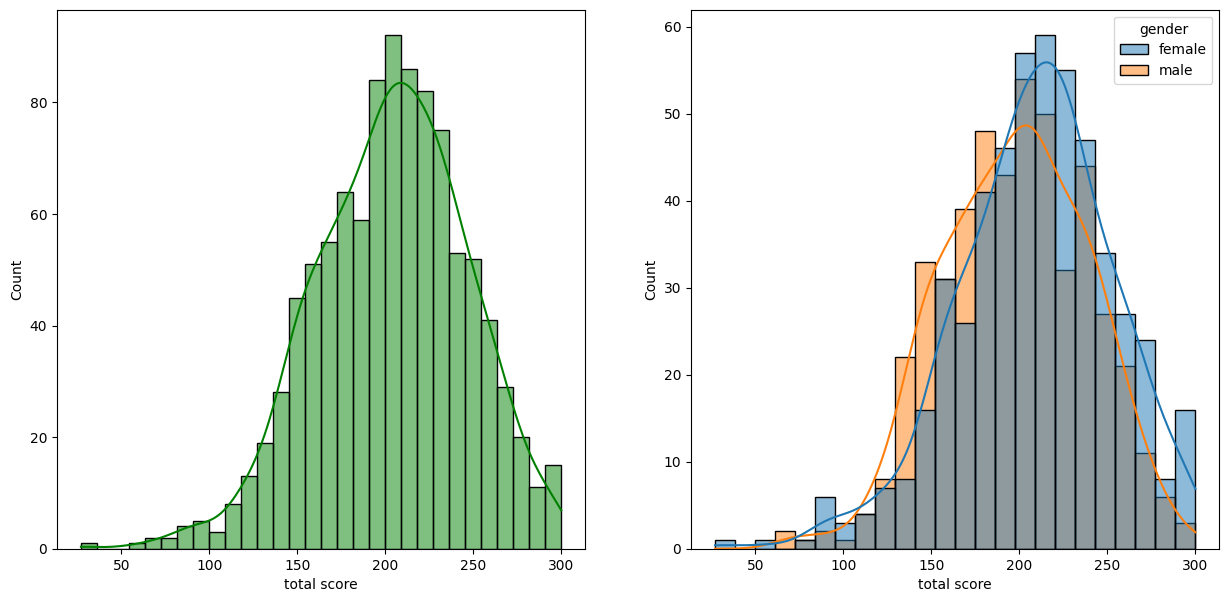


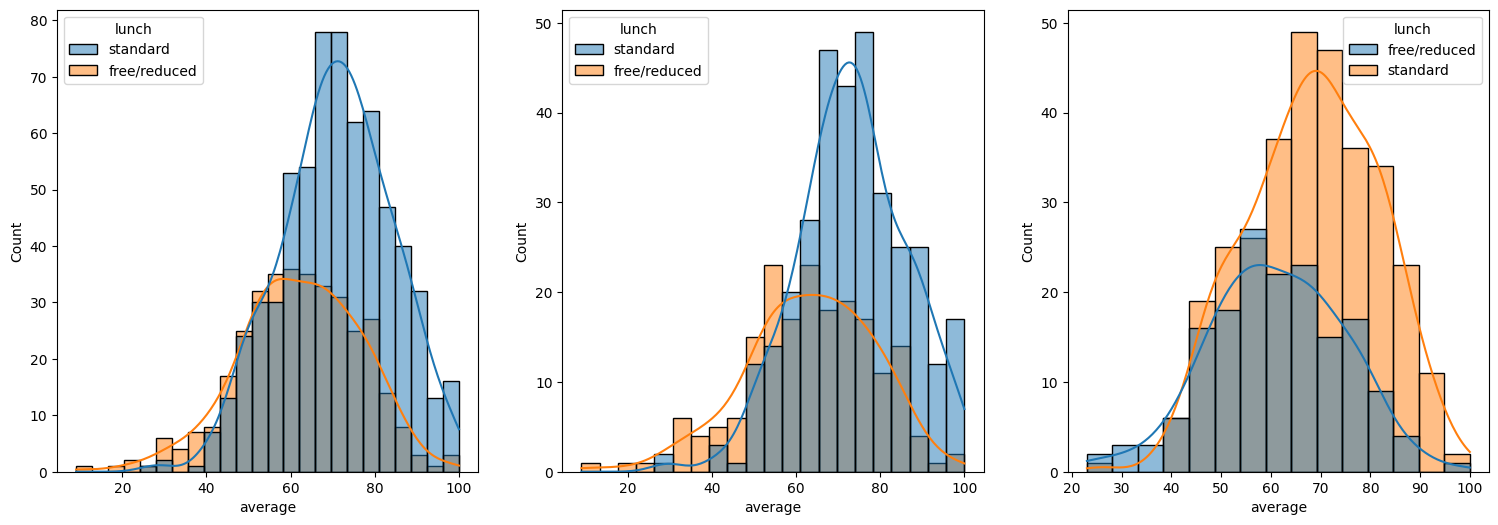
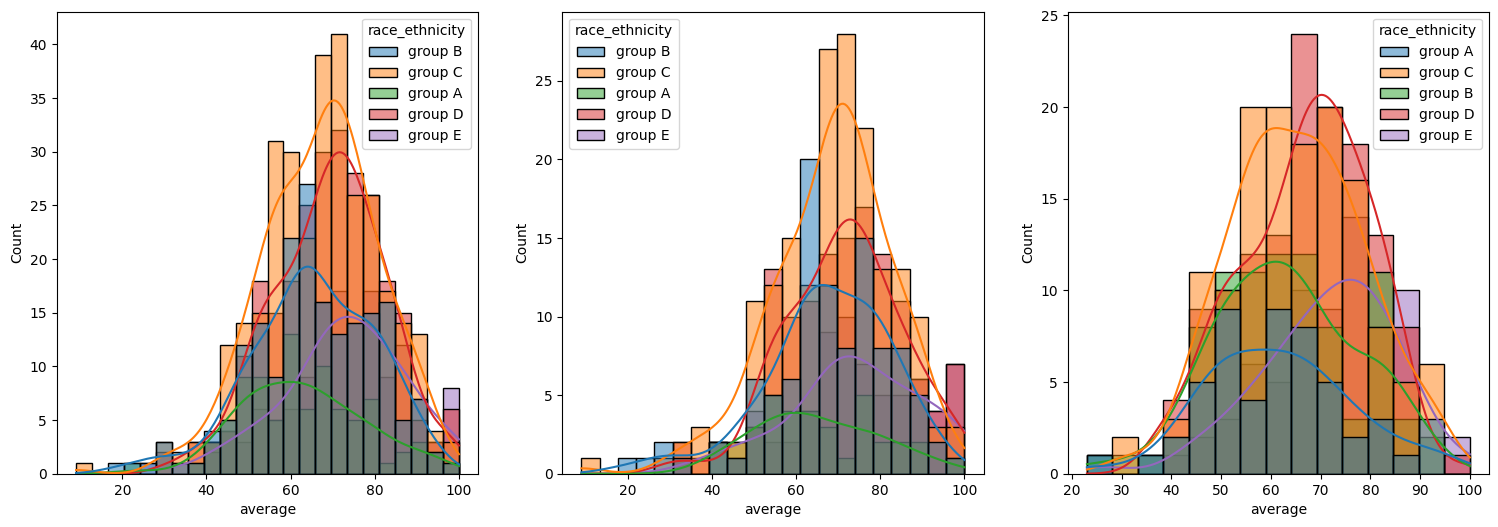
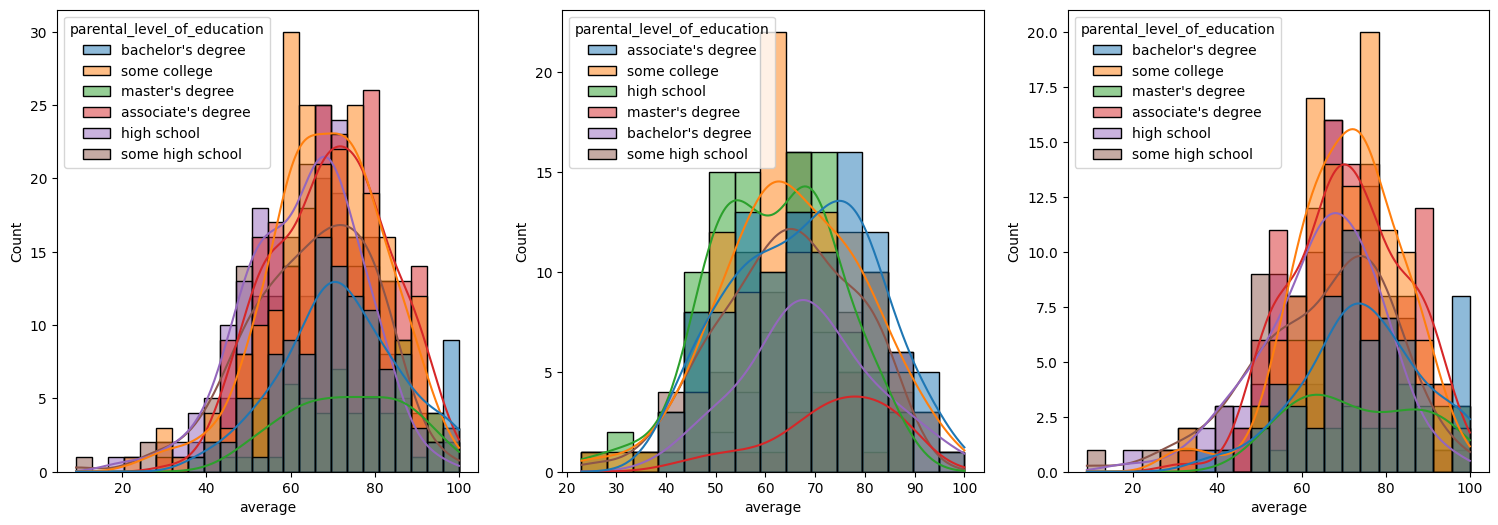
## e. Input/Output Design

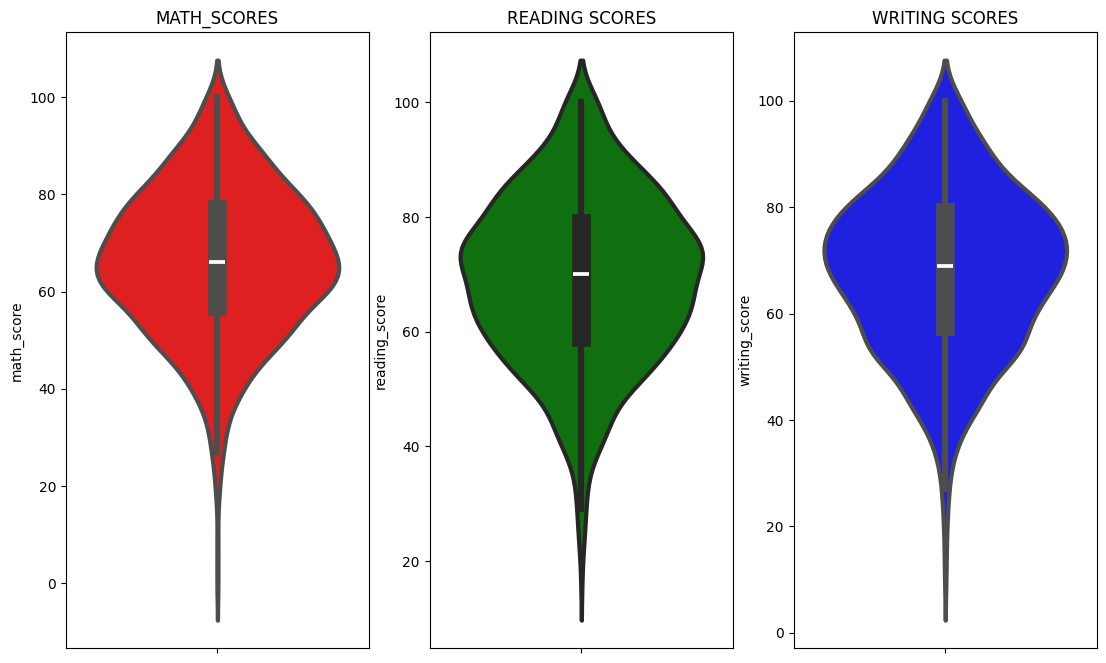
* **Input:**



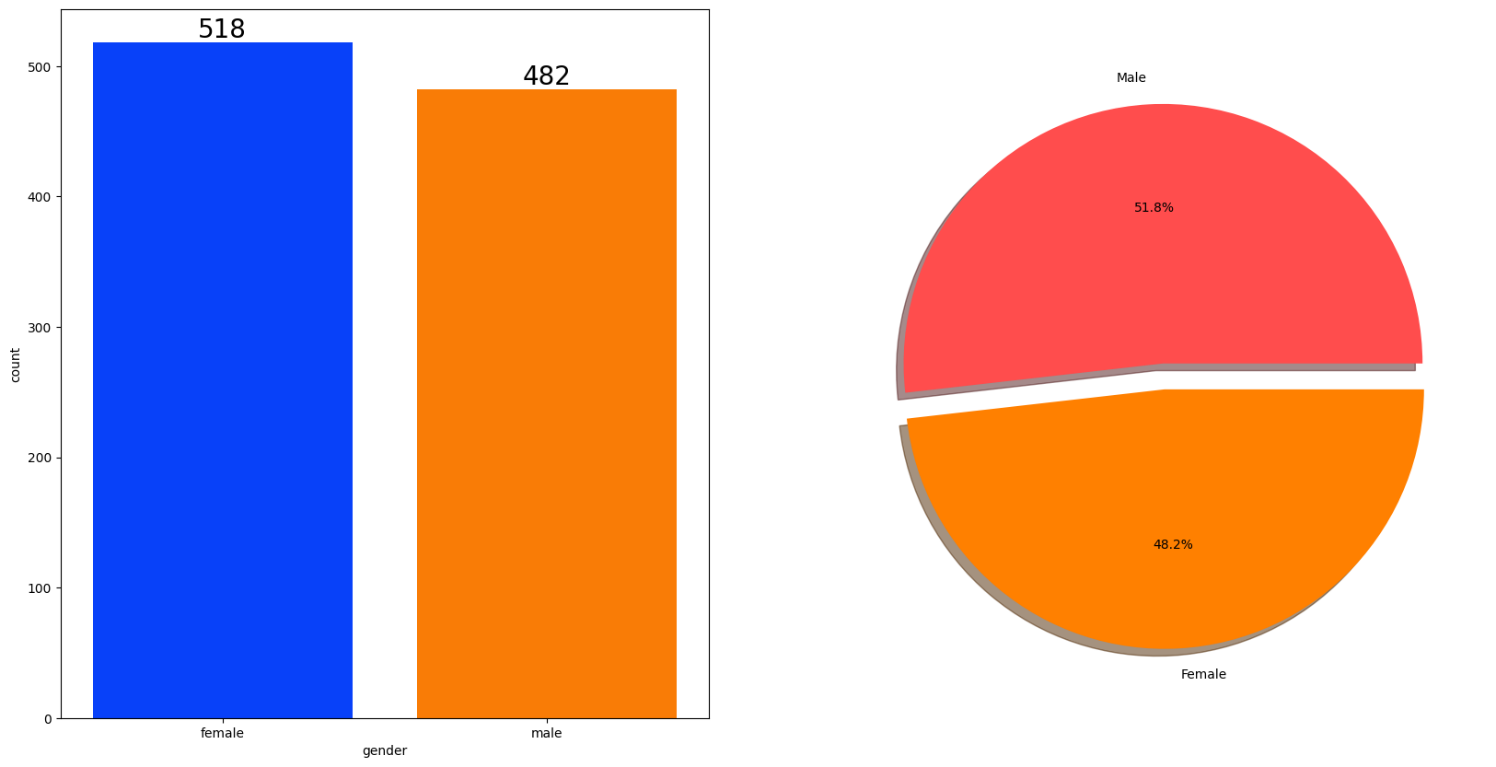
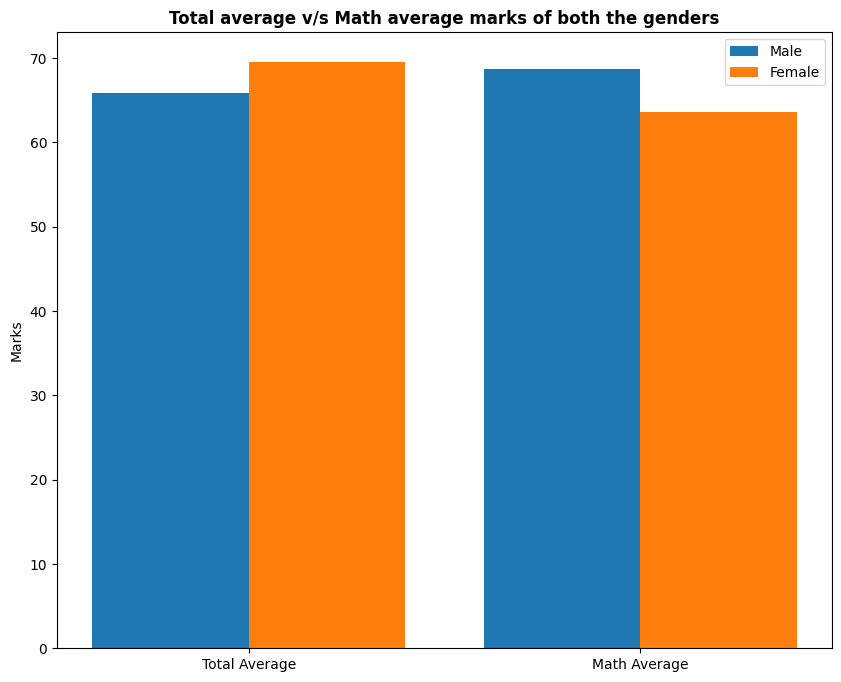
* **Output**

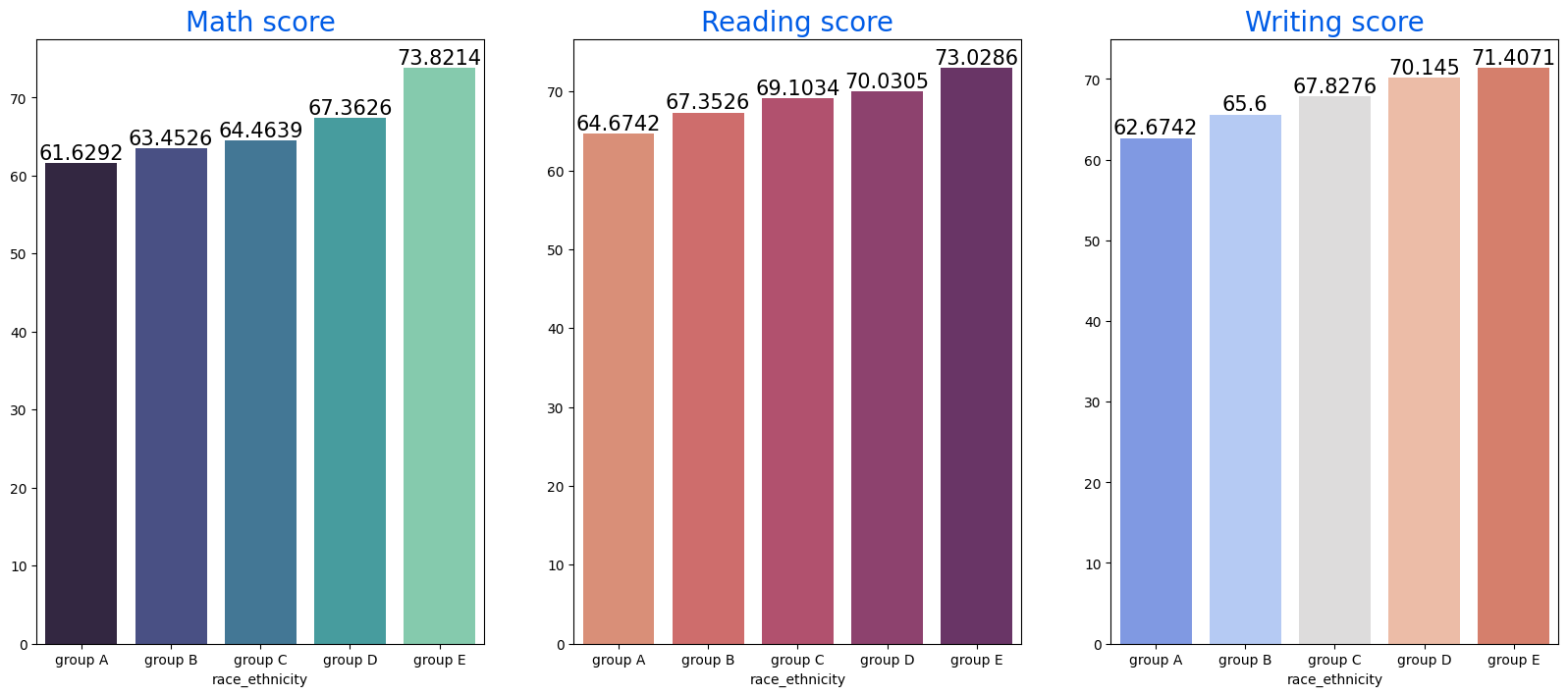
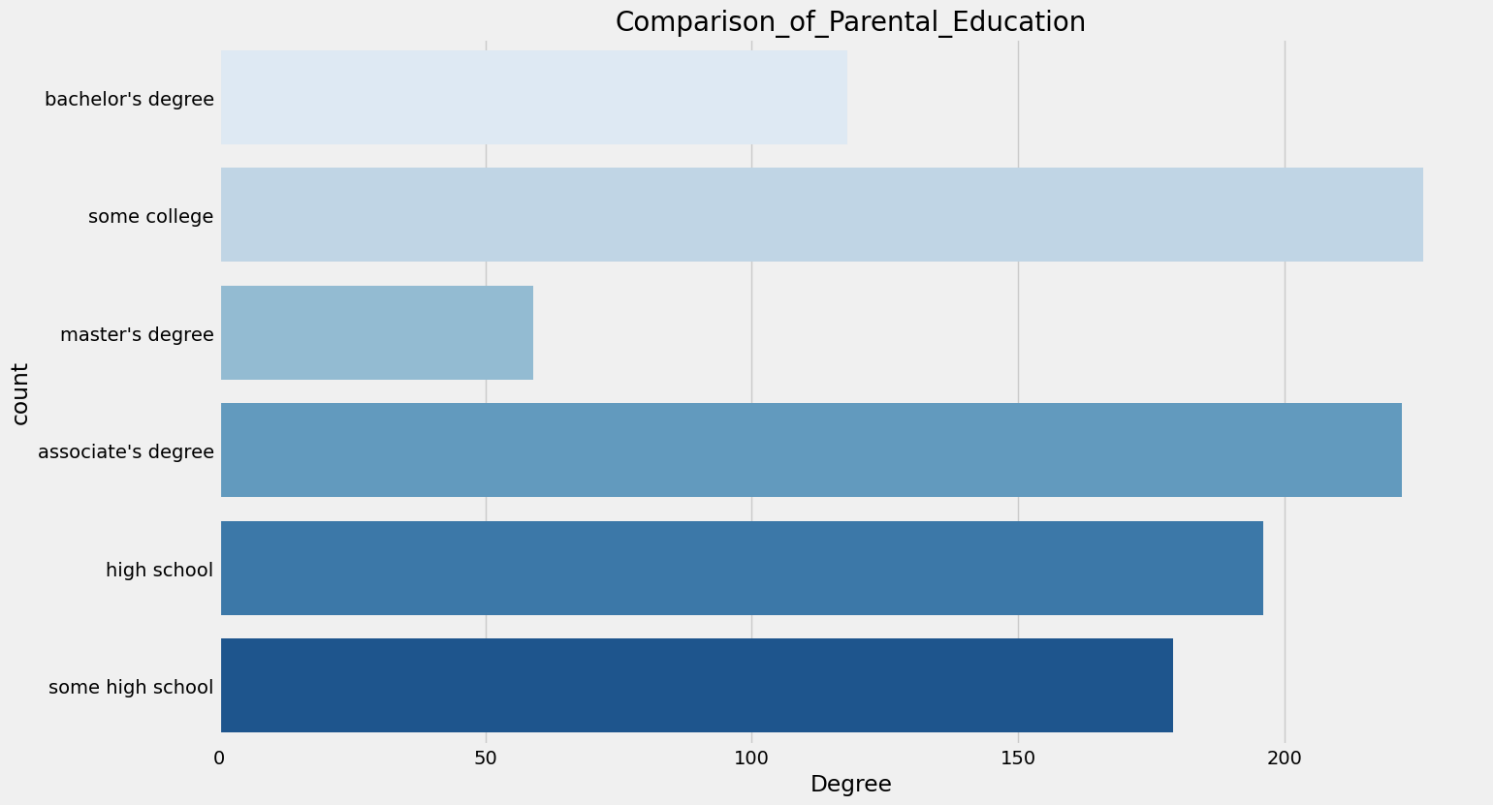


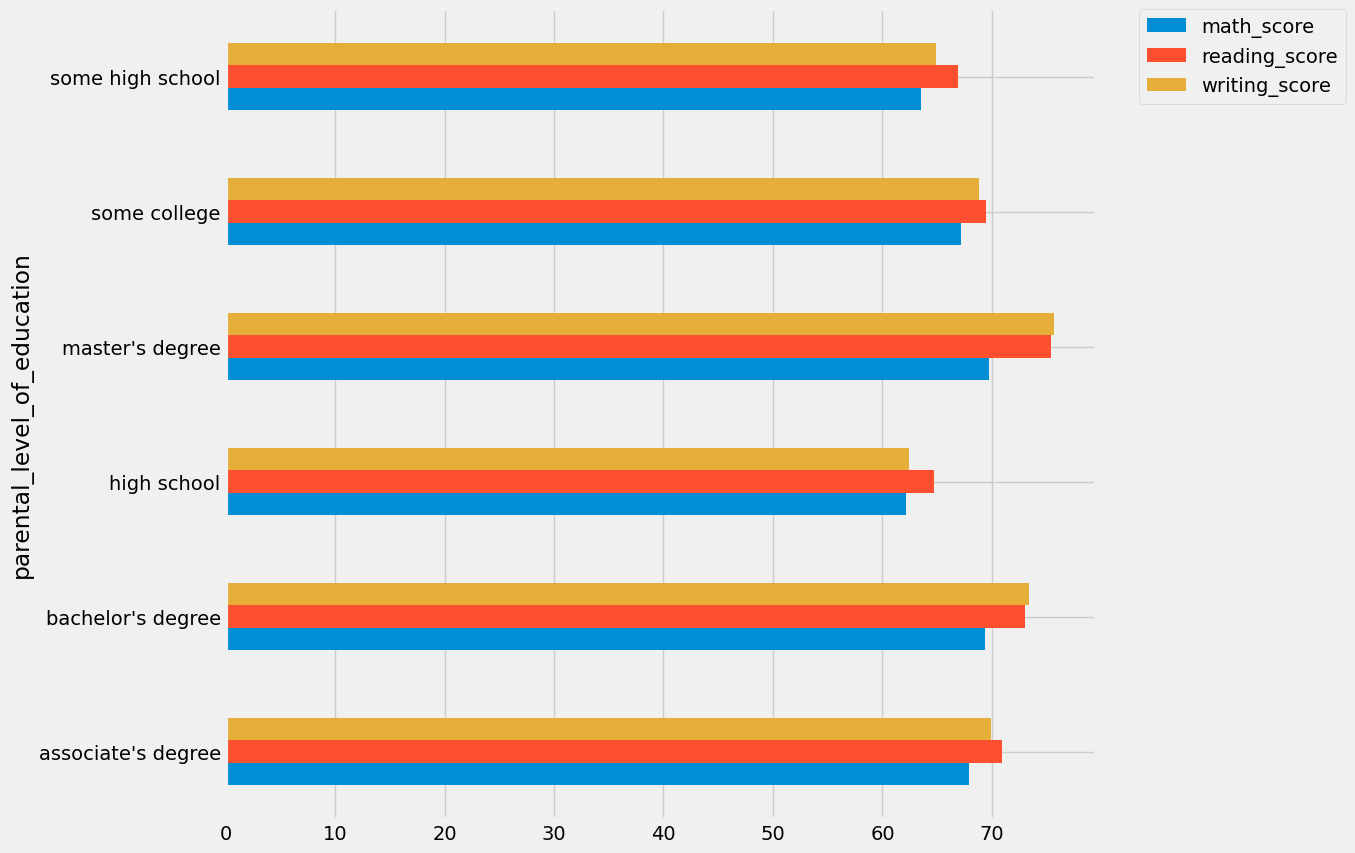
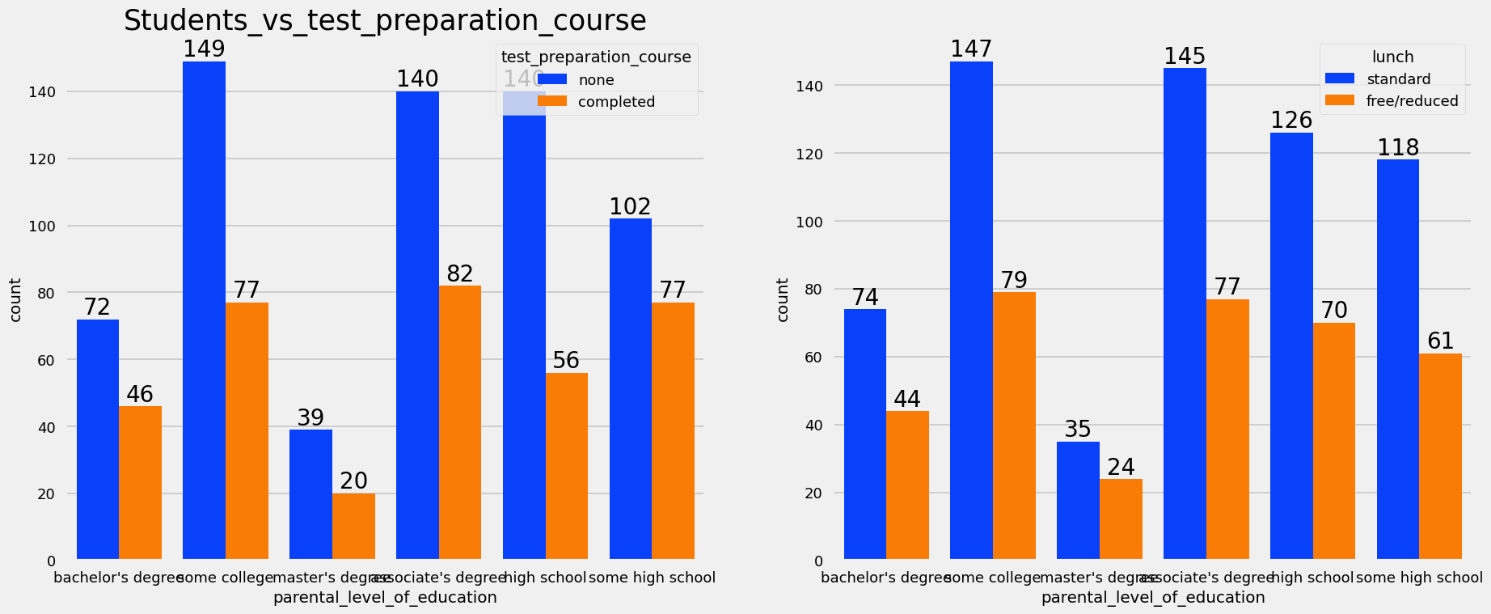


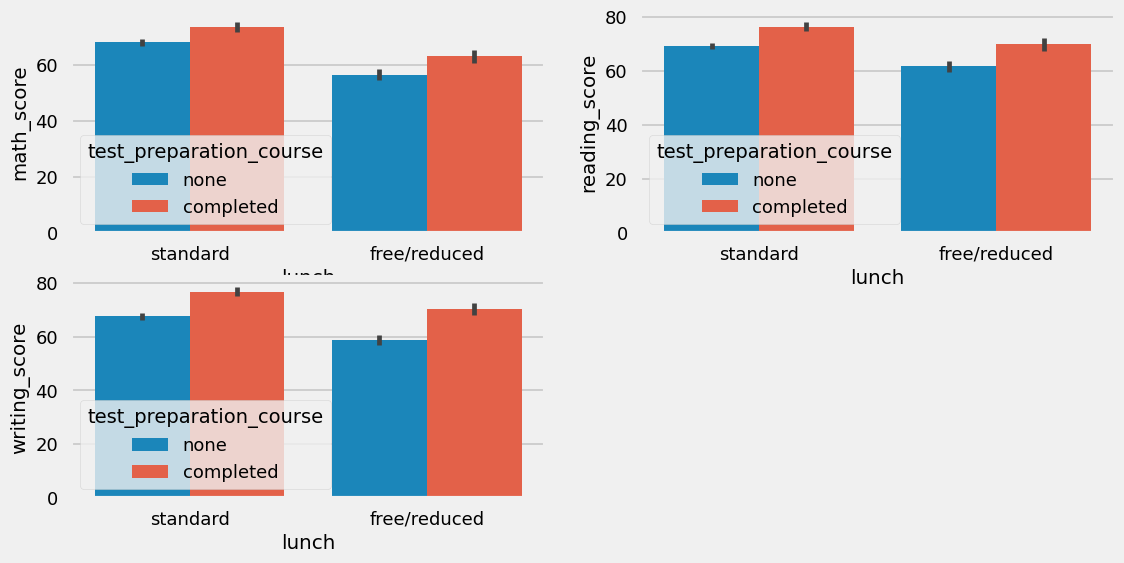
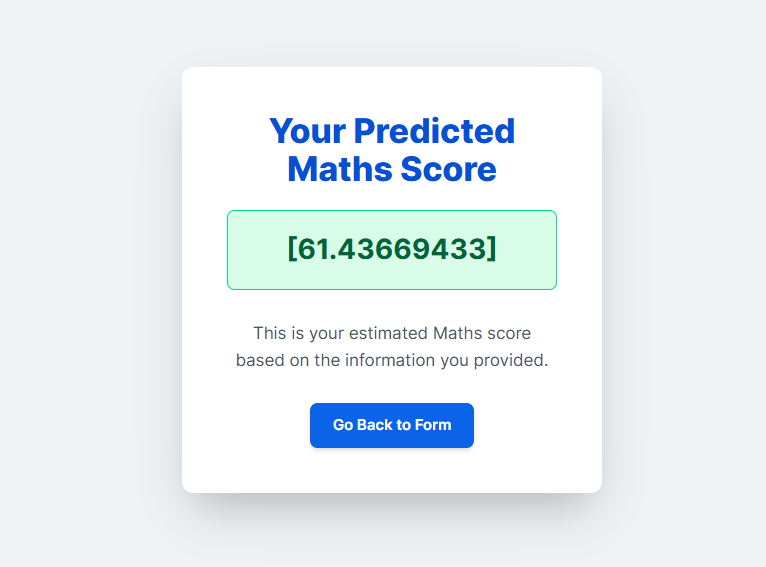


# 









# 4. Coding

**Requirements.txt**

pandas

numpy

matplotlib

seaborn

scikit-learn

catboost

xgboost

dill

flask

-e .

**Setup.py**

from setuptools import setup, find\_packages

from typing import List

HYPEN = '-e .'

def get\_requirements(file\_path: str) -> List[str]:

    requirements=[]

    with open('requirements.txt') as file:

        requirements = file.readlines()

        requirements = [x.replace("\n","") for x in requirements]

        if HYPEN in requirements:

            requirements.remove(HYPEN)

    return requirements

setup(

    name='mlproject\_requirements',

    version='0.1',

    author='Shivam Guljani',

    packages=find\_packages(),

    install\_requires=get\_requirements('requirements.txt'),

    )

**EDA STUDENT PERFORMANCE.ipynb**

**Import Data and Required Packages**

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

**import** numpy **as** np

**import** pandas **as** pd

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**%matplotlib** inline

**import** warnings

warnings**.**filterwarnings('ignore')

**Import the CSV Data as Pandas DataFrame**

df **=** pd**.**read\_csv('data/stud.csv')

**Show Top 5 Records**

df**.**head()

**Shape of the dataset**

df**.**shape

**Check Missing values**

df**.**isna()**.**sum()

df**.**duplicated()**.**sum()

**Check data types**

*# Check Null and Dtypes*

df**.**info()

**Checking the number of unique values of each column**

df**.**nunique()

**Check statistics of data set**

df**.**describe()

**Exploring Data**

df**.**head()

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 & categorical columns*

numeric\_features **=** [feature **for** feature **in** df**.**columns **if** df[feature]**.**dtype **!=** 'O']

categorical\_features **=** [feature **for** feature **in** df**.**columns **if** df[feature]**.**dtype **==** 'O']

*# print columns*

print('We have {} numerical features : {}'**.**format(len(numeric\_features), numeric\_features))

print('\nWe have {} categorical features : {}'**.**format(len(categorical\_features), categorical\_features))

df**.**head(2)

**Adding columns for "Total Score" and "Average"**

df['total score'] **=** df['math\_score'] **+** df['reading\_score'] **+** df['writing\_score']

df['average'] **=** df['total score']**/**3

df**.**head()

reading\_full **=** df[df['reading\_score'] **==** 100]['average']**.**count()

writing\_full **=** df[df['writing\_score'] **==** 100]['average']**.**count()

math\_full **=** df[df['math\_score'] **==** 100]['average']**.**count()

print(f'Number of students with full marks in Maths: {math\_full}')

print(f'Number of students with full marks in Writing: {writing\_full}')

print(f'Number of students with full marks in Reading: {reading\_full}')

reading\_less\_20 **=** df[df['reading\_score'] **<=** 20]['average']**.**count()

writing\_less\_20 **=** df[df['writing\_score'] **<=** 20]['average']**.**count()

math\_less\_20 **=** df[df['math\_score'] **<=** 20]['average']**.**count()

print(f'Number of students with less than 20 marks in Maths: {math\_less\_20}')

print(f'Number of students with less than 20 marks in Writing: {writing\_less\_20}')

print(f'Number of students with less than 20 marks in Reading: {reading\_less\_20}')

**Exploring Data ( Visualization )**

fig, axs **=** plt**.**subplots(1, 2, figsize**=**(15, 7))

plt**.**subplot(121)

sns**.**histplot(data**=**df,x**=**'average',bins**=**30,kde**=True**,color**=**'g')

plt**.**subplot(122)

sns**.**histplot(data**=**df,x**=**'average',kde**=True**,hue**=**'gender')

plt**.**show()

fig, axs **=** plt**.**subplots(1, 2, figsize**=**(15, 7))

plt**.**subplot(121)

sns**.**histplot(data**=**df,x**=**'total score',bins**=**30,kde**=True**,color**=**'g')

plt**.**subplot(122)

sns**.**histplot(data**=**df,x**=**'total score',kde**=True**,hue**=**'gender')

plt**.**show()

**Female students tend to perform well then male students.**

plt**.**subplots(1,3,figsize**=**(25,6))

plt**.**subplot(141)

sns**.**histplot(data**=**df,x**=**'average',kde**=True**,hue**=**'lunch')

plt**.**subplot(142)

sns**.**histplot(data**=**df[df**.**gender**==**'female'],x**=**'average',kde**=True**,hue**=**'lunch')

plt**.**subplot(143)

sns**.**histplot(data**=**df[df**.**gender**==**'male'],x**=**'average',kde**=True**,hue**=**'lunch')

plt**.**show()

**Standard lunch helps perform well in exams.**

**Standard lunch helps perform well in exams be it a male or a female.**

plt**.**subplots(1,3,figsize**=**(25,6))

plt**.**subplot(141)

ax **=**sns**.**histplot(data**=**df,x**=**'average',kde**=True**,hue**=**'parental\_level\_of\_education')

plt**.**subplot(142)

ax **=**sns**.**histplot(data**=**df[df**.**gender**==**'male'],x**=**'average',kde**=True**,hue**=**'parental\_level\_of\_education')

plt**.**subplot(143)

ax **=**sns**.**histplot(data**=**df[df**.**gender**==**'female'],x**=**'average',kde**=True**,hue**=**'parental\_level\_of\_education')

plt**.**show()

**In general parent's education don't help student perform well in exam.**

**2nd plot shows that parent's whose education is of associate's degree or master's degree their male child tend to perform well in exam**

**3rd plot we can see there is no effect of parent's education on female students.**

plt**.**subplots(1,3,figsize**=**(25,6))

plt**.**subplot(141)

ax **=**sns**.**histplot(data**=**df,x**=**'average',kde**=True**,hue**=**'race\_ethnicity')

plt**.**subplot(142)

ax **=**sns**.**histplot(data**=**df[df**.**gender**==**'female'],x**=**'average',kde**=True**,hue**=**'race\_ethnicity')

plt**.**subplot(143)

ax **=**sns**.**histplot(data**=**df[df**.**gender**==**'male'],x**=**'average',kde**=True**,hue**=**'race\_ethnicity')

plt**.**show()

**Students of group A and group B tends to perform poorly in exam.**

**Students of group A and group B tends to perform poorly in exam irrespective of whether they are male or female**

**Maximumum score of students in all three subjects**

plt**.**figure(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**.**violinplot(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()

**Multivariate analysis using pieplot**

plt**.**rcParams['figure.figsize'] **=** (30, 12)

plt**.**subplot(1, 5, 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(1, 5, 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(1, 5, 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')

plt**.**subplot(1, 5, 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(1, 5, 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**.**axis('off')

plt**.**tight\_layout()

plt**.**grid()

plt**.**show()

**GENDER COLUMN**

Fax**=**plt**.**subplots(1,2,figsize**=**(20,10))

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**=**20)

plt**.**pie(x**=**df['gender']**.**value\_counts(),labels**=**['Male','Female'],explode**=**[0,0.1],autopct**=**'%1.1f%%',shadow**=True**,colors**=**['#ff4d4d','#ff8000'])

plt**.**show()

gender\_group **=** df**.**groupby('gender')**.**mean()

gender\_group

plt**.**figure(figsize**=**(10, 8))

*# Define gender\_group with only numeric columns and calculate the average*

gender\_group **=** df**.**groupby('gender')[numeric\_features]**.**mean()

gender\_group['average'] **=** df**.**groupby('gender')['average']**.**mean()

X **=** ['Total Average', 'Math Average']

female\_scores **=** [gender\_group['average']['female'], gender\_group['math\_score']['female']]

male\_scores **=** [gender\_group['average']['male'], gender\_group['math\_score']['male']]

X\_axis **=** np**.**arange(len(X))

plt**.**bar(X\_axis **-** 0.2, male\_scores, 0.4, label **=** 'Male')

plt**.**bar(X\_axis **+** 0.2, female\_scores, 0.4, label **=** 'Female')

plt**.**xticks(X\_axis, X)

plt**.**ylabel("Marks")

plt**.**title("Total average v/s Math average marks of both the genders", fontweight**=**'bold')

plt**.**legend()

plt**.**show()

**RACE/EHNICITY COLUMN**

f,ax**=**plt**.**subplots(1,2,figsize**=**(20,10))

sns**.**countplot(x**=**df['race\_ethnicity'],data**=**df,palette **=** 'bright',ax**=**ax[0],saturation**=**0.95)

**for** container **in** ax[0]**.**containers:

ax[0]**.**bar\_label(container,color**=**'black',size**=**20)

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**)

plt**.**show()

Group\_data2**=**df**.**groupby('race\_ethnicity')

f,ax**=**plt**.**subplots(1,3,figsize**=**(20,8))

sns**.**barplot(x**=**Group\_data2['math\_score']**.**mean()**.**index,y**=**Group\_data2['math\_score']**.**mean()**.**values,palette **=** 'mako',ax**=**ax[0])

ax[0]**.**set\_title('Math score',color**=**'#005ce6',size**=**20)

**for** container **in** ax[0]**.**containers:

ax[0]**.**bar\_label(container,color**=**'black',size**=**15)

sns**.**barplot(x**=**Group\_data2['reading\_score']**.**mean()**.**index,y**=**Group\_data2['reading\_score']**.**mean()**.**values,palette **=** 'flare',ax**=**ax[1])

ax[1]**.**set\_title('Reading score',color**=**'#005ce6',size**=**20)

**for** container **in** ax[1]**.**containers:

ax[1]**.**bar\_label(container,color**=**'black',size**=**15)

sns**.**barplot(x**=**Group\_data2['writing\_score']**.**mean()**.**index,y**=**Group\_data2['writing\_score']**.**mean()**.**values,palette **=** 'coolwarm',ax**=**ax[2])

ax[2]**.**set\_title('Writing score',color**=**'#005ce6',size**=**20)

**for** container **in** ax[2]**.**containers:

ax[2]**.**bar\_label(container,color**=**'black',size**=**15)

**PARENTAL LEVEL OF EDUCATION COLUMN**

plt**.**rcParams['figure.figsize'] **=** (15, 9)

plt**.**style**.**use('fivethirtyeight')

sns**.**countplot(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()

*# Select only numeric columns for aggregation*

df**.**groupby('parental\_level\_of\_education')[numeric\_features]**.**mean()**.**plot(kind**=**'barh', figsize**=**(10, 10))

plt**.**legend(bbox\_to\_anchor**=**(1.05, 1), loc**=**2, borderaxespad**=**0.)

plt**.**show()

**LUNCH COLUMN**

plt**.**rcParams['figure.figsize'] **=** (15, 9)

plt**.**style**.**use('seaborn-talk')

sns**.**countplot(df['lunch'], palette **=** 'PuBu')

plt**.**title('Comparison of different types of lunch', fontweight **=** 30, fontsize **=** 20)

plt**.**xlabel('types of lunch')

plt**.**ylabel('count')

plt**.**show()

f,ax**=**plt**.**subplots(1,2,figsize**=**(20,8))

sns**.**countplot(x**=**df['parental\_level\_of\_education'],data**=**df,palette **=** 'bright',hue**=**'test\_preparation\_course',saturation**=**0.95,ax**=**ax[0])

ax[0]**.**set\_title('Students\_vs\_test\_preparation\_course',color**=**'black',size**=**25)

**for** container **in** ax[0]**.**containers:

ax[0]**.**bar\_label(container,color**=**'black',size**=**20)

sns**.**countplot(x**=**df['parental\_level\_of\_education'],data**=**df,palette **=** 'bright',hue**=**'lunch',saturation**=**0.95,ax**=**ax[1])

**for** container **in** ax[1]**.**containers:

ax[1]**.**bar\_label(container,color**=**'black',size**=**20)

**TEST PREPARATION COURSE COLUMN**

plt**.**figure(figsize**=**(12,6))

plt**.**subplot(2,2,1)

sns**.**barplot (x**=**df['lunch'], y**=**df['math\_score'], hue**=**df['test\_preparation\_course'])

plt**.**subplot(2,2,2)

sns**.**barplot (x**=**df['lunch'], y**=**df['reading\_score'], hue**=**df['test\_preparation\_course'])

plt**.**subplot(2,2,3)

sns**.**barplot (x**=**df['lunch'], y**=**df['writing\_score'], hue**=**df['test\_preparation\_course'])

**CHECKING OUTLIERS**

plt**.**subplots(1,4,figsize**=**(16,5))

plt**.**subplot(141)

sns**.**boxplot(df['math\_score'],color**=**'skyblue')

plt**.**subplot(142)

sns**.**boxplot(df['reading\_score'],color**=**'hotpink')

plt**.**subplot(143)

sns**.**boxplot(df['writing\_score'],color**=**'yellow')

plt**.**subplot(144)

sns**.**boxplot(df['average'],color**=**'lightgreen')

plt**.**show()

**MUTIVARIATE ANALYSIS USING PAIRPLOT**

sns**.**pairplot(df,hue **=** 'gender')

plt**.**show()

**Data\_ingestion.py**

import os

import sys

from src.exception import CustomException

from src.logger import logging

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from dataclasses import dataclass

from src.components.data\_transformation import DataTransformationConfig

from src.components.data\_transformation import DataTransformation

from src.components.model\_trainer import ModelTrainerConfig

from src.components.model\_trainer import ModelTrainer

#any input that is required is given through this data ingestion config class

@dataclass

class DataIngestionConfig:

train\_data\_path:str=os.path.join('artifacts','train.csv')

test\_data\_path:str=os.path.join('artifacts','test.csv')

raw\_data\_path:str=os.path.join('artifacts','data.csv')

# we can add more parameters if required in future

# it will tell components where to save the data and where to read the data from

# this is the config class for data ingestioncomponent

class DataIngestion:

def \_\_init\_\_(self):

self.ingestion\_config=DataIngestionConfig()

# this will create an instance of the config class and store it in the ingestion\_config attribute

# this will be used to access the parameters of the config class

# so that we can use it in the data ingestion component

def initiate\_data\_ingestion(self):

logging.info("Entered the data ingestion method or component")

try:

df=pd.read\_csv('notebook/data/stud.csv')

# reading the data from the csv file stored in the notebook/data folder locally on to the system

logging.info('Read the dataset as dataframe')

os.makedirs(os.path.dirname(self.ingestion\_config.train\_data\_path),exist\_ok=True)

# this will create the directory if it does not exist

df.to\_csv(self.ingestion\_config.raw\_data\_path,index=False,header=True)

# this will save the data in the raw data path

logging.info('Train test split initiated')

train\_set,test\_set=train\_test\_split(df,test\_size=0.2,random\_state=42)

# this will split the data into train and test set

train\_set.to\_csv(self.ingestion\_config.train\_data\_path,index=False,header=True)

test\_set.to\_csv(self.ingestion\_config.test\_data\_path,index=False,header=True)

# this will save the train and test set in the respective paths

logging.info('Ingestion of the data is completed')

return (

self.ingestion\_config.train\_data\_path,

self.ingestion\_config.test\_data\_path

)

# this will return the train and test data paths when the data ingestion is completed successfully

except Exception as e:

raise CustomException(e,sys)

# this will raise the custom exception if there is any error in the code

if \_\_name\_\_=="\_\_main\_\_":

obj=DataIngestion()

train\_data,test\_data=obj.initiate\_data\_ingestion()

# this will create an instance of the data ingestion class and call the

# initiate\_data\_ingestion method to start the data ingestion process

# this will be used to test the data ingestion component

data\_transformation = DataTransformation()

train\_arr,test\_arr,\_=data\_transformation.initiate\_data\_transformation(train\_data,test\_data)

# this will create an instance of the data transformation class and

# call the initiate\_data\_transformation method to start the data transformation process

# this will be used to test the data transformation component

model\_trainer = ModelTrainer()

print(model\_trainer.initiate\_model\_trainer(train\_arr, test\_arr))

**Data\_transformation.py**

# This module contains the DataTransformation class which is responsible for transforming the data

# and creating a preprocessor object that can be used to preprocess the data

import os

import sys

import pandas as pd

import numpy as np

from src.exception import CustomException

from src.logger import logging

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder,StandardScaler

from dataclasses import dataclass

from src.utlis import save\_object

@dataclass

class DataTransformationConfig:

# This class defines the configuration for data transformation

preprocessor\_obj\_file\_path: str = os.path.join('artifacts', 'preprocessor.pkl')

# This will store the path where the preprocessor will be saved

#preprocessed\_data\_path: str = os.path.join('artifacts', 'preprocessed\_data.csv')

#transformed\_data\_path: str = os.path.join('artifacts', 'transformed\_data.csv')

class DataTransformation:

def \_\_init\_\_(self):

self.data\_transformation\_config = DataTransformationConfig()

# This will create an instance of the DataTransformationConfig

# class and store it in the data\_transformation\_config attribute

def get\_data\_transformer\_object(self):

'''

this function is responsible for data transformation baed on the numerical and

categorical columns

It will create a preprocessor object that will be used to preprocess the data

'''

try:

numerical\_columns = ["writing\_score", "reading\_score"]

categorical\_columns = [

"gender",

"race\_ethnicity",

"parental\_level\_of\_education",

"lunch",

"test\_preparation\_course",

]

# These are the numerical and categorical columns in the dataset

# We will use these columns to create the preprocessor object

#create a pipeline for numerical columns

num\_pipeline = Pipeline(

steps=[

("imputer",SimpleImputer(strategy="median")),

("scaler",StandardScaler())

]

)

#this will run on the numerical columns of the training dataset

#create a pipeline for categorical columns

cat\_pipeline = Pipeline(

steps=[

("imputer",SimpleImputer(strategy="most\_frequent")),

("one\_hot\_encoder",OneHotEncoder()),

("scaler",StandardScaler(with\_mean=False))

]

)

logging.info(f"Numerical columns {numerical\_columns} are defined")

logging.info(f"and categorical columns {categorical\_columns} are defined")

preprcoessor = ColumnTransformer(

[

('num\_pipeline', num\_pipeline, numerical\_columns),

# this will apply the numerical pipeline to the numerical columns

('cat\_pipeline', cat\_pipeline, categorical\_columns)

# this will apply the categorical pipeline to the categorical columns

]

)

'''/This will create a preprocessor object that will apply

the numerical and categorical pipelines to the respective columns'''

logging.info("Preprocessor object created")

return preprcoessor

except Exception as e:

raise CustomException(e, sys)

# this will raise the custom exception if there is any error in the code

def initiate\_data\_transformation(self, train\_path, test\_path):

logging.info("Entered the data transformation method or component")

try:

train\_df = pd.read\_csv(train\_path)# reading the train data from the csv file

test\_df = pd.read\_csv(test\_path)# reading the test data from the csv file

logging.info("Read the train and test dataframes")

logging.info("Obtaining preprocessing object")

# this will call the get\_data\_transformer\_object function to create a preprocessor object

preprocessing\_obj = self.get\_data\_transformer\_object()

# this will create a preprocessor object that will be used to preprocess the data

target\_column\_name = "math\_score"# this is the target column in the dataset

# this will be used to separate the target feature from the input features

numerical\_columns = ["writing\_score", "reading\_score"]

# these are the numerical columns in the dataset

input\_feature\_train\_df = train\_df.drop(columns=[target\_column\_name], axis=1)

# dropping the target column from the train dataframe

# this will create a dataframe with only the input features

target\_feature\_train\_df = train\_df[target\_column\_name]

input\_feature\_test\_df = test\_df.drop(columns=[target\_column\_name], axis=1)

target\_feature\_test\_df = test\_df[target\_column\_name]

logging.info(

f"applying preprocessing object on training and testing datasets:

{input\_feature\_train\_df.shape}, {input\_feature\_test\_df.shape}"

)

# transform the training and testing data

input\_feature\_train\_arr = preprocessing\_obj.fit\_transform(input\_feature\_train\_df)

input\_feature\_test\_arr = preprocessing\_obj.fit\_transform(input\_feature\_test\_df)

'''this will apply the preprocessor object on the input

features of the train and test dataframes'''

train\_arr = np.c\_[

input\_feature\_train\_arr,

np.array(target\_feature\_train\_df)]

# this will combine the input features and target feature into a single array

# convert the input features and target feature to numpy arrays

test\_arr = np.c\_[

input\_feature\_test\_arr,

np.array(target\_feature\_test\_df)]

# this will combine the input features and target feature into a single array

# convert the input features and target feature to numpy arrays

logging.info("saved preprocessor object")

# save the preprocessor object to the specified path

# convert the target feature to numpy array

save\_object(

file\_path=self.data\_transformation\_config.preprocessor\_obj\_file\_path,

obj=preprocessing\_obj

)# this will save the preprocessor object to the specified path

logging.info("Data transformation completed")

return (

train\_arr,

test\_arr,

self.data\_transformation\_config.preprocessor\_obj\_file\_path

)

except Exception as e:

raise CustomException(e, sys)

**Model\_training.py**

import os

import sys

from dataclasses import dataclass

from catboost import CatBoostRegressor

from sklearn.ensemble import (

AdaBoostRegressor,

GradientBoostingRegressor,

RandomForestRegressor,

)

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor

from xgboost import XGBRegressor

from src.exception import CustomException

from src.logger import logging

from src.utlis import save\_object,evaluate\_models

@dataclass

class ModelTrainerConfig:

# Path where the trained model will be saved

trained\_model\_file\_path=os.path.join("artifacts","model.pkl")

# Path to save the trained model

# Path where the model artifacts will be saved

class ModelTrainer:

"""

This class is responsible for training the machine learning model

Model Trainer class to train and evaluate machine learning models.

This class handles the training of various regression models, evaluates their performance,"""

def \_\_init\_\_(self):

# Initialize the model trainer configuration

self.model\_trainer\_config=ModelTrainerConfig()

def initiate\_model\_trainer(self,train\_array,test\_array):

""" This method initiates the model training process.

Args:

train\_array (numpy.ndarray): The training data array.

test\_array (numpy.ndarray): The testing data array.

Returns:

float: The R-squared score of the best model on the test data.

"""

try:

logging.info("Split training and test input data")

X\_train,y\_train,X\_test,y\_test=(

train\_array[:,:-1],#this will take all rows and all columns except last column as X\_train\_value

train\_array[:,-1], #this will make the last column as y\_train\_value

test\_array[:,:-1],#this will take all rows and all columns except last column as X\_test\_value

test\_array[:,-1]#this will make the last column as y\_test\_value

)

models = {

"Random Forest": RandomForestRegressor(),

"Decision Tree": DecisionTreeRegressor(),

"Gradient Boosting": GradientBoostingRegressor(),

"Linear Regression": LinearRegression(),

# "XGBRegressor": XGBRegressor(),

# "CatBoosting Regressor": CatBoostRegressor(verbose=False),

# "AdaBoost Regressor": AdaBoostRegressor(),

}

model\_report:dict= evaluate\_models(

X\_train=X\_train,

y\_train=y\_train,

X\_test=X\_test,

y\_test=y\_test, models=models)

for model\_name in models:

print(f"Model: {model\_name} \n {model\_report[model\_name]} \n")

logging.info(f"Model Report: {model\_report}")

## To get best model score from dict

best\_model\_score = max(sorted(model\_report.values()))

## To get best model name from dict

best\_model\_name = list(model\_report.keys())[

list(model\_report.values()).index(best\_model\_score)

]

best\_model = models[best\_model\_name]

if best\_model\_score<0.6:

raise CustomException("No best model found")

logging.info(f"Best found model on both training and testing dataset")

save\_object(

file\_path=self.model\_trainer\_config.trained\_model\_file\_path,

obj=best\_model

)

predicted=best\_model.predict(X\_test)

r2\_square = r2\_score(y\_test, predicted)

return r2\_square

except Exception as e:

raise CustomException(e,sys)

**Exception.py**

import sys

from src.logger import logging

def error\_message\_details(error,error\_details:sys):

    \_,\_,exc\_tb=error\_details.exc\_info()

    file\_name=exc\_tb.tb\_frame.f\_code.co\_filename

    error\_message="Error occured in python script name[{0}] at

    line number[{1}] error message[{2}]".format(

        file\_name,

        exc\_tb.tb\_lineno,

        str(error)

        )

    return error\_message

class CustomException(Exception):

    def \_\_init\_\_(self,error\_message,error\_details:sys):

        super().\_\_init\_\_(error\_message)

        self.error\_message=error\_message\_details(error\_message,error\_details=error\_details)

    def \_\_str\_\_(self):

         return self.error\_message

**Logger.py**

import logging

import os

from datetime import datetime

LOG\_FILE=f"{datetime.now().strftime('%Y-%m-%d')}.log"

logs\_path=os.path.join(os.getcwd(),"logs",LOG\_FILE)  # logs path

os.makedirs(logs\_path,exist\_ok=True)

LOG\_FILE\_PATH=os.path.join(logs\_path,LOG\_FILE)

logging.basicConfig(

    filename=LOG\_FILE\_PATH,

    format="[%(asctime)s] %(lineno)d %(name)s - %(levelname)s - %(message)s",

    level=logging.INFO

    )

**Utils.py**

import os

import sys

import numpy as np

import pandas as pd

import dill

import pickle

from sklearn.metrics import r2\_score

from sklearn.model\_selection import GridSearchCV

from src.exception import CustomException

def save\_object(file\_path, obj):

    '''Function to save an object to a file using pickle.

    Args:

        file\_path (str): Path where the object will be saved.

        obj (object): The object to be saved.

    '''

    try:

        dir\_path = os.path.dirname(file\_path)

        os.makedirs(dir\_path, exist\_ok=True)

        with open(file\_path, 'wb') as file\_obj:

            dill.dump(obj, file\_obj)

    except Exception as e:

        raise CustomException(e, sys)

def evaluate\_models(X\_train, y\_train,X\_test,y\_test,models):

    try:

        report = {}

        for i in range(len(list(models))):

            model = list(models.values())[i]# Get the model from the dictionary

            #param\_grid = param[list(models.keys())[i]]  # Get the parameters for the model

            # gs = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=3, n\_jobs=-1, verbose=2)

            # gs.fit(X\_train, y\_train)

            # model.set\_params(\*\*gs.best\_params\_)# Set the best parameters found by GridSearchCV

            model.fit(X\_train, y\_train)  # Train model

            y\_train\_pred = model.predict(X\_train)

            y\_test\_pred = model.predict(X\_test)

            train\_model\_score = r2\_score(y\_train, y\_train\_pred)

            test\_model\_score = r2\_score(y\_test, y\_test\_pred)

            report[list(models.keys())[i]] = test\_model\_score

        return report

    except Exception as e:

        raise CustomException(e, sys)

def load\_object(file\_path):

    try:

        with open(file\_path, "rb") as file\_obj:

            return pickle.load(file\_obj)

    except Exception as e:

        raise CustomException(e, sys)

**Predict\_pipeline.py**

import sys

import pandas as pd

from src.exception import CustomException

from src.utlis import load\_object

import os

class PredictPipeline:

    def \_\_init\_\_(self):

        pass

    def predict(self,features):

        try:

            model\_path=os.path.join("artifacts","model.pkl")

            preprocessor\_path=os.path.join('artifacts','preprocessor.pkl')

            print("Before Loading")

            model=load\_object(file\_path=model\_path)

            preprocessor=load\_object(file\_path=preprocessor\_path)

            print("After Loading")

            data\_scaled=preprocessor.transform(features)

            preds=model.predict(data\_scaled)

            return preds

        except Exception as e:

            raise CustomException(e,sys)

class CustomData:

    def \_\_init\_\_(  self,

        gender: str,

        race\_ethnicity: str,

        parental\_level\_of\_education,

        lunch: str,

        test\_preparation\_course: str,

        reading\_score: int,

        writing\_score: int):

        self.gender = gender

        self.race\_ethnicity = race\_ethnicity

        self.parental\_level\_of\_education = parental\_level\_of\_education

        self.lunch = lunch

        self.test\_preparation\_course = test\_preparation\_course

        self.reading\_score = reading\_score

        self.writing\_score = writing\_score

    def get\_data\_as\_data\_frame(self):

        try:

            custom\_data\_input\_dict = {

                "gender": [self.gender],

                "race\_ethnicity": [self.race\_ethnicity],

                "parental\_level\_of\_education": [self.parental\_level\_of\_education],

                "lunch": [self.lunch],

                "test\_preparation\_course": [self.test\_preparation\_course],

                "reading\_score": [self.reading\_score],

                "writing\_score": [self.writing\_score],

            }

            return pd.DataFrame(custom\_data\_input\_dict)

        except Exception as e:

            raise CustomException(e, sys)

**Home.html**

<!DOCTYPE html>

<html>

<head>

    <title>Student Exam Performance Indicator</title>

    <style>

        body {

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

            background-color: #f0f2f5;

            display: flex;

            justify-content: center;

            align-items: center;

            min-height: 100vh;

            margin: 0;

        }

        .login {

            background-color: #ffffff;

            padding: 40px;

            border-radius: 12px;

            box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);

            width: 100%;

            max-width: 500px;

            box-sizing: border-box;

            text-align: center;

        }

        h1 {

            color: #333;

            margin-bottom: 30px;

            font-size: 2em;

            font-weight: 600;

        }

        form {

            text-align: left;

        }

        legend {

            font-size: 1.5em;

            color: #555;

            margin-bottom: 25px;

            text-align: center;

            font-weight: 500;

        }

        .mb-3 {

            margin-bottom: 20px;

        }

        label.form-label {

            display: block;

            margin-bottom: 8px;

            color: #333;

            font-weight: 500;

            font-size: 0.95em;

        }

        .form-control {

            width: 100%;

            padding: 12px 15px;

            border: 1px solid #ced4da;

            border-radius: 8px;

            box-sizing: border-box;

            font-size: 1em;

            color: #495057;

            transition: border-color 0.2s ease-in-out, box-shadow 0.2s ease-in-out;

            -webkit-appearance: none; /\* Remove default arrow for selects in some browsers \*/

            -moz-appearance: none;

            appearance: none;

            background-image: linear-gradient(45deg, transparent 50%, #ced4da 50%), linear-gradient(135deg, #ced4da 50%, transparent 50%);

            background-position: calc(100% - 20px) calc(1em + 2px), calc(100% - 15px) calc(1em + 2px);

            background-size: 5px 5px, 5px 5px;

            background-repeat: no-repeat;

        }

        .form-control:focus {

            border-color: #007bff;

            box-shadow: 0 0 0 0.2rem rgba(0, 123, 255, 0.25);

            outline: none;

        }

        /\* Style for disabled option in select \*/

        option.placeholder {

            color: #6c757d;

        }

        /\* Style for specific option values (optional, just for better visibility) \*/

        option[value="male"],

        option[value="female"],

        option[value="group A"],

        option[value="group B"],

        option[value="group C"],

        option[value="group D"],

        option[value="group E"],

        option[value="associate's degree"],

        option[value="bachelor's degree"],

        option[value="high school"],

        option[value="master's degree"],

        option[value="some college"],

        option[value="some high school"],

        option[value="free/reduced"],

        option[value="standard"],

        option[value="none"],

        option[value="completed"] {

            color: #333;

        }

        .btn-primary {

            background-color: #007bff;

            color: white;

            padding: 12px 25px;

            border: none;

            border-radius: 8px;

            cursor: pointer;

            font-size: 1.1em;

            font-weight: 600;

            transition: background-color 0.2s ease-in-out, transform 0.1s ease;

            width: 100%;

            margin-top: 10px;

        }

        .btn-primary:hover {

            background-color: #0056b3;

            transform: translateY(-2px);

        }

        .btn-primary:active {

            transform: translateY(0);

        }

        h2 {

            color: #28a745; /\* Green color for positive results \*/

            margin-top: 30px;

            font-size: 1.8em;

            font-weight: 700;

            text-align: center;

            background-color: #e9f7ef;

            padding: 15px;

            border-radius: 8px;

            border: 1px solid #c3e6cb;

        }

    </style>

</head>

<body>

    <div class="login">

        <h1>Student Exam Performance Indicator</h1>

        <form action="{{ url\_for('predict\_datapoint')}}" method="post">

            <h1>

                <legend>Student Exam Performance Prediction</legend>

            </h1>

            <div class="mb-3">

                <label class="form-label">Gender</label>

                <select class="form-control" name="gender" required>

                    <option class="placeholder" selected disabled value="">Select your Gender</option>

                    <option value="male">Male</option>

                    <option value="female">Female</option>

                </select>

            </div>

            <div class="mb-3">

                <label class="form-label">Race or Ethnicity</label>

                <select class="form-control" name="ethnicity" required>

                    <option class="placeholder" selected disabled value="">Select Ethnicity</option>

                    <option value="group A">Group A</option>

                    <option value="group B">Group B</option>

                    <option value="group C">Group C</option>

                    <option value="group D">Group D</option>

                    <option value="group E">Group E</option>

                </select>

            </div>

            <div class="mb-3">

                <label class="form-label">Parental Level of Education</label>

                <select class="form-control" name="parental\_level\_of\_education" required>

                    <option class="placeholder" selected disabled value="">Select Parent Education</option>

                    <option value="associate's degree">associate's degree</option>

                    <option value="bachelor's degree">bachelor's degree</option>

                    <option value="high school">high school</option>

                    <option value="master's degree">master's degree</option>

                    <option value="some college">some college</option>

                    <option value="some high school">some high school</option>

                </select>

            </div>

            <div class="mb-3">

                <label class="form-label">Lunch Type</label>

                <select class="form-control" name="lunch" required>

                    <option class="placeholder" selected disabled value="">Select Lunch Type</option>

                    <option value="free/reduced">free/reduced</option>

                    <option value="standard">standard</option>

                </select>

            </div>

            <div class="mb-3">

                <label class="form-label">Test preparation Course</label>

                <select class="form-control" name="test\_preparation\_course" required>

                    <option class="placeholder" selected disabled value="">Select Test\_course</option>

                    <option value="none">None</option>

                    <option value="completed">Completed</option>

                </select>

            </div>

            <div class="mb-3">

                <label class="form-label">Writing Score out of 100</label>

                <input class="form-control" type="number" name="reading\_score"

                    placeholder="Enter your Reading score" min='0' max='100' required />

            </div>

            <div class="mb-3">

                <label class="form-label">Reading Score out of 100</label>

                <input class="form-control" type="number" name="writing\_score"

                    placeholder="Enter your Writing Score" min='0' max='100' required />

            </div>

            <div class="mb-3">

                <input class="btn btn-primary" type="submit" value="Predict your Maths Score" />

            </div>

        </form>

    </div>

</body>

</html>

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Prediction Result</title>

    <script src="https://cdn.tailwindcss.com"></script>

    <style>

        /\* Custom styles for Inter font and general body styling \*/

        body {

            font-family: 'Inter', sans-serif;

            background-color: #f0f2f5; /\* Light gray background \*/

        }

    </style>

</head>

<body class="flex items-center justify-center min-h-screen p-4">

    <div class="bg-white p-8 md:p-12 rounded-xl shadow-2xl w-full max-w-md text-center">

        <h1 class="text-3xl md:text-4xl font-extrabold text-blue-700 mb-6">

            Your Predicted Maths Score

        </h1>

        {% if results is defined and results is not none %}

            <div class="bg-green-100 border-2 border-green-400 text-green-800 p-6 rounded-lg mb-8">

                <p class="text-2xl md:text-3xl font-bold">

                    {{ results }}

                </p>

            </div>

            <p class="text-gray-600 text-lg mb-8">

                This is your estimated Maths score based on the information you provided.

            </p>

        {% else %}

            <div class="bg-yellow-100 border-2 border-yellow-400 text-yellow-800 p-6 rounded-lg mb-8">

                <p class="text-xl md:text-2xl font-semibold">

                    No prediction available yet.

                </p>

            </div>

            <p class="text-gray-600 text-lg mb-8">

                Please go back to the form and submit your details to get a prediction.

            </p>

        {% endif %}

        <a href="{{ url\_for('predict\_datapoint') }}" class="inline-block bg-blue-600 hover:bg-blue-700 text-white font-semibold py-3 px-6 rounded-lg transition duration-300 ease-in-out transform hover:scale-105 shadow-md">

            Go Back to Form

        </a>

    </div>

</body>

</html>

**App.py**

from flask import Flask,request,render\_template, redirect, url\_for # Import redirect and url\_for

import numpy as np

import pandas as pd

from sklearn.preprocessing import StandardScaler

from src.pipeline.predict\_pipeline import CustomData,PredictPipeline

application=Flask(\_\_name\_\_)

app=application

## Route for a home page

@app.route('/')

def index():

    return redirect(url\_for('predict\_datapoint'))

@app.route('/predictdata',methods=['GET','POST'])

def predict\_datapoint():

    if request.method=='GET':

        return render\_template('home.html')

    else:

        data=CustomData(

            gender=request.form.get('gender'),

            race\_ethnicity=request.form.get('ethnicity'),

            parental\_level\_of\_education=request.form.get('parental\_level\_of\_education'),

            lunch=request.form.get('lunch'),

            test\_preparation\_course=request.form.get('test\_preparation\_course'),

            reading\_score=float(request.form.get('writing\_score')),

            writing\_score=float(request.form.get('reading\_score'))

        )

        pred\_df=data.get\_data\_as\_data\_frame()

        print(pred\_df)

        print("Before Prediction")

        predict\_pipeline=PredictPipeline()

        print("Mid Prediction")

        results=predict\_pipeline.predict(pred\_df) #this will return the prediction results

        print("after Prediction")

        return render\_template('index.html',results=results)

if \_\_name\_\_=="\_\_main\_\_":

    app.run(host="0.0.0.0",debug=True)

**gitignore**

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# C extensions

\*.so

# Distribution / packaging

.Python

build/

develop-eggs/

dist/

downloads/

eggs/

.eggs/

lib/

lib64/

parts/

sdist/

var/

wheels/

share/python-wheels/

\*.egg-info/

.installed.cfg

\*.egg

MANIFEST

# PyInstaller

#  Usually these files are written by a python script from a template

#  before PyInstaller builds the exe, so as to inject date/other infos into it.

\*.manifest

\*.spec

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.nox/

.coverage

.coverage.\*

.cache

nosetests.xml

coverage.xml

\*.cover

\*.py,cover

.hypothesis/

.pytest\_cache/

cover/

# Translations

\*.mo

\*.pot

# Django stuff:

\*.log

local\_settings.py

db.sqlite3

db.sqlite3-journal

# Flask stuff:

instance/

.webassets-cache

# Scrapy stuff:

.scrapy

# Sphinx documentation

docs/\_build/

# PyBuilder

.pybuilder/

target/

# Jupyter Notebook

.ipynb\_checkpoints

# IPython

profile\_default/

ipython\_config.py

# pyenv

#   For a library or package, you might want to ignore these files since the code is

#   intended to run in multiple environments; otherwise, check them in:

# .python-version

# pipenv

#   According to pypa/pipenv#598, it is recommended to include Pipfile.lock in version control.

#   However, in case of collaboration, if having platform-specific dependencies or dependencies

#   having no cross-platform support, pipenv may install dependencies that don't work, or not

#   install all needed dependencies.

#Pipfile.lock

# UV

#   Similar to Pipfile.lock, it is generally recommended to include uv.lock in version control.

#   This is especially recommended for binary packages to ensure reproducibility, and is more

#   commonly ignored for libraries.

#uv.lock

# poetry

#   Similar to Pipfile.lock, it is generally recommended to include poetry.lock in version control.

#   This is especially recommended for binary packages to ensure reproducibility, and is more

#   commonly ignored for libraries.

#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-file-to-version-control

#poetry.lock

# pdm

#   Similar to Pipfile.lock, it is generally recommended to include pdm.lock in version control.

#pdm.lock

#   pdm stores project-wide configurations in .pdm.toml, but it is recommended to not include it

#   in version control.

#   https://pdm.fming.dev/latest/usage/project/#working-with-version-control

.pdm.toml

.pdm-python

.pdm-build/

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and github.com/pdm-project/pdm

\_\_pypackages\_\_/

# Celery stuff

celerybeat-schedule

celerybeat.pid

# SageMath parsed files

\*.sage.py

# Environments

.env

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

# Spyder project settings

.spyderproject

.spyproject

# Rope project settings

.ropeproject

# mkdocs documentation

/site

# mypy

.mypy\_cache/

.dmypy.json

dmypy.json

# Pyre type checker

.pyre/

# pytype static type analyzer

.pytype/

# Cython debug symbols

cython\_debug/

# PyCharm

#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can

#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore

#  and can be added to the global gitignore or merged into this file.  For a more nuclear

#  option (not recommended) you can uncomment the following to ignore the entire idea folder.

#.idea/

# PyPI configuration file

.pypirc

# 5. Testing & Implementation

The project has successfully completed all key tasks related to data handling, model development, and integration:

* **Problem Definition:** Completed, defining project scope, goals, and measurable objectives.
* **Dataset Acquisition:** Completed, dataset downloaded and prepared from Kaggle.
* **Data Cleaning & Preprocessing:** Completed, handled missing values, normalization, and formatting.
* **Exploratory Data Analysis (EDA):** Completed, identified trends, distributions, and patterns in data.
* **Feature Selection & Engineering:** Completed, selected impactful attributes for model input.
* **Model Development:** Completed, algorithms like Linear Regression and Random Forest implemented.
* **Data Integration:** Completed, data is efficiently loaded and managed locally for model input/output.

# 6. Limitations & Future Scope

## Limitations Faced:

During the project's development lifecycle, several challenges were encountered and successfully addressed, providing valuable learning experiences:

* **Data Quality and Completeness:** A significant initial hurdle was the presence of numerous missing values and inconsistencies within the raw dataset. This necessitated extensive application of imputation techniques (e.g., mean, median, mode imputation) and normalization methods (e.g., Min-Max scaling, Z-score normalization) to transform the data into a clean and model-ready format.
* **Feature Selection Complexity:** Identifying the most influential features from a large pool of potential attributes proved to be a complex task. This required extensive experimentation with various statistical techniques (e.g., correlation analysis, chi-squared tests) and model-based selection methods (e.g., Recursive Feature Elimination, feature importance from tree-based models) to optimize model performance and interpretability.
* **Model Accuracy vs. Interpretability:** A persistent challenge involved striking the right balance between achieving highly accurate predictive models (often complex ensemble methods like Random Forests or Gradient Boosting) and maintaining interpretability (simpler models like Linear Regression). Interpretability was deemed crucial for educators to understand and trust the system's predictions, necessitating careful model selection and explanation techniques.
* **Scalability Concerns:** Early iterations of the system design were not fully optimized for scalability, particularly concerning the efficient handling of larger datasets and the potential for real-time data ingestion. This limitation was addressed through subsequent refactoring of the data processing pipeline and model serving architecture to ensure robust performance under increased loads.
* **Collaboration & Version Control:** Managing multiple updates, resolving dependency conflicts, and ensuring consistent documentation across a collaborative development environment on GitHub required the implementation of strict branching strategies (e.g., Git Flow), thorough pull request reviews, and disciplined commit strategies to maintain code integrity and team synchronization.

## Future Scope:

Building upon the solid foundation established by the completed project, several avenues for future enhancement and expansion have been identified:

* **Further research into advanced machine learning techniques for improved accuracy and interpretability:** This could involve exploring deep learning models, more sophisticated ensemble methods, or explainable AI (XAI) techniques to provide even more precise predictions and clearer insights into the factors driving student performance.
* **Potential integration with existing educational management systems for broader applicability:** Seamless integration with platforms commonly used by schools and universities (e.g., student information systems, learning management systems) would significantly enhance the system's utility and adoption by automating data flow and embedding predictions directly into existing workflows.
* **Exploration of cloud-based deployment options for enhanced scalability and accessibility:** Migrating the system to a cloud platform (e.g., AWS, Google Cloud, Azure) would provide superior scalability to handle larger student populations and real-time data streams, improve accessibility for users across different locations, and leverage managed services for easier maintenance and operations.
* **Development of a feedback loop mechanism:** Implementing a system where educators can provide feedback on the accuracy of predictions and the effectiveness of interventions would allow for continuous model retraining and improvement, making the system more adaptive and precise over time.
* **Expansion to include personalized learning recommendations:** Beyond prediction, the system could evolve to suggest personalized learning paths, resources, or intervention strategies based on individual student performance patterns and predicted challenges.

# 7. Conclusion

The "Student Marks Performance Prediction" project has reached its successful completion, marking a significant achievement in leveraging machine learning for educational insights. The system's core functionality, encompassing robust data processing, accurate predictive modeling, and efficient local data storage, has been thoroughly implemented and validated. The integration of these components provides a reliable tool capable of analyzing both academic and non-academic features of students to forecast their future performance. This capability is crucial for educators and institutions, enabling them to identify students who may require additional support proactively and to tailor interventions effectively. This project highlights the potential of data-based methods to improve learning outcomes and create a more supportive educational environment.

# Bibliography

* [Predicting Students' Performance Using Machine Learning Techniques](https://www.researchgate.net/publication/332893829)
* Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter, Third Edition
* Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems
* <https://www.w3schools.com/datascience/>
* <https://www.kaggle.com/datasets/spscientist/students-performance-in-exams?datasetId=74977>