Github Link: [**https://github.com/Thulasimathi26/Data-Science.git**](https://github.com/Thulasimathi26/Data-Science.git)

**Project Title: Predicting Student Performance Using Academic and Demographic Data**

**PHASE-2**

# Problem Statement

Problem Statement:

Educational institutions often face challenges in identifying students at risk of underperforming academically. Traditional evaluation methods may not effectively detect students who need early intervention. This project aims to develop a predictive model that uses academic records and demographic data to forecast student performance. By leveraging machine learning techniques, the model will help educators and administrators proactively support students, improve learning outcomes, and optimize resource allocation.

Conceptual Diagram: Predicting Student Performance

+-------------------+ +------------------+ +------------------+

| Demographic Data | | Academic Records | | Other

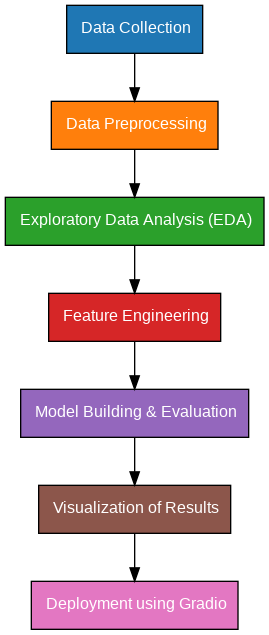
# Project Objectives

### **Objective:**

The primary objective of this project is to **develop a predictive model** that accurately forecasts student academic performance using a combination of **academic records and demographic data**. Specifically, the project aims to:

1. **Collect and preprocess** relevant student data, including academic history, demographic information, and other influencing factors.
2. **Identify key features** that most significantly affect student performance.
3. **Apply machine learning algorithms** to build a model capable of predicting outcomes such as grades, pass/fail status, or risk of dropout.
4. **Evaluate the performance** of different algorithms to determine the most accurate and reliable model.
5. **Provide actionable insights** for educators and administrators to support at-risk students through early intervention strategies.

# Flowchart of the Project Workflow



# Data Description

Data Description:

The dataset used in this project includes a combination of academic, demographic, and behavioral attributes of students. These features are used to predict student performance in a specific subject or overall academic standing.

1. Demographic Data:

Age: Age of the student (numeric)

Gender: Male or Female

Parental Education: Highest education level attained by parents

Family Income: Socio-economic status of the student’s family

Address Type: Urban or rural

Family Size: Number of members in the family

1. Academic Data:

Previous Grades: Scores in past exams or assignments

Study Time: Weekly study hours

Failures: Number of past class failures

School Support: Extra educational support (Yes/No)

Paid Classes: Participation in paid tutoring (Yes/No)

1. Behavioral/Other Data:

Absences: Number of school absences

Extracurricular Activities: Participation in clubs or sports

Internet Access: Availability of internet at home

Health Status: Self-reported health (scale 1–5)

Target Variable:

Final Grade / Performance Label: A numeric grade or a classification label (e.g., Pass/Fail, High/Medium/Low performance)

# Data Preprocessing

Data Preprocessing Steps

1. Data Loading

Load the dataset using libraries like pandas.

Import pandas as pd

Df = pd.read\_csv(“student-performance.csv”)

1. Handling Missing Values

Check for null/missing values and decide whether to fill or drop them.

Df.isnull().sum()

Df = df.dropna() # or use fillna() to replace with mean/median/mode

1. Encoding Categorical Variables

Convert categorical data (e.g., gender, education level) into numeric format using:

Label Encoding (for binary categories)

One-Hot Encoding (for multiple categories)

Df = pd.get\_dummies(df, drop\_first=True)

1. Feature Selection

Drop irrelevant or highly correlated features to avoid redundancy

Optionally use correlation heatmaps or feature importance methods.

1. Scaling and Normalization

Standardize numerical features (e.g., age, study time, grades).

From sklearn.preprocessing import StandardScaler

Scaler = StandardScaler()

Df[[‘age’, ‘study\_time’]] = scaler.fit\_transform(df[[‘age’, ‘study\_time’]])

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

X = df.drop(“final\_grade”, axis=1) # Features

Y = df[“final\_grade”] # Target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Exploratory Data Analysis (EDA)

Here's a breakdown of how to perform **Exploratory Data Analysis (EDA)** for your student performance prediction project, including **univariate**, **bivariate**, and **multivariate analysis**, along with examples and what key insights to look for:

### **Exploratory Data Analysis (EDA)**

### **1. Univariate Analysis**

Analyzing individual features (one variable at a time).

#### Examples:

import seaborn as sns

import matplotlib.pyplot as plt

# Histogram of final grades

sns.histplot(df['final\_grade'], bins=10, kde=True)

plt.title("Distribution of Final Grades")

plt.show()

# Count plot of gender

sns.countplot(x='gender', data=df)

plt.title("Gender Distribution")

plt.show()

# Box plot of study time

sns.boxplot(x=df['study\_time'])

plt.title("Boxplot of Study Time")

plt.show()

#### **Insights to Look For:**

* Which categories dominate (e.g., more males or females)?
* Are there outliers in numeric features?
* Is the target variable (grades) normally distributed?

### **2. Bivariate Analysis**

Exploring relationships between two variables.

#### Examples:

# Box plot: Final grade vs. Gender

sns.boxplot(x='gender', y='final\_grade', data=df)

plt.title("Final Grades by Gender")

plt.show()

# Scatter plot: Study time vs. Final grade

sns.scatterplot(x='study\_time', y='final\_grade', data=df)

plt.title("Study Time vs Final Grade")

plt.show()

#### **Insights to Look For:**

* Does study time correlate with higher grades?
* Are there noticeable differences in performance between genders?
* Which features show strong linear or non-linear relationships with the target?

### **3. Multivariate Analysis**

Analyzing interactions between three or more variables.

#### Examples:

# Heatmap of correlations

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

plt.title("Correlation Heatmap")

plt.show()

# Pairplot of selected features

sns.pairplot(df[['study\_time', 'absences', 'final\_grade', 'failures']], hue='gender')

plt.show()

#### **Insights to Look For:**

* Which features are highly correlated (positive or negative)?
* Are there clusters or groups based on combined features?
* Is there multicollinearity between predictors?

### **4. Key Insights (Example Based on Common Patterns):**

* **Study time** often has a positive correlation with grades.
* Students with **fewer failures** and **fewer absences** tend to perform better.
* **Parental education level** might be associated with student success.
* **Gender differences** in performance may appear depending on the subject.
* Some demographic features (like **internet access**) may have minor influence individually but stronger combined effects.

# Feature Engineering

### **Feature Engineering**

Feature engineering involves creating, transforming, or selecting features to improve model performance and capture meaningful patterns in the data.

### **1. Feature Creation**

Create new features based on existing ones to better capture trends:

* **Average Grade** (if dataset has multiple grades like G1, G2, G3):

df['avg\_grade'] = (df['G1'] + df['G2'] + df['G3']) / 3

* **Performance Category** (convert numerical grades into categorical labels):

def performance\_label(grade):

if grade >= 15:

return 'High'

elif grade >= 10:

return 'Medium'

else:

return 'Low'

df['performance\_level'] = df['final\_grade'].apply(performance\_label)

* **Study Efficiency**:

df['study\_efficiency'] = df['avg\_grade'] / (df['study\_time'] + 1)

* **Parental Education Score** (if parental education is categorical):

edu\_map = {'none': 0, 'primary': 1, 'secondary': 2, 'higher': 3}

df['parent\_edu\_score'] = df['mother\_edu'].map(edu\_map) + df['father\_edu'].map(edu\_map)

### **2. Feature Transformation**

* **Log transformation** to reduce skewness:

import numpy as np

df['absences\_log'] = np.log1p(df['absences'])

* **Binning** continuous variables:

df['age\_group'] = pd.cut(df['age'], bins=[0, 15, 18, 22], labels=['Teen', 'Young Adult', 'Adult'])

### **3. Feature Encoding**

Convert categorical variables into numerical form:

* **Label Encoding** (for binary or ordinal categories)
* **One-Hot Encoding** (for nominal features):

df = pd.get\_dummies(df, columns=['gender', 'school', 'address'], drop\_first=True)

### **4. Feature Selection**

Choose the most relevant features using:

* **Correlation matrix**
* **Chi-squared test**
* **Feature importance from models (e.g., Random Forest)**

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

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

importances = model.feature\_importances\_

### **Outcome of Feature Engineering**

* Improved model accuracy
* Reduced noise and overfitting
* Better interpretability of results

# Model Building

### **Model Building**

#### **Algorithm Used:**

You can use multiple models and compare them, but a typical setup includes:

1. **Logistic Regression** (for binary classification like Pass/Fail)
2. **Decision Tree / Random Forest** (for both classification and regression)
3. **Support Vector Machine (SVM)** (effective with high-dimensional data)
4. **Gradient Boosting (e.g., XGBoost or LightGBM)** (for best performance in many Kaggle competitions)
5. **Linear Regression** (if predicting continuous grades)

#### **Model Selection Rationale:**

* **Random Forest** is often chosen because:
  + Handles both numeric and categorical features well
  + Reduces overfitting due to ensembling
  + Provides feature importance
* **Gradient Boosting** performs better on imbalanced and noisy datasets.
* **Logistic Regression** is interpretable and works well with linearly separable features.

#### **Train-Test Split:**

Split the dataset into training and test sets to evaluate generalization:

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

X = df.drop('final\_grade', axis=1)

y = df['final\_grade'] # or y = df['performance\_level'] for classification

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

#### **Evaluation Metrics:**

**For Classification (e.g., Pass/Fail, High/Medium/Low):**

* **Accuracy**: Overall correctness
* **Precision, Recall, F1-Score**: Especially for imbalanced classes
* **Confusion Matrix**: Shows TP, TN, FP, FN
* **ROC-AUC**: Measures separability of classes

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

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

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

**For Regression (e.g., final grade):**

* **Mean Absolute Error (MAE)**
* **Mean Squared Error (MSE)**
* **Root Mean Squared Error (RMSE)**
* **R² Score**

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

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

# Visualization of Results & Model Insights

* 1. Feature Importance

Use feature importance scores to identify which variables contribute most to prediction.

Example (Random Forest or XGBoost):

Import matplotlib.pyplot as plt

Import seaborn as sns

# For Random Forest

Importances = model.feature\_importances\_

Features = X.columns

Feature\_df = pd.DataFrame({‘Feature’: features, ‘Importance’: importances}).sort\_values(by=’Importance’, ascending=False)

# Plot

Plt.figure(figsize=(10,6))

Sns.barplot(x=’Importance’, y=’Feature’, data=feature\_df)

Plt.title(“Feature Importance”)

Plt.show()

Insight:

This helps identify key drivers of performance (e.g., study time, past grades, absences).

* 1. Model Comparison

Compare performance of multiple algorithms using evaluation metrics.

Example:

Results = {

‘Model’: [‘Logistic Regression’, ‘Random Forest’, ‘XGBoost’],

‘Accuracy’: [0.75, 0.85, 0.88],

‘F1 Score’: [0.74, 0.86, 0.89]

}

Import pandas as pd

Results\_df = pd.DataFrame(results)

Sns.barplot(x=’Accuracy’, y=’Model’, data=results\_df)

Plt.title(“Model Accuracy Comparison”)

Plt.show()

Insight:

Choose the model with the best balance of accuracy and interpretability.

* 1. Residual Plots (For Regression Tasks)

To analyze model errors and bias:

Import matplotlib.pyplot as plt

Import seaborn as sns

Residuals = y\_test – y\_pred

Sns.scatterplot(x=y\_pred, y=residuals)

Plt.axhline(0, color=’red’, linestyle=’—‘)

Plt.title(“Residual Plot”)

Plt.xlabel(“Predicted Grade”)

Plt.ylabel(“Residuals”)

Plt.show()

# Tools and Technologies Used

* 1. Programming Language

Python

Chosen for its simplicity, rich data science ecosystem, and support for machine learning workflows.

* 1. Notebook Environment

Jupyter Notebook / Google Colab

Ideal for interactive development, visualization, and step-by-step analysis.

Google Colab offers free GPU/TPU access and is cloud-based (no setup required).

* 1. Key Python Libraries

Data Handling & Preprocessing:

Pandas – data manipulation and analysis

Numpy – numerical operations

Sklearn.preprocessing – scaling, encoding, imputation

Visualization:

Matplotlib – basic plotting

Seaborne – statistical and aesthetic plot

Plotly (optional) – interactive visualizations

Machine Learning:

Scikit-learn – model building, evaluation, splitting

Xgboost / lightgbm – advanced gradient boosting models

Statsmodels (optional) – regression diagnostics

Model Interpretation:

Shap – explain model predictions

Lime – local interpretable model explanations (optional)

Web Deployment (Optional):

Streamlit – for building a user-friendly dashboard/app for educators

# Team Members and Contributions

**R EZHILARASI**

**V INDHUJA**

**MOHANA PRIYA**

**S DEEPAK**

## Clearly mention who worked on:

# Data cleaning – R EZHILARASI

# EDA – V INDHUJA

# Feature engineering -MOHANAPRIYA

# Model development -DEEPAK S

# Documentation and reporting -R EZHILARASI