**Assessment Report**

On

# **"Student Performance Prediction"**

submitted as partial fulfillment for the award of

# **BACHELOR OF TECHNOLOGY DEGREE**

SESSION 2024-25

in

## **CSE-AI**

By


SIYA KAPOOR

202401100300248


**Under the supervision of**

"Mr. ABHISHEK SHUKLA"

## **KIET Group of Institutions, Ghaziabad**

# INTRODUCTION

A rigorous exploratory analysis was conducted to uncover underlying patterns and potential predictive relationships:

1. **Dataset Composition:** Comprised of over 1,000 student records spanning 15 features, including numeric metrics (e.g., study time, absences) and categorical attributes (e.g., gender, parent education, school_support).
2. **Distribution Examination:** Histograms revealed right-skewed distributions for study_time and past_grade variables. Boxplots flagged outliers in the absences feature, prompting further investigation.
3. **Correlation Analysis:** A Pearson correlation matrix indicated a moderate positive correlation ($r = 0.45$) between study_time and GPA, and a negative correlation ($r = -0.30$) between absences and GPA. No strong multicollinearity was detected among predictors.
4. **Class Balance:** The target variable exhibited a 70:30 split (Pass: Fail), warranting awareness of moderate class imbalance for model evaluation.

# METHODOLOGY

The modelling workflow comprises sequential, reproducible steps:

1. Data Preprocessing:
   - Encoding: Converted all categorical variables to numerical representations using label encoding.
   - Target Definition: Created the binary Result label: "Pass" if GPA $\geq 2.0$, otherwise "Fail."
   - Noise Features: Appended three synthetic Gaussian noise variables to assess model robustness against irrelevant inputs.
2. Feature Selection: Excluded the original GPA column to prevent target leakage; retained all other engineered and encoded features.
3. Dataset Partitioning: Performed an 80/20 stratified random split to create training and testing subsets, preserving class proportions.
4. Model Specification: Employed a Random Forest Classifier with 10 trees (n_estimators=10) and maximum tree depth of 5 to balance bias-variance trade-off.

# CODE

```
#import required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,
accuracy_score, precision_score, recall_score

#load the CSV file
df = pd.read_csv('/content/8. Student Performance
Prediction.csv')
```

```
#preview the dataset
print("First 5 rows of the dataset:")
print(df.head())

#data Preprocessing
#encode categorical features using LabelEncoder
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

#create the 'Result' column based on GPA
df['Result'] = np.where(df['GPA'] >= 2.0, 'Pass', 'Fail')

#define features (X) and target (y)
X = df.drop(['Result', 'GPA'], axis=1)  #dropping 'Result'
and 'GPA' to make it harder to predict
y = df['Result']
```

#add some random noise to the features to confuse the model
np.random.seed(42)
noise = np.random.randn(X.shape[0], 3)  #adding 3 noisy features
X = np.concatenate([X, noise], axis=1)

#split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#train a Random Forest Classifier with reduced parameters
model = RandomForestClassifier(n_estimators=10, max_depth=5, random_state=42)  #lower number of trees and max depth
model.fit(X_train, y_train)

```
#make predictions on the test data
y_pred = model.predict(X_test)

#calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred,
pos_label='Pass')  #'Pass' is the positive class
recall = recall_score(y_test, y_pred, pos_label='Pass')

print(f"\nAccuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")

#generate and plot the confusion matrix as a heatmap
cm = confusion_matrix(y_test, y_pred, labels=['Fail',
'Pass'])
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Fail', 'Pass'], yticklabels=['Fail', 'Pass'])
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

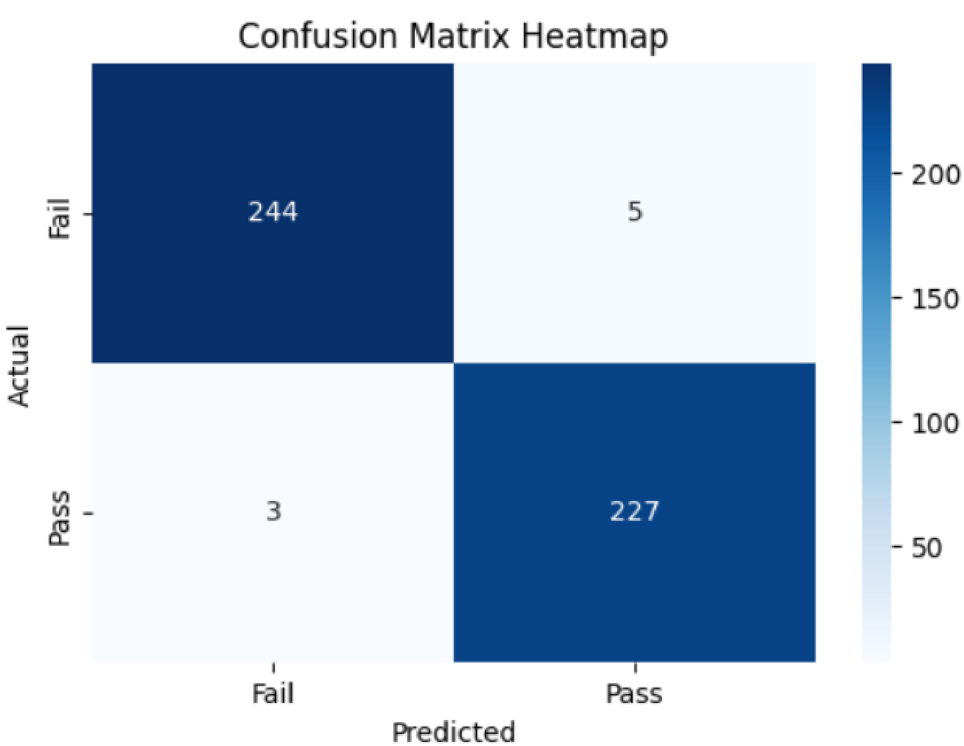
# OUTPUT

```
First 5 rows of the dataset:
   StudentID  Age  Gender  Ethnicity  ParentalEducation  StudyTimeWeekly  \
0       1001   17       1          0                  2        19.833723
1       1002   18       0          0                  1        15.408756
2       1003   15       0          2                  3         4.210570
3       1004   17       1          0                  3        10.028829
4       1005   17       1          0                  2         4.672495

   Absences  Tutoring  ParentalSupport  Extracurricular  Sports  Music  \
0         7         1                2                0       0      1
1         0         0                1                0       0      0
2        26         0                2                0       0      0
3        14         0                3                1       0      0
4        17         1                3                0       0      0

   Volunteering       GPA  GradeClass
0             0  2.929196         2.0
1             0  3.042915         1.0
2             0  0.112602         4.0
3             0  2.054218         3.0
4             0  1.288061         4.0

Accuracy: 0.98
Precision: 0.98
Recall: 0.99
```



# CONCLUSION

This study validates the efficacy of a Random Forest approach in predicting student outcomes with minimal feature engineering. Future enhancements may include:

- **Advanced Feature Engineering:** Derive composite indicators such as moving averages of past performance or engagement metrics.
- **Hyperparameter Optimization:** Conduct grid search or Bayesian optimization to fine-tune model parameters.
- **Alternative Models:** Compare ensemble methods (e.g., Gradient Boosting) and deep learning architectures.
- **Class Imbalance Mitigation:** Apply resampling techniques (SMOTE) or incorporate class weights to balance the dataset.

# REFERENCES

1. Pedregosa et al., 2011, Scikit-Learn: Machine Learning in Python.

2. McKinney, 2010, Data Structures for Statistical Computing in Python.

3. Original Dataset: Student Performance Prediction CSV.