Student Performance Predictor — Project Report

Project Title: Student Performance Predictor (SPP)

Author: Paras Dwivedi

---

1. Cover Page

Project: Student Performance Predictor (SPP)

Course: AI/ML (B.Tech)

Institution: VIT

Submitted By: Paras Dwivedi

Reg. No : 25BAI10621

---

2. Introduction

Students and educators can benefit from early, data-driven estimates of academic outcomes. This project builds a lightweight machine learning system that predicts a student's final exam score (or categorical performance) using easily collectible features such as study hours, attendance, previous scores, and assignment completion.

---

3. Problem Statement

Many students are unsure whether their current habits will yield the desired results. Teachers lack a compact tool to forecast outcomes early in a term. The SPP aims to predict final performance and highlight key influencing factors so interventions can be made timely.

---

4. Objectives

- Build a reliable ML model to predict student exam performance.
- Provide a simple interface to input student features and obtain predictions.
- Visualize feature importance and model metrics.
- Create well-documented code and a reproducible GitHub repository.

---

5. Functional Requirements

1. **Data Input Module**
    - Accept CSV dataset uploads and manual input via CLI.
    - Validate formats and missing values.

2. **Data Processing Module**
    - Handle missing values, encode categorical features, scale/normalize numeric features.
    - Feature selection and engineering.
3. **Model Training & Evaluation Module**
    - Train models (Linear Regression, Random Forest, Decision Tree).
    - Evaluate using appropriate metrics (MAE/MSE/R^2 for regression; Accuracy/F1 for classification).
4. **Prediction Module**
    - Accept a new student record and return predicted score or category.
5. **Reporting & Visualization Module**
    - Plot distributions, correlations, feature importance, and model performance charts.
6. **Repository & Documentation**
    - README.md, statement.md, source code, and project report in PDF.

---

6. Non-Functional Requirements

- **Usability:** Simple CLI; clear error messages and instructions.
- **Performance:** Prediction result should be produced in under 1 second for single record inference.
- **Maintainability:** Modular codebase with separate files for preprocessing, modeling, and utilities.
- **Reliability:** Robust input validation and exception handling.
- **Scalability:** Able to handle increased dataset sizes without major code changes.

---

7. System Architecture

- **Data Layer:** CSV or Pandas DataFrame containing features and target.
- **Preprocessing Layer:** Cleaning, encoding, scaling.
- **Model Layer:** Training, cross-validation, and persistence (joblib/pickle).
- **Interface Layer:** CLI script for user interaction; future upgrade path to web UI.

Simple flow:

User → Input Module → Preprocessor → Model Trainer → Persisted Model → Predictor → Output

---

8. Process Flow / Workflow

1. Load dataset (CSV) or sample dataset provided.
2. Explore and visualize data.
3. Clean and preprocess features.
4. Split into train/test (80/20) and optionally k-fold cross-validation.
5. Train candidate models and compare metrics.
6. Select best model and save it.
7. Use saved model to predict for new user inputs.
8. Generate visual reports and feature importance.

---

9. UML & Design Artefacts (to include as images or diagrams in final PDF)

- **Use Case Diagram:** Actors: Student/User, System. Use cases: Upload Data, Train Model, Make Prediction, View Report.
- **Class Diagram:** DataLoader, Preprocessor, ModelTrainer, Predictor, Visualizer.
- **Sequence Diagram:** User → CLI → Preprocessor → Trainer → Model → Predictor → User.

(Placeholders for diagrams — include exported images in docs/diagrams/ in the GitHub repo.)

---

10. Dataset Description

Sample features (suggested):

- student_id (optional)
- attendance_pct (numeric: 0–100)
- study_hours_per_week (numeric)
- assignment_avg (numeric: 0–100)
- midterm_score (numeric)
- sleep_hours (numeric)
- social_media_hours (numeric)
- participation (categorical: High/Medium/Low)

Target: final_score (numeric) or performance_label (categorical: Below Average / Average / Above Average)

Data Source: Synthetic data generated for demonstration, or public student performance datasets (if used, cite source).

---

11. Model Selection & Rationale

- **Linear Regression:** Baseline for numeric prediction — fast and interpretable.
- **Random Forest Regressor / Classifier:** Handles non-linear patterns, robust to outliers, and provides feature importance.
- **Decision Tree:** Simple to visualize and explain to stakeholders.

Choose based on metric performance and interpretability trade-offs.

---

12. Evaluation Methodology

- **Train/Test split:** 80/20
- **Metrics for regression:** Mean Absolute Error (MAE), Mean Squared Error (MSE), R^2 score.
- **Metrics for classification:** Accuracy, Precision, Recall, F1-score, Confusion Matrix.
- **Cross-Validation:** 5-fold CV recommended for robust metric estimates.

---

13. Implementation Details

- **Language:** Python 3.10+
- **Libraries:** pandas, numpy, scikit-learn, matplotlib, seaborn (optional for visuals), joblib
- **Project structure (recommended):**

```
SPP/
├── data/
│   └── student_data.csv
├── docs/
│   └── diagrams/
├── spp/
│   ├── __init__.py
│   ├── data_loader.py
│   ├── preprocessor.py
│   ├── trainer.py
│   ├── predictor.py
│   └── visualizer.py
├── tests/
```

├── README.md

├── statement.md

└── requirements.txt

- **Model persistence:** Save the trained model with joblib.dump(model, 'model.joblib').
- **Entry point:** predict_cli.py accepts manual inputs (or path to CSV) and prints predictions.

---

14. Testing Approach

- **Unit tests** for data validation, preprocessing steps, and metric calculations.
- **Integration test**: Full pipeline run on sample dataset to ensure end-to-end correctness.
- **Manual tests**: Try edge-case inputs and missing values to ensure graceful handling.

---

15. Challenges & Risk Mitigation

- **Small dataset:** Use cross-validation and synthetic data augmentation.
- **Data quality:** Robust cleaning and clearly documented assumptions.
- **Overfitting:** Use regularization, pruning, and cross-validation.

---

16. Learnings & Key Takeaways

- Importance of feature engineering for predictive power.
- Trade-offs between model complexity and interpretability.
- Practical experience with model evaluation and reproducibility.

---

17. Future Enhancements

- Web UI (Flask/FastAPI + simple frontend) for non-technical users.
- Dashboard with real-time predictions and cohort analytics.
- Use of time-series or sequence models if temporal data available.

---

18. References

- VIT Project Guidelines: BuildYourOwnProject.pdf
- Scikit-learn documentation: https://scikit-learn.org/
