1

Table of Contents

Abstract

This study utilized five machine learning (ML) models to classify students' academic performance using demographic attributes and exam scores. After Exploratory Data Analysis (EDA) and preprocessing, handling class imbalance with SMOTE, removing outliers via IQR, and encoding categorical variables, five M L models were tested: Logistic Regression (LR), Naive Bayes (NB), AdaBoost (AB), Support Vector Classifier (SVC), and K-Nearest Neighbors (KNN). SVM achieved the best performance with 99\% accuracy, precision, recall, and F1-score, whereas KNN, NB, and LR also performed well. AdaBoost exhibited the weakest results. Overall, the findings highlight the potential of machine learning, particularly ensemble methods, as decision-support tools for identifying at-risk students and designing targeted interventions while ensuring fairness and data privacy.

Introduction

Predicting student performance is a key focus of educational data mining and learning analytics, supported by increased access to datasets and advances in AI/ML (Romero & Ventura, 2010; Siemens & Baker, 2013). Accurate classification helps identify at-risk learners, guide interventions, and support evidence-based teaching (Kotsiantis, Pierrakeas, & Pintelas, 2010), especially where socioeconomic factors influence outcomes (Tinto, 1993). ML enables the discovery of complex patterns beyond traditional methods (Baker & Inventado, 2014) using algorithms such as decision trees, ensembles, K-nearest neighbors (KNN), support vector machines (SVM), and logistic regression (Breiman, 2001; Chen & Guestrin, 2016), with ensemble methods such as AdaBoost and Gradient Boosting offering strong accuracy and generalizability (Hastie, Tibshirani, & Friedman, 2009). This study uses supervised learning on a dataset of 1,000 students to (1) analyze demographic-performance relationships via EDA, (2) evaluate classification models, and (3) identify optimal predictive approaches while addressing issues of fairness, bias, and privacy.

Background

Student performance prediction lies within the fields of educational data mining (EDM) and learning analytics (LA). EDM develops methods to analyze learner data (Romero & Ventura, 2010), whereas LA focuses on measuring and analyzing data to optimize learning (Siemens & Baker, 2013). Academic success is shaped by demographic, socioeconomic, and behavioral factors such as parental education (Tinto, 1993) and study habits (Kuh et al., 2008). Machine learning enhances this analysis using models such as Logistic Regression (Kotsiantis, Pierrakeas & Pintelas, 2010), Naïve Bayes, and KNN, as well as modern approaches such as SVC and ensemble methods such as AdaBoost (Breiman, 2001; Chen & Guestrin, 2016; Hastie, Tibshirani & Friedman, 2009). Exploratory Data Analysis (Baker & Inventado, 2014) helps identify patterns, whereas SMOTE addresses class imbalance (Chawla et al., 2002).

Aim and Objectives

This study aimed to evaluate and compare the performance of multiple supervised machine learning algorithms for classifying students' academic performance using demographic attributes and exam scores while ensuring fairness, bias mitigation, and data privacy.

- Perform an EDA to discern patterns, correlations, and anomalies within the dataset.
- Preprocess the data by employing techniques such as encoding, scaling, outlier removal, and addressing class imbalance through SMOTE.

- Implement and assess classification models such as Logistic Regression, Naive Bayes Classifier, Ada Boost Classifier, SVC, and KNN.
- Evaluate model performance by precision, recall, F1-score, and accuracy.

Dataset Description

The dataset entitled *Students Performance in Exams* encompasses data pertaining to students' demographic and educational backgrounds, in conjunction with their examination scores across three subjects: mathematics, reading, and writing. This dataset is publicly accessible on the Kaggle platform and comprises 1000 records .

Figure 1 shows the code and result for initial five rows of the dataset.

```
df = pd.read_csv('/content/StudentsPerformance.csv')
df.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

Figure 1: Code used for loading in the dataset and output.

Figure 2 shows the code and output of the Pandas info() method, which summarizes the dataset structure.

```
1 print("\nDataset Info:")
2 print(df.info())
3
```

```
Dataset 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
None
```

Figure 2: Code and output to display basic information about the dataset.

Problem to be Addressed

Accurate prediction of student performance is crucial in today's education system. This project focuses on developing AI models that use demographic attributes and examination scores to predict grades. The goal is to help educators identify at-risk students early and design targeted interventions while ensuring fairness.

Artificial Intelligence Models

Support Vector Classifier (SVC)

Support vector classifiers (SVC) are supervised learning models that find the optimal hyperplane to separate data points of different classes (Cortes & Vapnik, 1995). SVCs perform well in high-dimensional spaces and are effective when the number of features exceeds the number of observations.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, instance-based learning algorithm used for classification and regression. It predicts the target value based on the majority class or average of the nearest neighbors in the feature space (Cover & Hart, 1967).

Naïve Bayes

Naive Bayes is a family of probabilistic classifiers based on Bayes' theorem, which assumes independence among features (Rish, 2001). Despite this strong assumption, it performs remarkably well in many real-world applications, particularly in text classification and spam detection.

Logistic Regression

Logistic Regression is a popular model for binary and multiclass classification problems. It estimates the probability of class membership using the logistic function, modeling the relationship between independent and categorical dependent variables (Cox, 1958).

Ada Boost Classifier

Adaptive Boosting (AdaBoost) is an ensemble learning method that combines multiple weak learners, typically decision stumps, into a strong classifier (Freund & Schapire, 1997). It assigns higher weights to misclassified instances in each iteration, forcing subsequent learners to focus on more difficult cases.

Exploratory Data Analysis (EDA)

Check Missing Values

Figure 3 shows the check for missing values, confirming that all columns have zero null entries and that the dataset is complete.

```
# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
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
```

Figure 3: Code used to check for missing values and output.

Check Target Column

We first created a new target variable, average score, as shown the code and output in Figure 4, which represents the mean of the math, reading, and writing scores for each student.

```
df['average_score'] = df[['math score', 'reading score', 'writing score']].mean(axis=1)
df['average_score']=df['average_score'].astype(int)
df
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	average_score
0	female	group B	bachelor's degree	standard	none	72	72	74	72
1	female	group C	some college	standard	completed	69	90	88	82
2	female	group B	master's degree	standard	none	90	95	93	92
3	male	group A	associate's degree	free/reduced	none	47	57	44	49
4	male	group C	some college	standard	none	76	78	75	76
...	...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95	94
996	male	group C	high school	free/reduced	none	62	55	55	57
997	female	group C	high school	free/reduced	completed	59	71	65	65
998	female	group D	some college	standard	completed	68	78	77	74
999	female	group D	some college	free/reduced	none	77	86	86	83

1000 rows × 9 columns

Figure 4: Code to show average score, assigned grades, and grade distribution and output.

Figure 5 shows how the average scores were converted into letter grades (A + to F).

```
def assign_grade(score):
    if score >= 90:
        return 'A+'
    elif score >= 80:
        return 'A'
    elif score >= 70:
        return 'B'
    elif score >= 60:
        return 'C'
    elif score >= 50:
        return 'D'
    else:
        return 'F'

df['grade'] = df['average_score'].apply(assign_grade)

print("Grade Distribution:")
print(df['grade'].value_counts())
```

```
Grade Distribution:
grade
B     261
C     256
D     182
A     146
F      89
A+     52
Name: count, dtype: int64
```

Figure 5: Code Output for creating assigning grades (A+ to F).

Figures 6 and 7 show the student performance distribution. Most students scored between 60 and 80, with grades B and C being the most common, while F and A+ were rare.

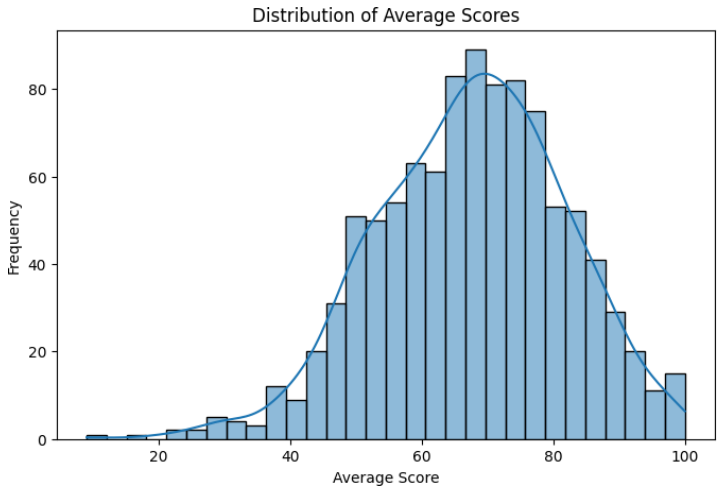


Figure 6: Distribution of students' average scores with histogram and KDE curve.

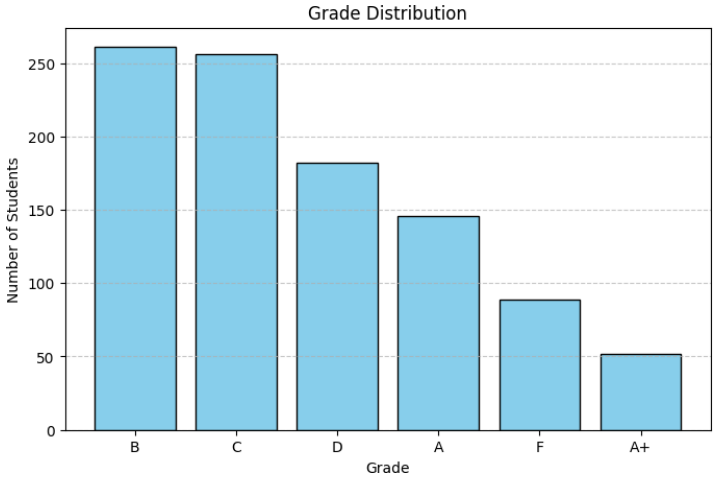


Figure 7: Distribution of students across grade categories based on average scores.

Check Outliers

We used the Interquartile Range (IQR) method to detect outliers, flagging values below Q1–1.5×IQR or above Q3+1.5×IQR. The results showed:

Math Score: 8 outliers

Reading Score: 6 outliers

Writing Score: 5 outliers

Average Score: 9 outliers

```python
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"\n Column: {col}")
    print(f"Number of outliers: {len(outliers)}")
```

```
 Column: math score
Number of outliers: 8

 Column: reading score
Number of outliers: 6

 Column: writing score
Number of outliers: 5

 Column: average_score
Number of outliers: 9
```

Figure 8: Outlier detection in numeric columns using the interquartile range (IQR) method.

To better visualize these results, we created a boxplot Figure 9.

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

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

plt.figure(figsize=(8,5))

sns.boxplot(data=df[numeric_cols])
plt.title("Boxplots for All Numeric Columns")
plt.ylabel('Score')

plt.show()

```

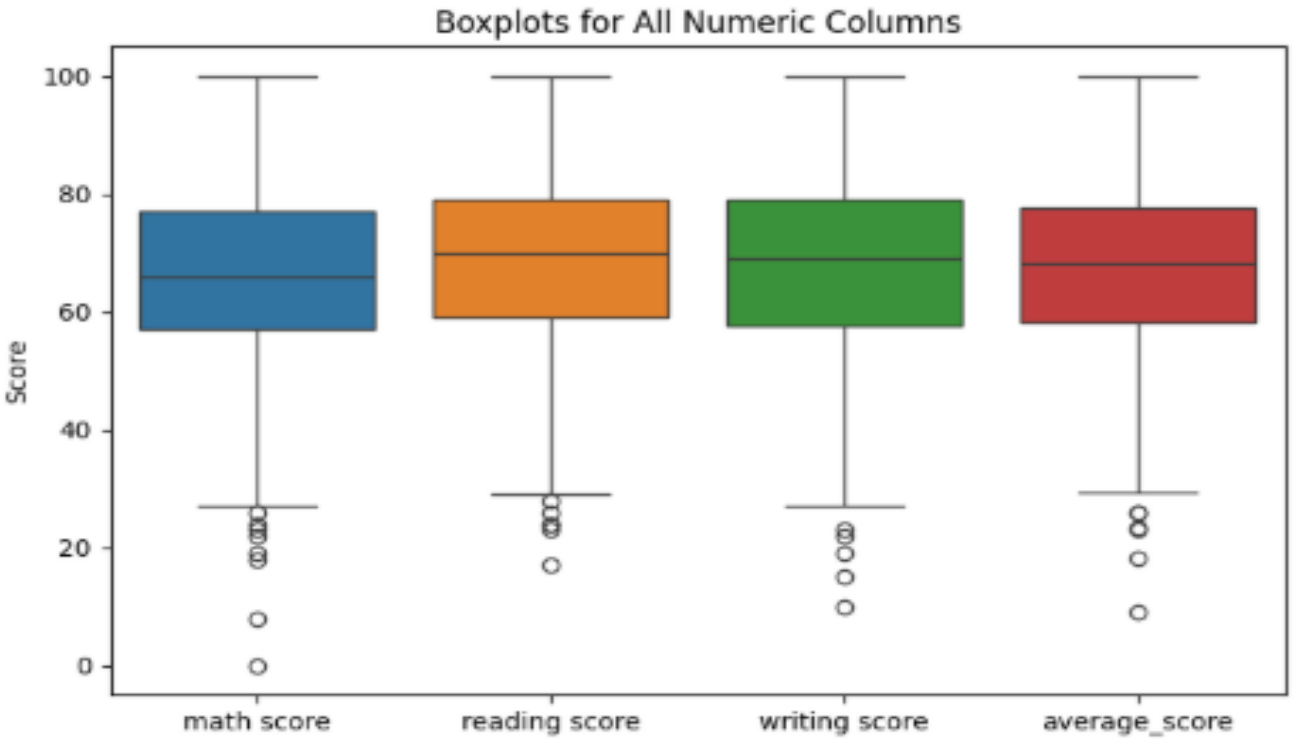


Figure 9: Code and output to Boxplot visualization of score distributions with outliers shown as points beyond the whiskers.

Visualise Correlations (Heatmap)

We created a heatmap, as shown in the code in Figure 10, to explore correlations between variables.

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

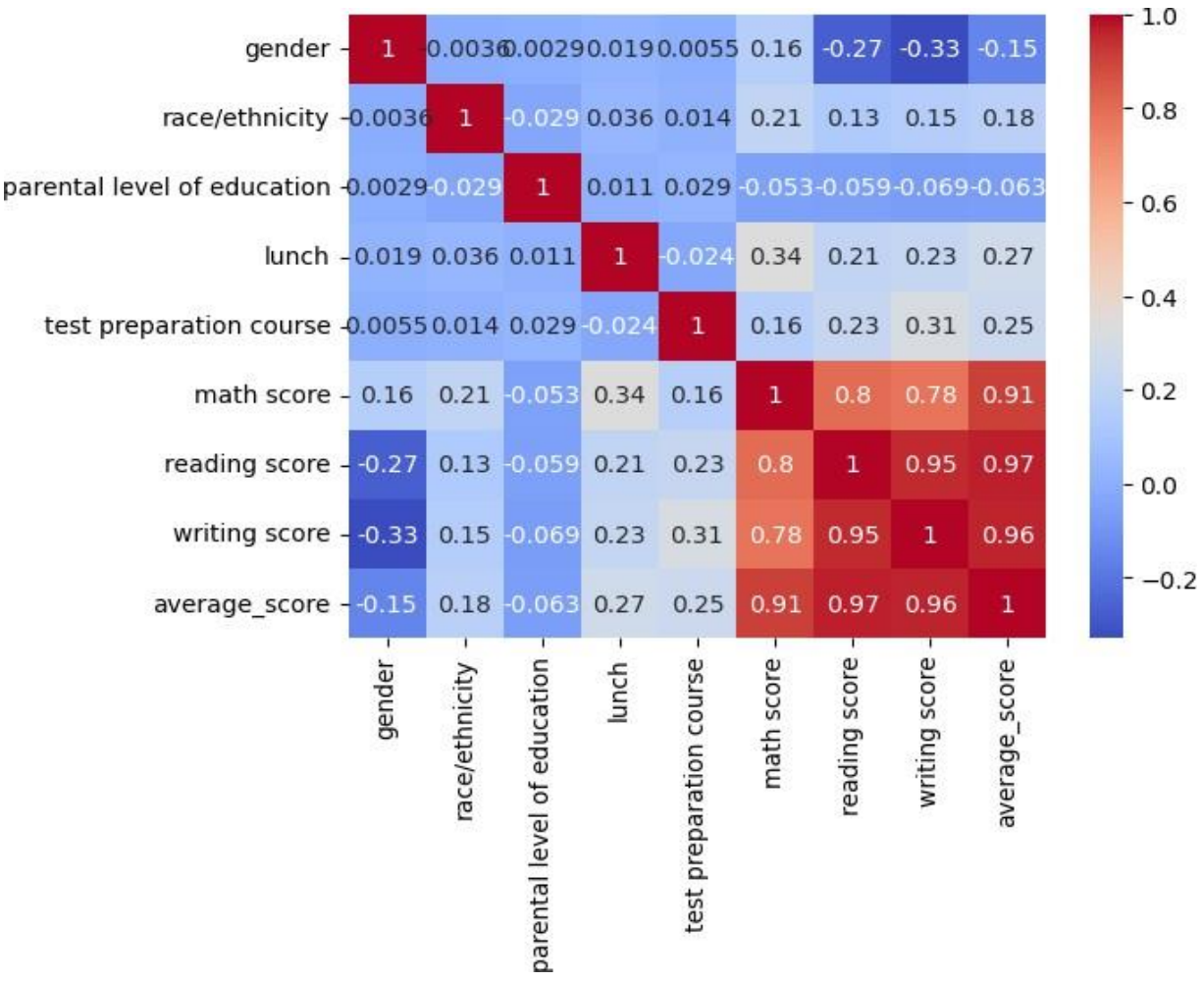


Figure 10: Code and output for correlation heatmap.

Figure 11 shows the relationship between the categorical features and the average scores. Gender had a weak negative correlation, while test preparation and lunch showed weak positive correlations. ANOVA revealed significant differences across parental education level and race/ethnicity.

```python
from scipy.stats import pointbiserialr, f_oneway

# Define target column
target = 'average_score'

# Dictionary of binary variables and their mapping
binary_vars = {
    'gender': {'male': 1, 'female': 0},
    'test preparation course': {'completed': 1, 'none': 0},
    'lunch': {'standard': 1, 'free/reduced': 0}
}

# List of multi-class categorical variables
multi_class_vars = ['parental level of education', 'race/ethnicity']

# --- Binary variables correlation ---
print("Correlation for Binary variables\n")
for col, mapping in binary_vars.items():
    df[col + '_encoded'] = df[col].map(mapping)
    r, p = pointbiserialr(df[col + '_encoded'], df[target])
    print(f"{col} → Correlation (r): {r:.2f}, p-value: {p:.4f}")

# --- Multi-class variables ANOVA ---
print("\nCorrelation for Multi-class variables\n")
for col in multi_class_vars:
    groups = [df[df[col] == level][target] for level in df[col].unique()]
    f_stat, p_val = f_oneway(*groups)
    print(f"{col} → F-statistic: {f_stat:.2f}, p-value: {p_val:.4f}")
```

```
Correlation for Binary variables

gender → Correlation (r): -0.13, p-value: 0.0000
test preparation course → Correlation (r): 0.26, p-value: 0.0000
lunch → Correlation (r): 0.29, p-value: 0.0000

Correlation for Multi-class variables

parental level of education → F-statistic: 10.75, p-value: 0.0000
race/ethnicity → F-statistic: 9.10, p-value: 0.0000
```

Figure 11: Code output to find the relationship between categorical features.

Plot Histogram

Figure 12 shows the histograms of the Reading, Writing, and Math scores. Reading and Writing follow a bell-shaped distribution (60–80 range), while math is slightly left-skewed, although all three show similar patterns, with most students near the average.

```
import seaborn as sns

plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
sns.histplot(df['reading score'], bins=30, kde=True, color='blue')
plt.title('Distribution of Reading Score')
plt.xlabel('Reading Score')
plt.ylabel('Frequency')


plt.subplot(1, 3, 2)
sns.histplot(df['writing score'], bins=30, kde=True, color='green')
plt.title('Distribution of Writing Score')
plt.xlabel('Writing Score')
plt.ylabel('Frequency')


plt.subplot(1, 3, 3)
sns.histplot(df['math score'], bins=30, kde=True, color='red')
plt.title('Distribution of Math Score')
plt.xlabel('Math Score')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

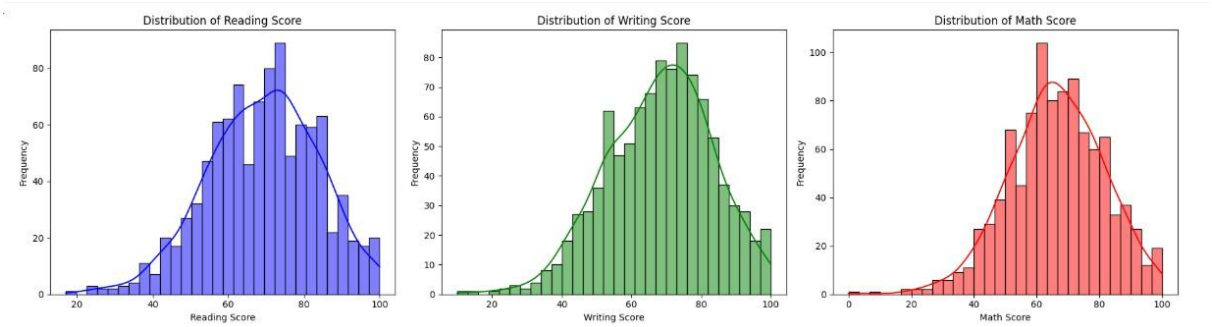


Figure 12: Code and output to Histograms displaying the distribution of Reading, Writing, and Math scores for each subject.

Visualise The Relationship Between Features

To observe the relationships between the numerical features, we created scatter plots, as shown in the code and results in Figure 13.

```python
import matplotlib.pyplot as plt

num_cols = ['math score', 'reading score', 'writing score', 'average_score']


fig, axes = plt.subplots(2, 3, figsize=(12, 8))
fig.suptitle("Scatter Plots Between Numerical Features", fontsize=16)

combinations = [
    ('math score', 'reading score'),
    ('math score', 'writing score'),
    ('math score', 'average_score'),
    ('reading score', 'writing score'),
    ('reading score', 'average_score'),
    ('writing score', 'average_score'),
]

for ax, (x, y) in zip(axes.flat, combinations):
    ax.scatter(df[x], df[y], alpha=0.6, color='blue')
    ax.set_xlabel(x)
    ax.set_ylabel(y)
    ax.set_title(f'{x} vs {y}')

plt.tight_layout()
plt.subplots_adjust
plt.show()
```

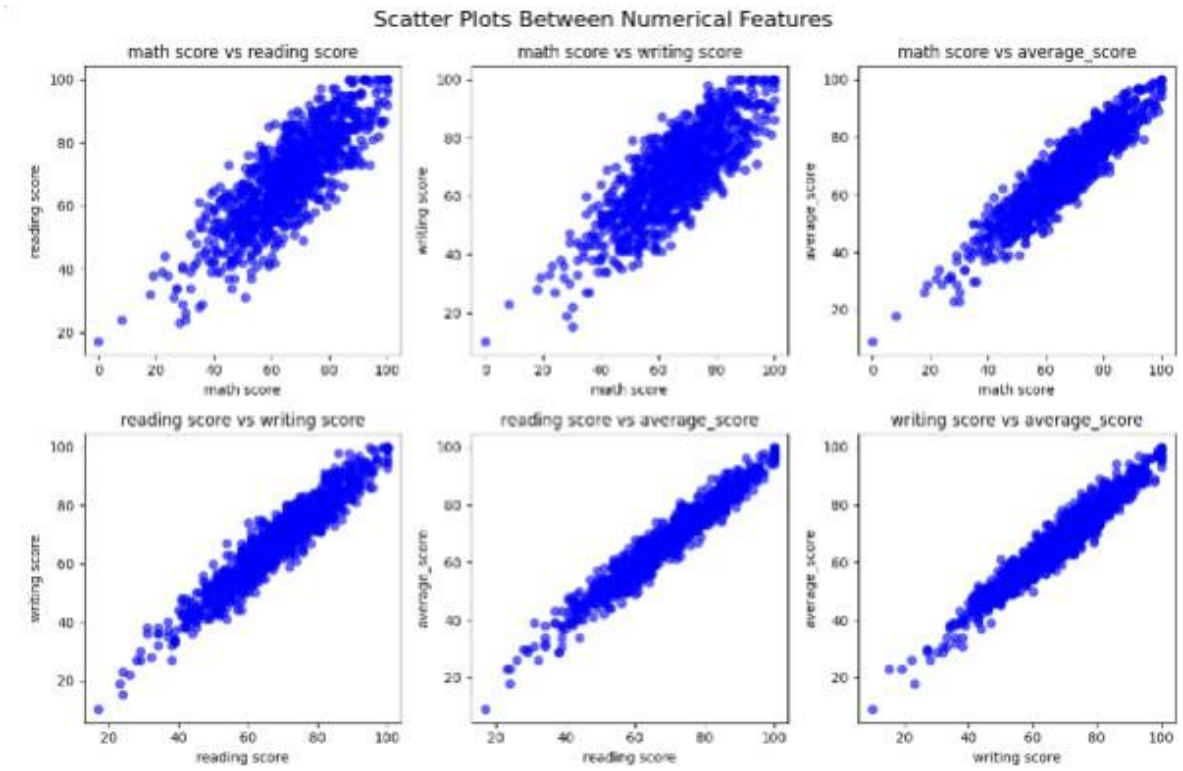


Figure 13: The code and Scatter plots between numerical features.

The code and result bar plots in Figure 14 show how different categories impact the average score.

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

fig, axes = plt.subplots(2, 3, figsize=(14,10))
fig.suptitle('Bar Plots: Categorical Features vs Average Score', fontsize=16)

sns.barplot(x='gender', y='average_score', data=df, ax=axes[0, 0])
axes[0, 0].set_title('Gender')

sns.barplot(x='lunch', y='average_score', data=df, ax=axes[0, 1])
axes[0, 1].set_title('Lunch')

sns.barplot(x='parental level of education', y='average_score', data=df, ax=axes[0, 2])
axes[0, 2].set_title('Parental Education')
axes[0, 2].tick_params(axis='x', rotation=45)

sns.barplot(x='test preparation course', y='average_score', data=df, ax=axes[1, 0])
axes[1, 0].set_title('Test Preparation')

sns.barplot(x='race/ethnicity', y='average_score', data=df, ax=axes[1, 1])
axes[1, 1].set_title('Race/Ethnicity')

axes[1, 2].axis('off')

plt.tight_layout()
plt.subplots_adjust(top=0.88)
plt.show()
```

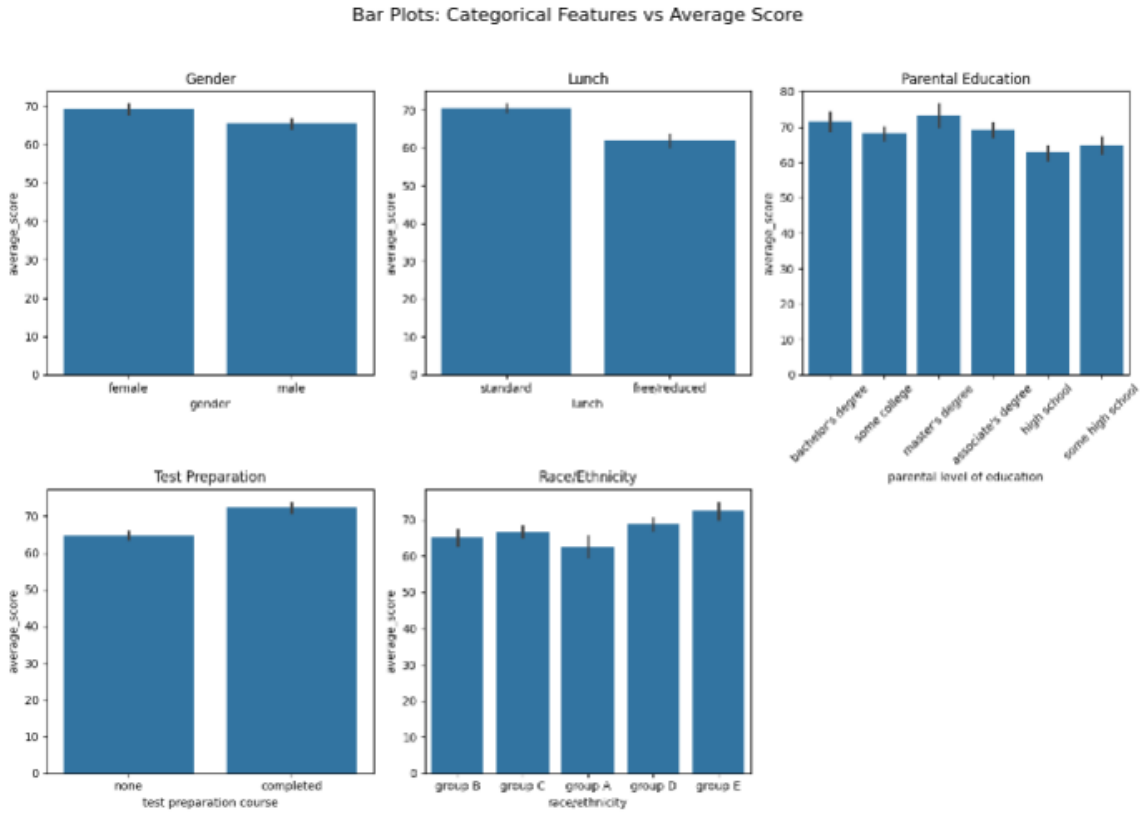


Figure 14: Code and output for Bar plots: categorical features vs average score.

The code for pie charts and result in Figure 15 show the proportion of each category in the dataset.

```
plt.figure(figsize=(16, 10))
plt.subplot(2, 3, 1)
df["gender"].value_counts().plot.pie(
    autopct="%1.1f%%", startangle=90, colors=['skyblue', 'lightpink'],
    wedgeprops=dict(edgecolor='w'))
plt.title('Gender Distribution')
plt.ylabel('')

plt.subplot(2, 3, 2)
df["lunch"].value_counts().plot.pie(
    autopct="%1.1f%%", startangle=90, colors=['lightgreen', 'orange'],
    wedgeprops=dict(edgecolor='w'))
plt.title('Lunch Type Distribution')
plt.ylabel('')

plt.subplot(2, 3, 3)
df["test preparation course"].value_counts().plot.pie(
    autopct="%1.1f%%", startangle=90, colors=['lightcoral', 'lightblue'],
    wedgeprops=dict(edgecolor='w'))
plt.title('Test Preparation Course')
plt.ylabel('')

plt.subplot(2, 3, 4)
df["parental level of education"].value_counts().plot.pie(
    autopct="%1.1f%%", startangle=90,
    wedgeprops=dict(edgecolor='w'))
plt.title('Parental Education Level Distribution')
plt.ylabel('')

plt.subplot(2, 3, 5)
df["race/ethnicity"].value_counts().plot.pie(
    autopct="%1.1f%%", startangle=90,
    wedgeprops=dict(edgecolor='w'))
plt.title('Race/Ethnicity Group Distribution')
plt.ylabel('')

plt.tight_layout()
plt.show()
```

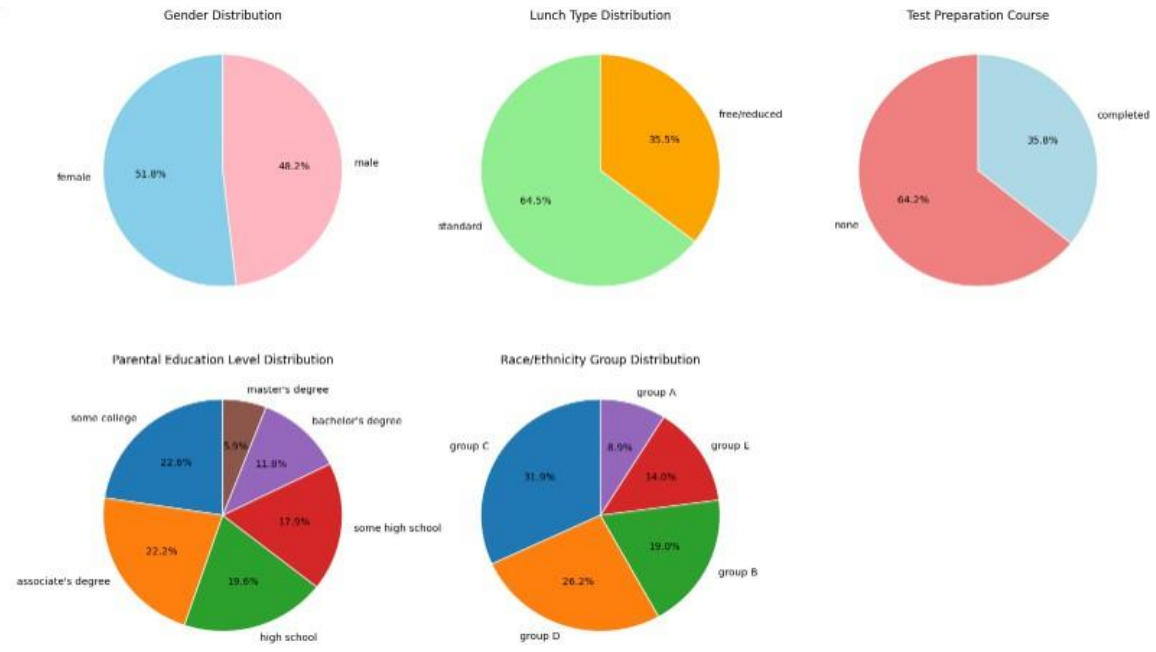


Figure 15: Code and result for Pie charts.

Data Cleaning

Handle Missing Values

During this analysis, we first checked the dataset for missing values. We found that there were no missing values in the dataset. Therefore, we did not need to take any action on them and proceeded to the next step of our analysis.

Handle Imbalance

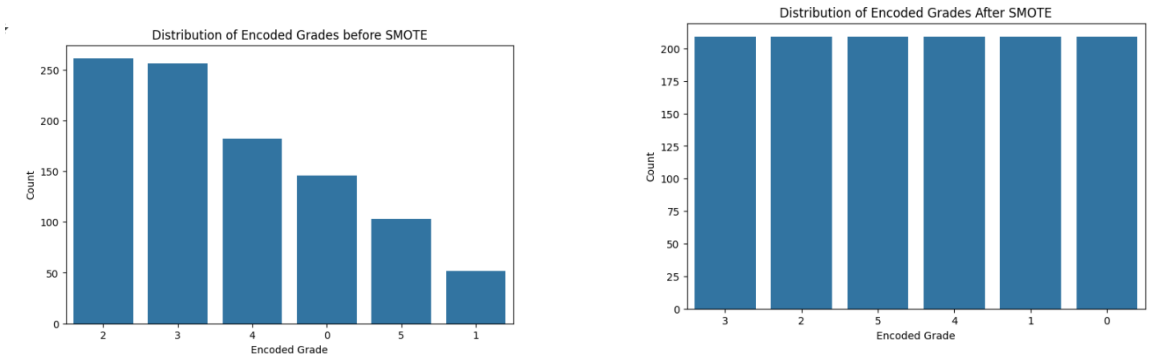
While building the classification model, we found that our target column, **'grades,** had an imbalanced dataset. To handle this imbalance, we used the Synthetic Minority Over-Sampling Technique (SMOTE), as shown in Figure 16.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

Figure 16: Code for balance grade column.

SMOTE creates new synthetic data points for the minority class, which helps balance the dataset. This technique allowed our model to better recognize minority classes. The graphs in Figures 17 show the data distribution before Figures 17a and after Figures 17b applying SMOTE.

```
plt.figure(figsize=(8, 5))
sns.countplot(x=y, order=y.value_counts().index)
plt.title('Distribution of Grades before SMOTE')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(8, 5))
sns.countplot(x=y_train_res, order=y_train_res.value_counts().index)
plt.title('Distribution of Grades After SMOTE')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.show()
```



(a) Before imbalance (b) After imbalance

Figure 17: Code and output for comparison of data before and after imbalance handling.

Handle Outliers

We handled the previously identified outliers using the IQR method, as shown in Figure 18. This process removed 14 outliers, reducing the dataset from 1,000 rows to 986 rows and making the data cleaner and more reliable.

```python
numeric_cols_to_check = ['math score', 'reading score', 'writing score', 'average_score']
df_no_outliers = df.copy()

for col in numeric_cols_to_check:
    Q1 = df_no_outliers[col].quantile(0.25)
    Q3 = df_no_outliers[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df_no_outliers = df_no_outliers[(df_no_outliers[col] >= lower_bound) & (df_no_outliers[col] <= upper_bound)]
print("Shape of DataFrame before removing outliers:", df.shape)
print("Shape of DataFrame after removing outliers:", df_no_outliers.shape)

df = df_no_outliers
for col in numeric_cols_to_check:
    plt.figure(figsize=(6, 1.5))
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot of {col}\n")
    plt.show()
```

Figure 18: Code to handle outliers.

The boxplots in Figure 19 display math, reading, writing, and average scores on their original scale (0–100). All subjects showed similar distributions, with median scores of approximately 65–70. An outlier was present in the math scores, representing a lower value, whereas no clear outliers appeared in the reading, writing, or average scores.

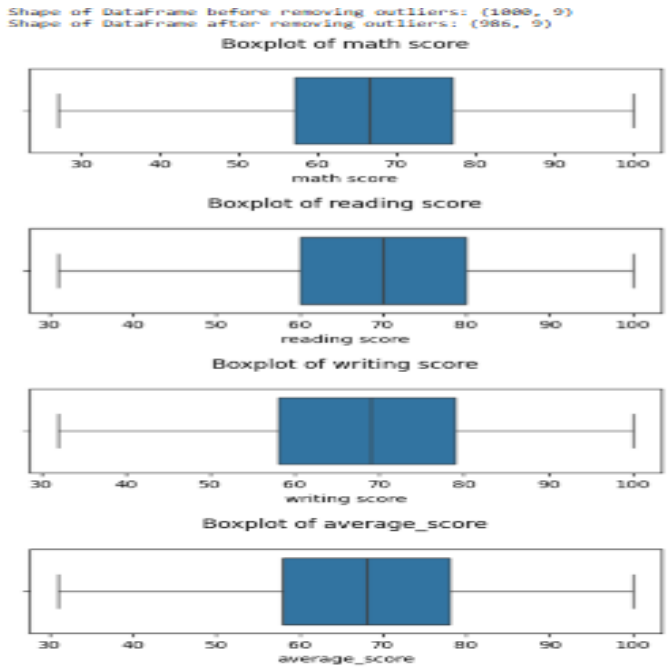


Figure 19: Boxplot of math score, reading score, writing score and average score.

Data Preprocessing

Separate Features (X) and Target Column (Y)

Before building the machine learning models, we separated the features (all scores except average_score) into **X** and the target column (average_score) into **y** as shown in Figure 20. We then split the dataset into 80% training and 20% testing sets using train_test_split from scikit-learn to ensure proper training and an unbiased performance evaluation.

```
from sklearn.model_selection import train_test_split
X = df.drop(columns=['grade'])
y = df['grade']

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, stratify=y)
```

Figure 20: Code to separate features (X) and target column (Y)

Feature Selection

For feature selection, we first removed the non-numeric columns for the initial analysis. The correlation heatmap in Figure 10 and the scatter plots in Figure 13 show that the math, reading, and writing scores had a strong positive correlation with average_score; therefore, we selected these as key features. The remaining categorical columns were label-encoded and included in the model.

Model Training, Evaluation and Testing

We used five ML models, namely, LR, SVC, Naïve Bayes, AdaBoost, and KNN, to classify student performance. Class imbalance was fixed with resampling, and models were tested using 5-fold stratified cross-validation. Predictions were evaluated with precision, recall, F1-score, and confusion matrices, giving insights into how well each model distinguished between grade categories (A+ to F).

Result Overview

The classification performance of five supervised models LR Table 1, NB Table 2, AB (Table 3), SVC Table 4, and KNN Table 5 was evaluated using precision, recall, F1- score, and support. The results revealed distinct differences in the predictive performance across the models.

Table 1: Classification Report for Logistic Regression

Class	Precision	Recall	F1-score	Support
0	0.92	0.90	0.91	209
1	0.94	0.98	0.96	209
2	0.93	0.89	0.91	209
3	0.92	0.94	0.92	209
4	0.92	0.92	0.92	209
5	0.95	0.97	0.96	209
Accuracy			**0.93**	1254
Macro Avg	0.93	0.93	0.93	1254
Weighted Avg	0.93	0.93	0.93	1254

Table 2: Classification Report for Naive Bayes Classifier

Class	Precision	Recall	F1-score	Support
0	0.96	0.95	0.95	209
1	0.98	0.99	0.98	209
2	0.95	0.94	0.94	209
3	0.95	0.94	0.95	209
4	0.93	0.96	0.95	209
5	0.98	0.96	0.97	209
Accuracy			**0.96**	1254
Macro Avg	0.96	0.96	0.96	1254
Weighted Avg	0.96	0.96	0.96	1254

Table 3: Classification Report for Ada BoostClassifier

Class	Precision	Recall	F1-score	Support
0	0.33	0.60	0.43	209
1	1.00	0.20	0.33	209
2	0.28	0.40	0.33	209
3	0.30	0.60	0.40	209
4	0.49	0.20	0.28	209
5	1.00	0.20	0.33	209
Accuracy			**0.36**	1254
Macro Avg	0.57	0.36	0.35	1254
Weighted Avg	0.57	0.36	0.35	1254

Table 4: Classification Report for Support Vector Classifier

Class	Precision	Recall	F1-score	Support
0	0.98	0.96	0.97	209
1	0.96	1.00	0.98	209
2	1.00	0.98	0.99	209
3	1.00	1.00	1.00	209
4	1.00	0.99	0.99	209
5	0.99	1.00	1.00	209
Accuracy			**0.91**	1254
Macro Avg	0.91	0.91	0.90	1254
Weighted Avg	0.91	0.91	0.91	1254

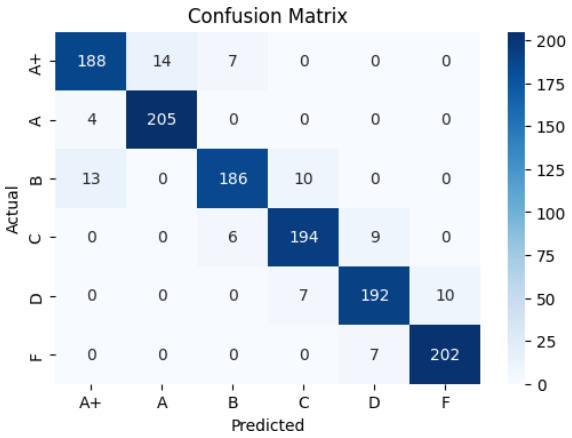
Table 5: Classification Report for K-Nearest Neighbors Classifier

Class	Precision	Recall	F1-score	Support
0	0.98	0.96	0.97	209
1	0.96	1.00	0.98	209
2	1.00	0.98	0.99	209
3	1.00	1.00	1.00	209
4	1.00	0.99	0.99	209
5	0.99	1.00	1.00	209
Accuracy			**0.99**	1254
Macro Avg	0.99	0.99	0.99	1254
Weighted Avg	0.99	0.99	0.99	1254

Figures 21 to 25 show the code and confusion matrices of all five classifiers. Overall, SVC and KNN were the most reliable, followed by logistic regression and NB, with AB performing the weakest.

```python
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold, cross_validate
model = LogisticRegression(max_iter=1000, random_state=42)


cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X_train_res, y_train_res, cv=cv)
cm = confusion_matrix(y_train_res, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['A+', 'A', 'B', 'C', 'D', 'F'],
            yticklabels=['A+', 'A', 'B', 'C', 'D', 'F'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

(a) Code

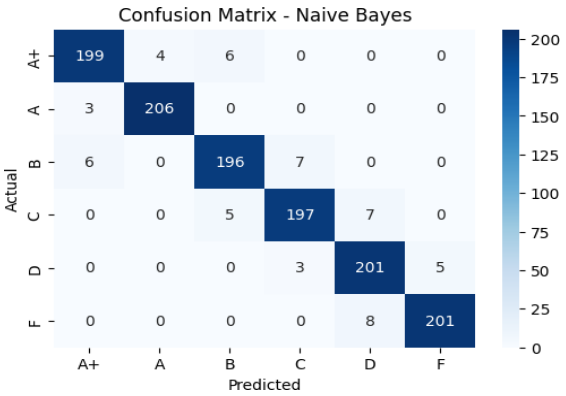


(b) Confusion Matrix

Figure 21: Code and Confusion Matrix for LR.

```python
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X_train_res, y_train_res, cv=cv)
cm = confusion_matrix(y_train_res, y_pred)

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['A+', 'A', 'B', 'C', 'D', 'F'],
            yticklabels=['A+', 'A', 'B', 'C', 'D', 'F'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Naive Bayes")
plt.show()
```

(a) Code

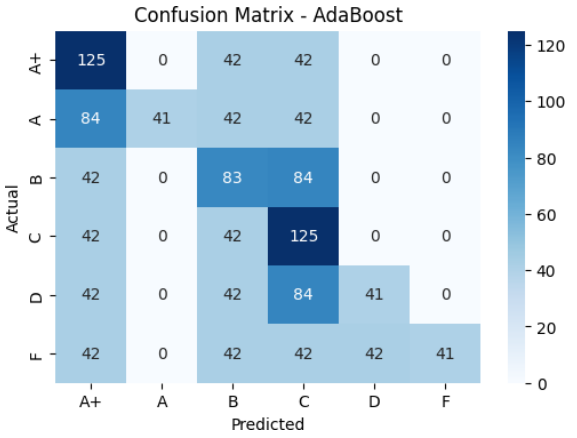


(b) Confusion Matrix

Figure 22: Code and Confusion Matrix for NB Classifier.

```python
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import AdaBoostClassifier

# Initialize AdaBoost model
model = AdaBoostClassifier(n_estimators=100, random_state=42)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X_train_res, y_train_res, cv=cv)
cm = confusion_matrix(y_train_res, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['A+', 'A', 'B', 'C', 'D', 'F'],
            yticklabels=['A+', 'A', 'B', 'C', 'D', 'F'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - AdaBoost")
plt.show()
```

(a) Code

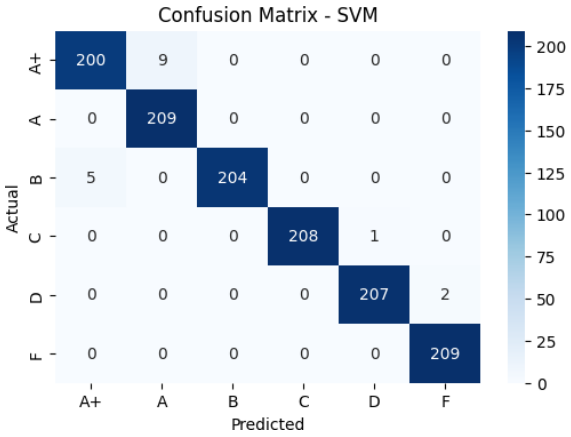


(b) Confusion Matrix

Figure 23: Code and Confusion Matrix for AB Classifier.

```python
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.svm import SVC

# Initialize SVM model
model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X_train_res, y_train_res, cv=cv)

# Confusion Matrix
cm = confusion_matrix(y_train_res, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['A+', 'A', 'B', 'C', 'D', 'F'],
            yticklabels=['A+', 'A', 'B', 'C', 'D', 'F'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - SVM")
plt.show()
```

(a) Code

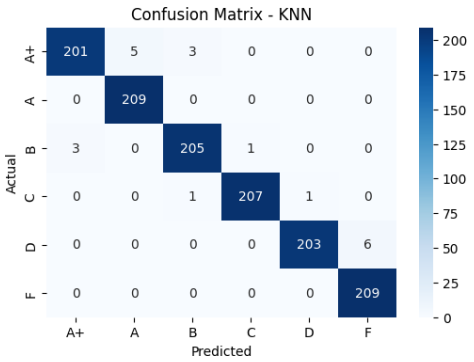


(b) Confusion Matrix

Figure 24: Code and Confusion Matrix for SVC.

```python
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier

# Initialize KNN model
model = KNeighborsClassifier(n_neighbors=5)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X_train_res, y_train_res, cv=cv)
cm = confusion_matrix(y_train_res, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['A+', 'A', 'B', 'C', 'D', 'F'],
            yticklabels=['A+', 'A', 'B', 'C', 'D', 'F'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - KNN")
plt.show()
```

(a) Code



(b) Confusion Matrix

Figure 25: Code and Confusion Matrix for KNN.

Results and Discussion

Table 6 and Figure 26 summarize the performances of the five classifiers. The SVC achieved the highest accuracy (99%), closely followed by KNN (98%) and NB (96%). LR also performed well, with 93% accuracy. In contrast, AB showed poor performance, with only 35% accuracy, indicating weak generalization.

Table 6: Classification Performance Metrics of Different Models

Model	Precision	Recall	F1-score	Accuracy
Logistic Regression	0.93	0.93	0.93	0.93
Naive Bayes Classifier	0 .96	0.96	0.96	0.96
AdaBoost Classifier	0.57	0.36	0.36	0.35
Support Vector Classifier	0 .99	0.99	0.99	0.99
K-Nearest Neighbors Classifier	0.98	0.98	0.98	0.98

Results Comparison

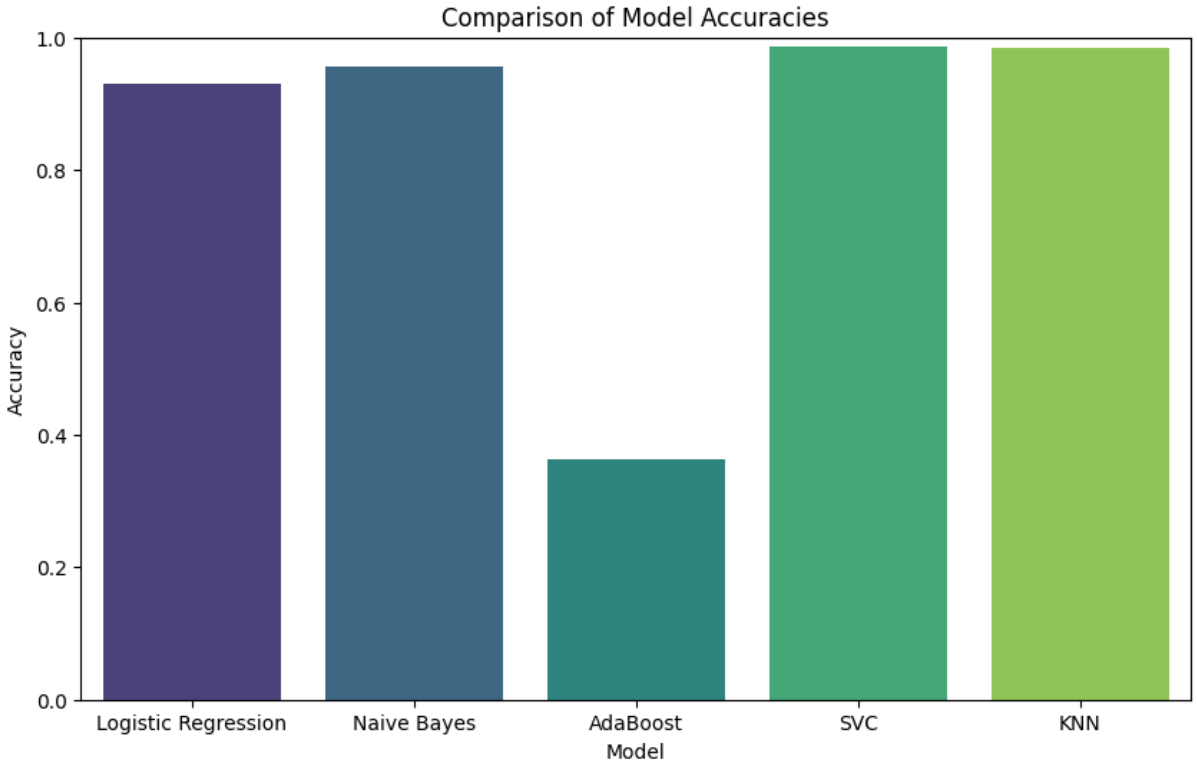


Figure 26: Comparison of Model Accuracies

The Figure 27, 28, and 29 shows the Comparison of recall, precision, and f1-score Across Grades per Model.

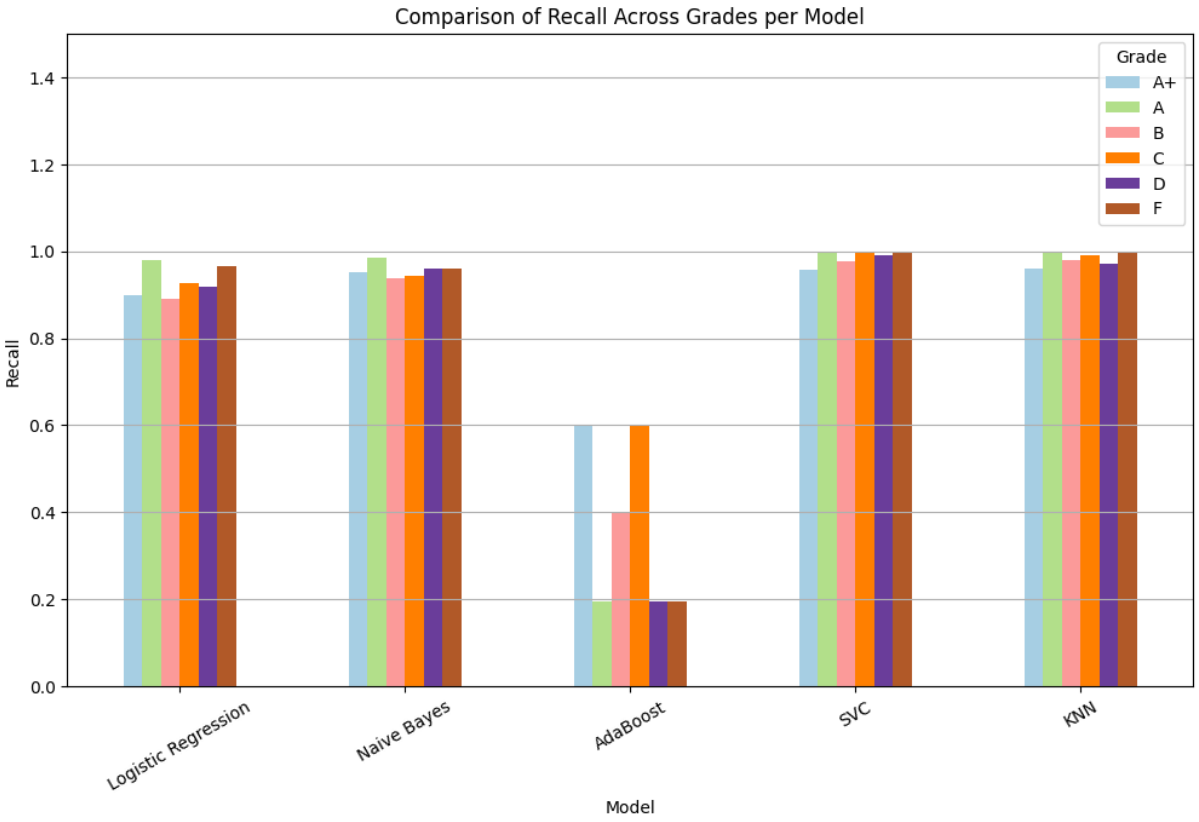


Figure 27: Comparison of Recall Across Grades per Model

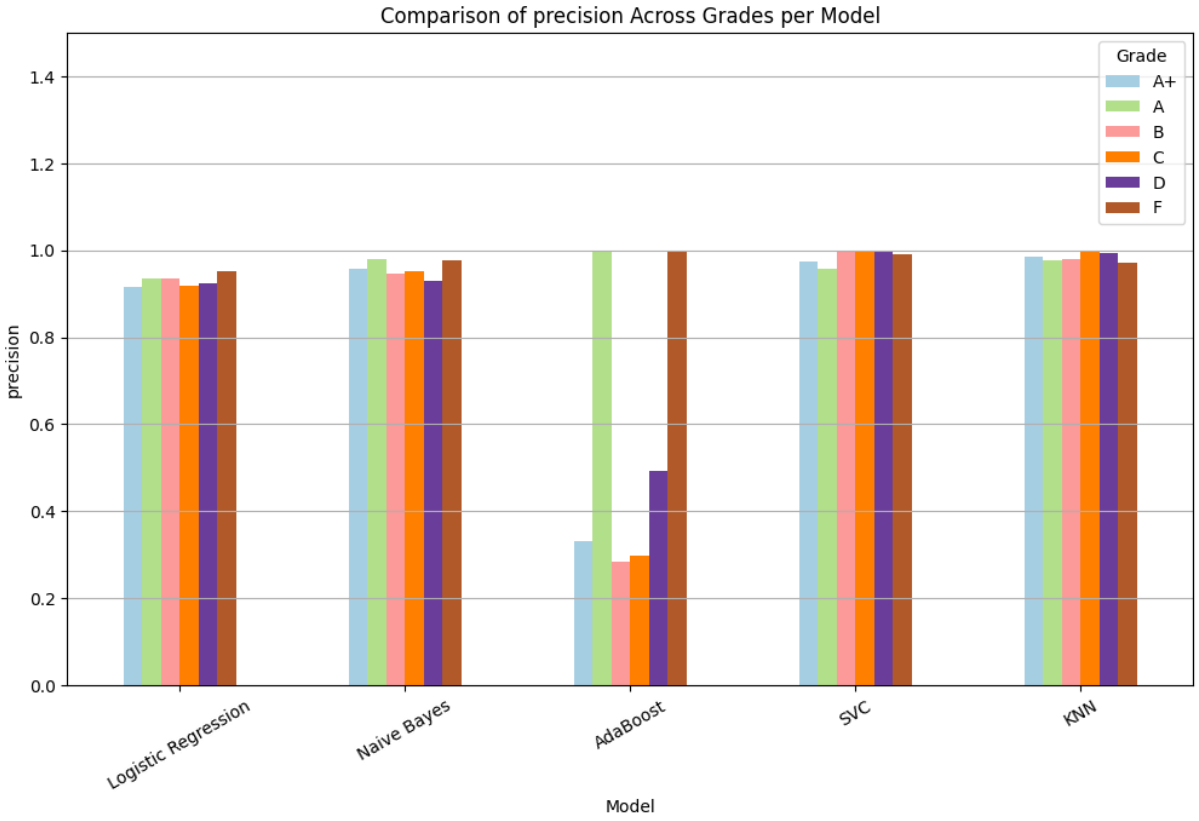


Figure 28: Comparison of precision Across Grades per Model

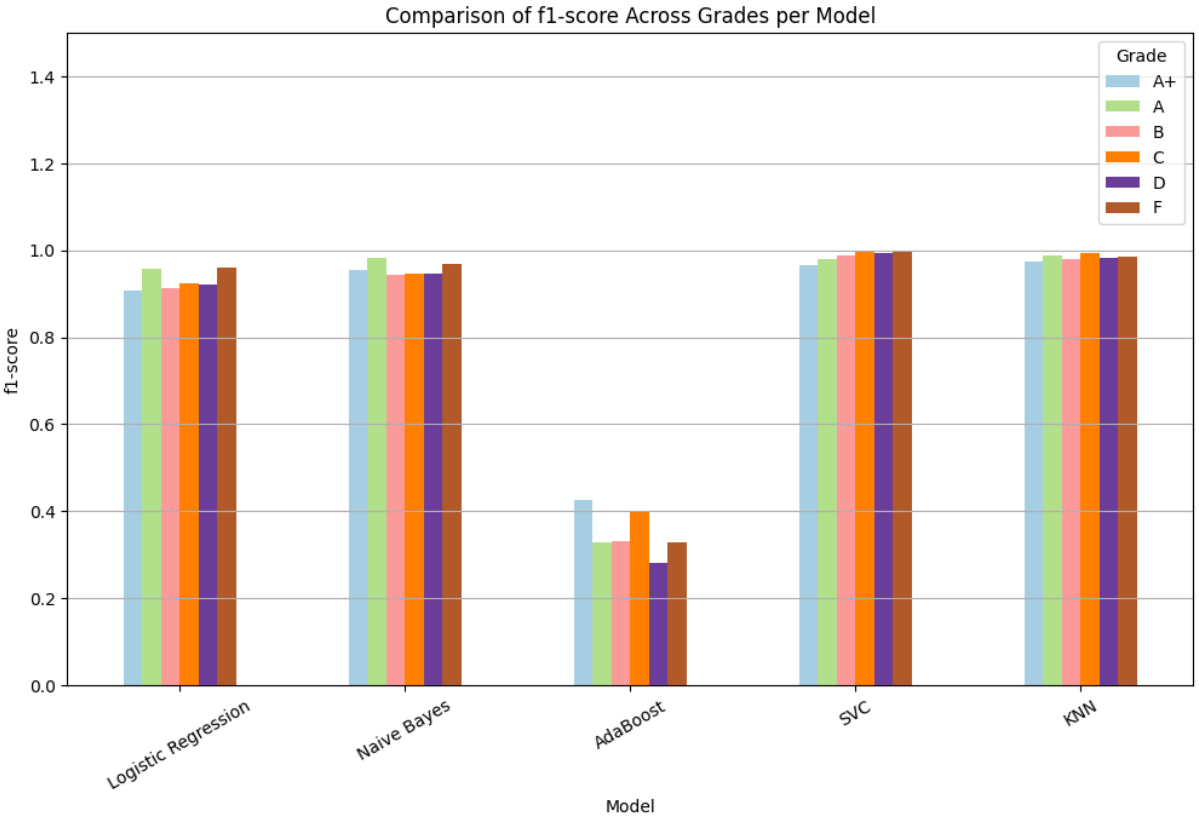


Figure 29: 'Comparison of f1-score Across Grades per Model.

AI Ethics

We applied ethical principles by defining a clear research scope and avoiding harmful, discriminatory, or privacy-invasive objectives. The project aimed to support educators in identifying students who need help and promote fair, data-driven decisions rather than punitive measures.

The dataset was obtained from publicly available Kaggle data with no PII, ensuring privacy compliance. Demographic attributes were anonymized to prevent re-identification of the participants.

To avoid bias, we carefully examined demographic features (gender, race/ethnicity, parental education) and ensured that predictions would not disadvantage specific groups. SMOTE was used to address the class imbalance, and the performance was evaluated across all classes beyond accuracy alone.

Conclusion

This study applied several machine learning models to predict student performance using demographic and examination data. SVC achieved the best results with 99% accuracy, followed by KNN with 98%, NB with 96%, and LR with 93%. In contrast, AB performed poorly with only 35% accuracy. Overall, SVC proved most effective, highlighting the potential of ML models to help educators identify at-risk students and design targeted interventions.

Recommendation

- Adoption of Ensemble Models: Use robust models for accurate student performance prediction.
- Ethical Integration: Apply clear guidelines to ensure fairness and transparency.
- Early Intervention: Integrate predictive tools to flag students who need support.

Future Work

Future research could focus on validating model performance across diverse datasets, exploring deep learning methods such as LSTM and CNN, and applying explainable AI techniques for transparency. Developing real-time prediction systems and implementing fairness-aware algorithms to mitigate demographic biases can further enhance the reliability and ethical use of these models.

1. Baker, R.S. and Inventado, P.S., 2014. Educational data mining and learning analytics. Springer Handbook of Learning Analytics, pp.61–75.

2. Breiman, L., 2001. Random forests. Machine Learning, 45(1), pp.5–32.

3. Chen, T. and Guestrin, C., 2016. XGBoost: A scalable tree-boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp.785–794.

4. Hastie, T., Tibshirani, R. and Friedman, J., 2009. The Elements of Statistical Learning: Data Mining, Inference and Prediction. 2nd ed. New York: Springer.

5. Kotsiantis, S.B., Pierrakeas, C. and Pintelas, P., 2010. Predicting students' per- formance in distance learning using machine learning techniques. Applied Artificial Intelligence, 18(5), pp.411–426.

6. Romero, C. and Ventura, S., 2010. Educational data mining: A review of the state of the art. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 40(6), pp.601–618.

7. Tinto, V., 1993. Leaving College: Rethinking the Causes and Cures of Student Attrition. 2nd ed. Chicago: University of Chicago Press.

8. Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority oversampling technique. Journal of Artificial Intelligence Research, 16, pp.321–357.

9. Kuh, G.D., Kinzie, J., Buckley, J.A., Bridges, B.K. and Hayek, J.C., 2008. What Matters to Student Success: A Review of the Literature. Washington, DC: National Postsecondary Education Cooperative.

10. Siemens, G. and Baker, R.S.J.d., 2013. Learning analytics and educational data mining: Towards communication and collaboration. Springer, New York.

11. Montgomery, D.C., Peck, E.A. and Vining, G.G., 2012. Introduction to Linear Regression Analysis. 5th ed. Hoboken, NJ: John Wiley & Sons.

12. Speiser, J.L., Miller, M.E., Tooze, J. and Ip, E., 2019. A comparison of random forest variable selection methods for classification prediction modeling. Expert Systems with Applications, 134, pp.93–101.

13. Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., 1984. Classification and Regression Trees. Belmont, CA: Wadsworth International Group.

14. Cortes, C. and Vapnik, V., 1995. Support-vector networks. Machine Learning, 20(3), pp.273–297.

15. Cover, T. and Hart, P., 1967. Nearest-neighbor pattern classification. IEEE Trans- actions on Information Theory, 13(1), pp.21–27.

16. Cox, D.R., 1958. Regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological), 20(2), pp.215–232.