MINI PROJECT REPORT – CA2

The Student Performance Predictor Project using Ensemble Learning and AI Integration via API Endpoints

Subject: DMBI

Submitted By:

Atharva Lotankar (27)

Nupur Ghangarekar (33)

Vaishnavi Avhad (41)

D15C

Submitted To:

Ms. Pradnya Karekar

Vivekanand Education Society's Institute of Technology, Chembur – 400 071

2025-2026

Introduction

The Student Performance Predictor Project leverages artificial intelligence to forecast academic success using demographic and test score data. Its primary objective is to offer rapid, actionable insights for educators and students by analysing key performance indicators and background factors.

Background and Overview

Academic performance prediction is significant for guiding interventions in education. This project uses data from a real-world dataset of student records with features like gender, ethnicity, parental education, lunch type, test preparation, and exam scores. Predictive models are trained to classify students as "Pass" or "Fail" based on their average scores, facilitating early interventions and personalized guidance.

Tech Stack Used

- **Backend**: Python 3, Flask (server and API endpoints)
- **Data Processing**: pandas, numpy (data manipulation), scikit-learn (ML), TensorFlow/Keras (deep learning)
- **ML Models**: Decision Tree, Random Forest, Neural Network (Keras Sequential API)
- **Frontend**: HTML5, CSS3 (custom styles, responsive design), JavaScript (fetch API, dynamic result updates)
- **Integration**: Groq API (external AI-powered insights)
- **Deployment**: Localhost, scalable for cloud environments.

Methodology

The solution is built on a Python Flask web app:

- The Dataset is extracted as StudentsPerformance.csv from https://www.kaggle.com/datasets/spscientist/students-performance-in-exams
- The dataset is loaded and pre-processed—categorical features are encoded and scores are normalized.
- Labels are created by averaging math, reading, and writing scores—students are classified as “Pass” if their average score is ≥ 60, otherwise “Fail”.
- The data is split into training and test sets.
- Multiple machine learning models (Decision Tree, Random Forest, Neural Network) are trained for classification.
- Key metrics (accuracy, precision, recall, F1-score) are computed for comparison.
- Ensemble prediction is provided using majority voting among trained models.

Solution: Code Snippets / Algorithm

Step 1: Data Preprocessing and Label Encoding

```
# Load and preprocess data

df = pd.read_csv('StudentsPerformance.csv')

label_encoders = {}

categorical_columns = ['gender',
'race/ethnicity', 'parental level of
education', 'lunch', 'test preparation
course']



for col in categorical_columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le



# Create target label based on average score

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

df['performance'] = (df['average_score'] >=
60).astype(int)
```

Step 2: Feature Scaling and Train-Test Split

```
# Prepare features and target

X = df.drop(['performance', 'average_score'],
axis=1)

y = df['performance']



# Scale features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)



# Split data

X_train, X_test, y_train, y_test =
train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
```

Step 3: Model Training (Decision Tree, Random Forest, Neural Network)

```
# Decision Tree

dt_model =
DecisionTreeClassifier(max_depth=5,
random_state=42)

dt_model.fit(X_train, y_train)



# Random Forest

rf_model =
RandomForestClassifier(n_estimators=100,
max_depth=10, random_state=42, n_jobs=-1)

rf_model.fit(X_train, y_train)



# Neural Network

nn_model = Sequential([

    Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),

    BatchNormalization(),

    Dropout(0.3),

    Dense(64, activation='relu'),

    BatchNormalization(),

    Dropout(0.3),

    Dense(32, activation='relu'),

    Dropout(0.2),

    Dense(1, activation='sigmoid')

])

nn_model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])

early_stop =
EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)

nn_model.fit(X_train, y_train, epochs=100,
batch_size=32, validation_split=0.2,
callbacks=[early_stop], verbose=0)
```

Step 4: Prediction and Ensemble Logic

```
# Ensemble prediction using majority voting

def predict_ensemble(model_dt, model_rf,
model_nn, scaler, input_data):

    input_scaled =
scaler.transform(input_data)

    pred_dt =
model_dt.predict(input_scaled)[0]

    pred_rf =
model_rf.predict(input_scaled)[0]

    pred_nn =
(model_nn.predict(input_scaled)[0][0] >
0.5).astype(int)
```

```
    # Majority voting

    vote = np.round((pred_dt + pred_rf +
pred_nn) / 3)

    return int(vote), {"Decision Tree":
int(pred_dt), "Random Forest": int(pred_rf),
"Neural Network": int(pred_nn)}
```

Step 4: Groq API Integration for AI Insights

```
# Generate AI-powered insights using Groq API

def get_groq_insights(prompt):

    headers = {"Authorization": f"Bearer
{GROQ_API_KEY}", "Content-Type":
"application/json"}

    data = {"model": "mixtral-8x7b-32768",
"messages": [{"role": "user", "content":
prompt}]}

    response = requests.post(GROQ_API_URL,
headers=headers, json=data)

    return response.json().get("choices",
[{}])[0].get("message", {}).get("content",
"No insight generated.")
```

Code Logic Analysis

- **Data preprocessing**: Label encoding for categorical features, score normalization for neural network input.
- **Model training**: Three separately optimized models, including memory-efficient settings, early stopping for neural networks.
- **API and web logic**: User submits demographic and score data via a form; Python backend processes and predicts, returning JSON.
- **Results and explanations**: Feature importance is calculated for interpretability. Results are shown for all models and ensemble outcome, with confidence metrics.
- **Groq API Integration**: Two key usages:
    1. *Performance Analysis*: The backend sends model metrics (accuracy, F1-score, etc.) to Groq's Llama API for automated, expert-level assessment that appears in the UI's metrics analysis section.
    2. *Personalized Insights*: After a prediction, user profile, feature importance, and prediction outcome are sent to Groq API, generating natural-language feedback and actionable, positive advice tailored to each student. This "AI Insight" is displayed beside the scores.

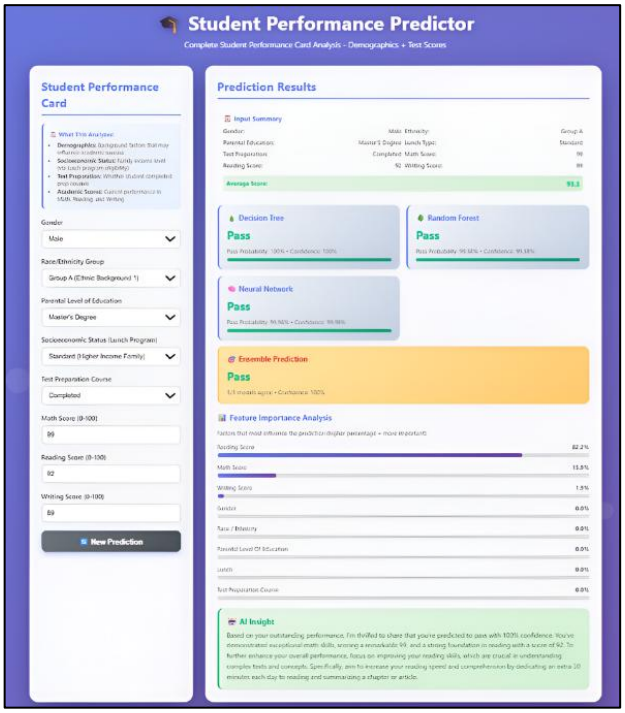
Challenges

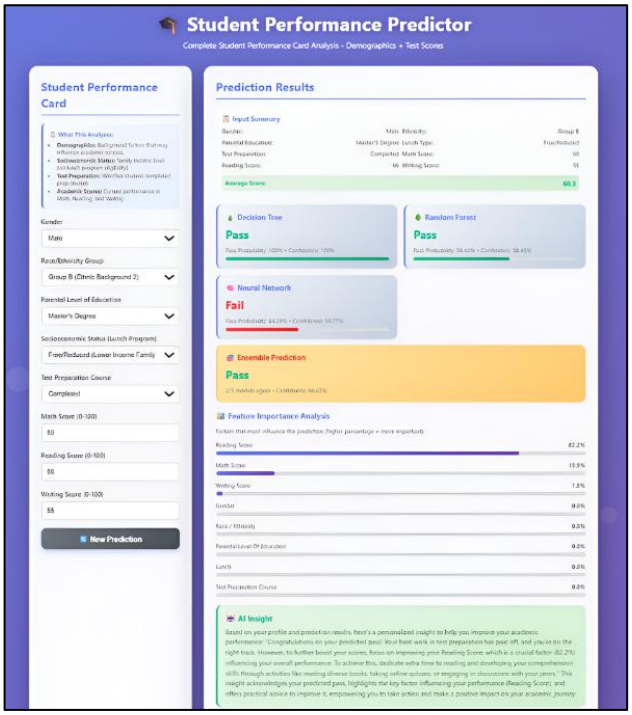
This project faced challenges such as handling diverse categorical and numerical data, ensuring effective feature encoding and scaling, and balancing model complexity to avoid underfitting or overfitting. Achieving high prediction accuracy while maintaining model interpretability was also a key concern, alongside managing computational resources efficiently for training multiple models.

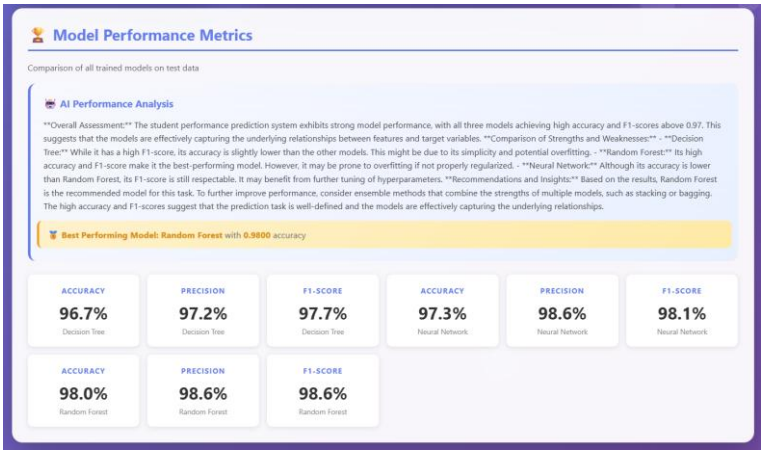
Goal

The primary goal was to build a reliable, AI-powered system capable of accurately predicting student academic performance from demographic and test score data. The project aimed to identify key influencing factors, enable early intervention by educators, and provide personalized, actionable insights to improve student outcomes.

Results: Output of Mini Project







Future Outcome

Future enhancements could include integrating additional data sources like attendance and behavioural logs, experimenting with deeper learning architectures, and expanding the system for multi-class performance levels or continuous score prediction. The project can evolve into a comprehensive educational analytics platform supporting adaptive learning and targeted student support in real time.

Conclusion

The project uses data mining techniques like preprocessing, label encoding, and scaling to prepare student data. Supervised machine learning models—Decision Tree, Random Forest, and Neural Networks—are trained to classify student performance. An ensemble voting method combines predictions to enhance accuracy and reliability.